Novell Nsure Identity Manager Driver for JDBC*

2.0

www.novell.com

IMPLEMENTATION GUIDE

May 3, 2006

Novell®

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.

www.novell.com

Identity Manager Driver for JDBC: Implementation Guide
May 3, 2006

Online Documentation: To access the online documentation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Contents

About This Guide **9**

1 Introducing the Identity Manager Driver for JDBC **11**

Overview . . . 11
New Features . . . 11
    Driver Features . . . 11
    Identity Manager New Features . . . 12
Driver Concepts . . . 12
    JDBC . . . 12
    Identity Manager Driver for JDBC . . . 12
    Third-Party JDBC Driver . . . 13
    Identity Vault . . . 13
    Directory Schema . . . 13
    Application Schema . . . 13
    Database Schema . . . 14
    Synchronization Schema . . . 14
    Logical Database Class . . . 14
    XDS . . . 14
Database Concepts . . . 14
    Structured Query Language . . . 14
    Data Manipulation Language . . . 14
    Data Definition Language . . . 15
    View . . . 15
    Identity Columns/Sequences . . . 15
    Transaction . . . 16
    Stored Procedures or Functions . . . 16
    Trigger . . . 17
    Instead-Of-Trigger . . . 17
Data Synchronization Models . . . 18
    Indirect Synchronization . . . 19
    Direct Synchronization . . . 20
Triggerless Publication . . . 21
    Triggerless vs. Triggered Publication . . . 21
    Direct Triggerless Publication . . . 22
    Indirect Triggerless Publication . . . 22

2 Understanding Driver Prerequisites **23**

Driver Prerequisites . . . 23
    Supported Platforms . . . 23
    Supported Databases . . . 23
    Supported Third-Party JDBC Drivers . . . 23
    Known Issues . . . 23
    Limitations . . . 24

3 Installing or Upgrading the Driver for JDBC **25**

Installing . . . 25

Identity Manager-Side Installation 25
Remote Loader Installation 27
Database-Side Installation 27
MySQL Installation 30
Upgrading 32
Upgrading from Versions Earlier than 1.5 32
Upgrading from 1.5 or Later to 2.0 32
Backward Incompatibilities 32
Activating 33

4 Configuring the Driver for JDBC 35

Smart Configuration 35
Configuration Parameters 37
Deprecated Parameters 38
Authentication Parameters 38
Authentication ID 38
Authentication Context 38
Application Password 38
Driver Parameters 39
Uncategorized Parameters 40
Database Scoping Parameters 42
Connectivity Parameters 44
Compatibility Parameters 46
Subscription Parameters 54
Uncategorized Parameters 54
Primary Key Parameters 56
Publication Parameters 61
Uncategorized Parameters 61
Triggered Publication Parameters 63
Triggerless Publication Parameters 65
Polling Parameters 65
Trace Levels 67
Configuring Third-Party JDBC Drivers 68

5 Advanced Configuration 69

Schema Mapping 69
Logical Database Classes 69
Indirect Synchronization 69
Direct Synchronization 76
Synchronizing Primary Key Columns 79
Synchronizing Multiple Classes 79
Mapping Multivalue Attributes to Single-Value Database Fields 79
XDS Event to SQL Statement Mapping 80
The Event Log Table 81
Event Log Columns 82
Event Types 84
Embedding SQL Statements in XDS Events 89
Variable Substitution 90
Statement Placement 91
Manual vs. Automatic Transactions 92
Transaction Isolation Level 93
Statement Type 94
SQL Queries 95
Data Definition Language (DDL) Statements 96
Logical Operations 96
Best Practices 97

6 Third-Party JDBC Drivers 99

Third-Party JDBC Driver Interoperability . . . 99
JDBC Drivers: Four Types . . . 99
Which Type To Use? . . . 100
Supported Third-Party JDBC Drivers . . . 100
Third-Party JDBC Driver Features . . . 100
JDBC URL Syntaxes . . . 101
JDBC Driver Class Names . . . 101
BEA Weblogic jDriver for Microsoft SQL Server . . . 102
IBM DB2 Universal Database JDBC Drivers . . . 103
Informix JDBC Driver . . . 105
Microsoft SQL Server 2000 Driver for JDBC . . . 106
MySQL Connector/J JDBC Driver. . . . 107
Oracle Thin Client JDBC Drivers . . . 108
PostgreSQL JDBC Driver . . . 109
Sybase Adaptive Server Enterprise JConnect JDBC Driver . . . 110
Using Unsupported Third-Party JDBC Drivers . . . 111
Security Issues . . . 111

7 Supported Databases 113

Database Interoperability . . . 113
Supported Databases . . . 113
Database Characteristics . . . 114
Database Features . . . 114
Current Time Stamp Statements . . . 115
Stored Procedure and Function JDBC Call Syntaxes. . . . 115
Left Outer Join Operators . . . 115
Undelimited Identifier Case-Sensitivity . . . 116
Supported Transaction Isolation Levels. . . . 116
Commit Keywords. . . . 117
IBM DB2 Universal Database (UDB) . . . 117
Informix Dynamic Server (IDS) . . . 118
Microsoft SQL Server . . . 118
MySQL . . . 119
Oracle . . . 120
PosgreSQL . . . 120
Sybase Adaptive Server Enterprise (ASE) . . . 121

8 Using the Association Utility 123

Understanding the Association Utility . . . 123
Before You Begin . . . 124
Using the Association Utility . . . 124
Editing Associations. . . . 125

9 Uninstalling the IDM Driver for JDBC 127

Deleting IDM Driver Objects . . . 127
Running the Product Uninstaller . . . 127
Executing Database Uninstallation Scripts . . . 127
IBM DB2 Universal Database (UDB) Uninstallation. . . . 128
Informix Dynamic Server (IDS) Uninstallation. . . . 128
Microsoft SQL Server Uninstallation . . . 128
MySQL Uninstallation . . . 128
Oracle Uninstallation . . . 129
PostgreSQL Uninstallation . . . 129
Sybase Adaptive Server Enterprise (ASE) Uninstallation. . . . 129

A **Best Practices** **131**

B **FAQ** **133**

Why can't the driver see my tables or views? 133
How do I synchronize with tables located in multiple schemas? 133
Why isn't the driver processing rows in the event log table? 134
Can the driver manage database user accounts? 134
Can the driver synchronize large binary and string data types? 134
Why is publication so slow? 134
Can the driver synchronize multiple classes? 134
Does the driver support encrypted transport? 135
How do I map multivalue attributes to single-value database fields? 135
Why is the driver synchronizing garbage strings? 135

C **Supported Data Types** **137**

D **java.sql.DatabaseMetaData Methods** **139**

E **Utilized JDBC Methods** **141**

F **Third-Party JDBC Driver Descriptor DTD** **147**

G **Third-Party JDBC Driver Descriptor Import DTD** **149**

H **Database Descriptor DTD** **151**

I **Database Descriptor Import DTD** **153**

J **Documentation Updates** **155**

May 3, 2006 155

About This Guide

The Nsure™ Identity Manager Driver for Java* Database Connectivity (JDBC) provides a generic solution for synchronizing data between an Identity Vault and relational databases.

This guide provides an overview of the driver's technology as well as configuration instructions.

Additional Documentation

For documentation on using Identity Manager and the other drivers, see the Identity Manager Documentation Web site (http://www.novell.com/documentation/lg/dirxmldrivers).

Documentation Updates

For the most recent version of this document, see the Identity Manager Documentation Web site (http://www.novell.com/documentation/lg/dirxmldrivers/index.html).

Documentation Conventions

In this documentation, a greater-than symbol (>) is used to separate actions within a step and items within a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell® trademark. An asterisk (*) denotes a third-party trademark.

User Comments

We want to hear your comments and suggestions about this manual and the other documentation included with Novell Identity Manager. Please use the User Comment feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

1 Introducing the Identity Manager Driver for JDBC

The Nsure™ Identity Manager Driver for Java DataBase Connectivity (JDBC), subsequently referred to as the driver, 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.

Overview

In this section, you will find information on the following topics:

- "New Features" on page 11
- "Driver Concepts" on page 12
- "Database Concepts" on page 14
- "Data Synchronization Models" on page 18
- "Triggerless Publication" on page 21

New Features

This section includes information about the driver's new features.

Driver Features

- Publication without triggers. See "Triggerless Publication" on page 21.
- Future event processing. See "Enable future event processing?" on page 64.
- Daily publication. See "Publication Time of Day" on page 65.
- Batch processing. See "Batch Size" on page 66.
- Expanded database support. See "Supported Databases" on page 113.
- Enhanced support for database time types. See "Time Syntax" on page 40.
- Enhanced ease of use. See "Smart Configuration" on page 35.
- Schema filtering. See "Include Filter Expression" on page 43 and "Exclude Filter Expression" on page 44.
- Extended view support. See "Direct Synchronization" on page 76.

- Enhanced support for third-party driver encryption mechanisms. See "Connection Properties" on page 45.
- Password modify and check support.
- Improved driver configuration/database SQL* scripts.

Identity Manager New Features

For more information on the new features in Identity Manager, refer to the Nsure Identity Manager Administration Guide (http://www.novell.com/documentation/dirxml20/index.html).

Driver Concepts

The following are some important terms and concepts you should know before installing and configuring the driver:

- "JDBC" on page 12
- "Identity Manager Driver for JDBC" on page 12
- "Third-Party JDBC Driver" on page 13
- "Identity Vault" on page 13
- "Directory Schema" on page 13
- "Application Schema" on page 13
- "Database Schema" on page 14
- "Synchronization Schema" on page 14
- "Logical Database Class" on page 14
- "XDS" on page 14

JDBC

An acronym that stands for Java DataBase Connectivity. JDBC is a cross-platform database interface standard developed by Sun* Microsystems*. Most enterprise database vendors provide a unique implementation of the JDBC interface.

Three 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)

The Identity Manager Driver for JDBC primarily uses the JDBC 1 interface. It will use more advanced features when supported by third-party JDBC drivers.

Identity Manager Driver for JDBC

An Identity Manager driver that uses the JDBC interface to synchronize data and identities between eDirectory and relational databases. The driver consists of four jar files:

- `JDBCShim.jar`

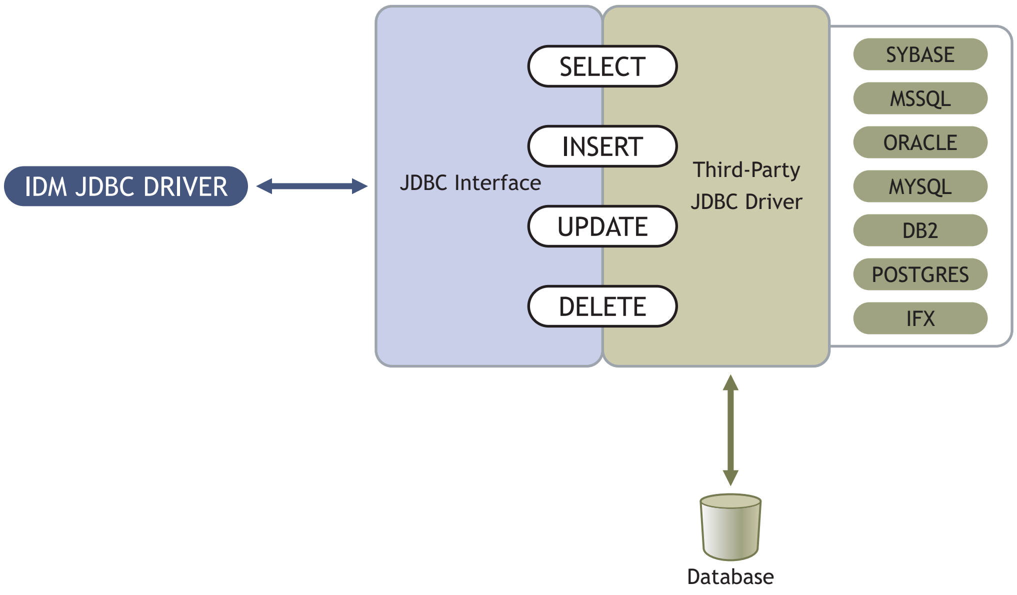
- `JDBCUtil.jar`
- `JDBCConfig.jar`
- `CommonDriverShim.jar`

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

Third-Party JDBC Driver

One of the numerous JDBC interface implementations that the Driver for JDBC uses to communicate with a particular database. For example, classes12.zip 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 Driver for JDBC and third-party JDBC drivers.



Identity Vault

The datastore (Novell® eDirectory™) that Identity Manager uses.

Directory Schema

The set of object classes and attributes in the directory. For example, the eDirectory User class and Given Name attribute are part of the eDirectory schema.

Application Schema

The set of classes and attributes in an application. Because databases have no concept of classes or attributes, the driver maps eDirectory classes to tables or views and maps eDirectory attributes to columns.

Database Schema

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. In the context of the Identity Manager Driver for JDBC, schema is useful for database scoping (reducing the number of database objects visible to the driver at runtime).

Ownership is often expressed by using a qualified dot notation such as `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 database schema visible to the driver at runtime.

Logical Database Class

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

XDS

A generic, extensible, portable data format that supports XML

Database Concepts

- "Structured Query Language" on page 14
- "Data Manipulation Language" on page 14
- "Data Definition Language" on page 15
- "View" on page 15
- "Identity Columns/Sequences" on page 15
- "Transaction" on page 16
- "Stored Procedures or Functions" on page 16
- "Trigger" on page 17
- "Instead-Of-Trigger" on page 17

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 Identity Manager Driver for JDBC is essentially 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 Identity Manager Driver for JDBC is DML-based, you can embed DDL statements in XDS events. For additional information, refer to "Embedding SQL Statements in XDS Events" on page 89.

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 the implementation guide are for the Oracle database.

View

A logical table. When queried via a `SELECT` statement, the view is constituted 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.

```
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.

An identity column is a self-incrementing column used to uniquely identify a row in a table. Identity columns 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

Manual transactions usually contain more than one statement. DDL statements typically cannot be grouped with DML statements in a manual transaction. The following example shows 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. When a statement runs automatically, it is autonomous of any other statement. The following example shows 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 for the purpose of creating 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 is an example of 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 fires or executes under certain conditions. Triggers are often useful for creating side effects in a database. In the context of this 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 would be committed or rolled back together.

Instead-Of-Trigger

An instead-of-trigger is programmatic logic associated with a view, which fires or executes under certain conditions. Instead-of-triggers are useful for making views writable/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 fires if the statement satisfies the conditions specified in the trigger. Unlike triggers, instead-of-triggers always fire 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:

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

Instead-of-trigger `t_view_usr_i` would fire and execute the following statement:

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

instead of the original insert statement.

In this example, the statements happen to be equivalent.

Data Synchronization Models

The 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.

Direct synchronization is usually associated with views because views provide the abstraction mechanism that best facilitates integration with existing customer tables.

Indirect synchronization is usually associated with tables because customer tables likely don't match the structure required by the driver. Therefore, they can serve as an intermediate staging area only. Although it is possible that the structures might match, it is highly unlikely.

For all practical purposes, this means that

- Direct synchronization = view

- Indirect synchronization = table

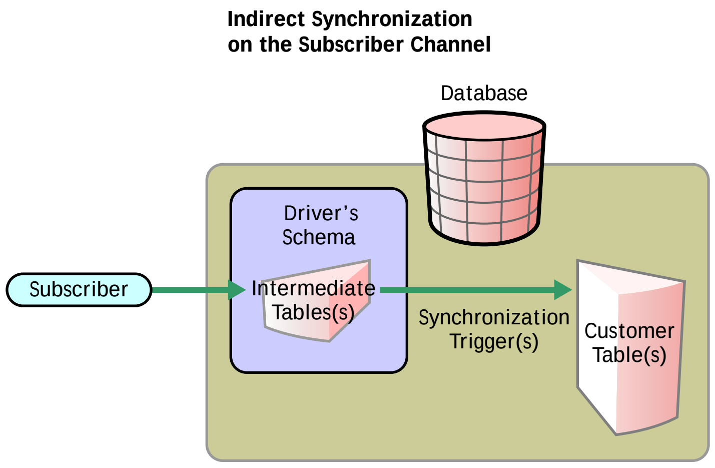
The following sections describe how direct and indirect synchronization work on both the Subscriber and Publisher channels.

Indirect Synchronization

Indirect synchronization uses intermediate staging tables to synchronize data between an Identity Vault and a database.

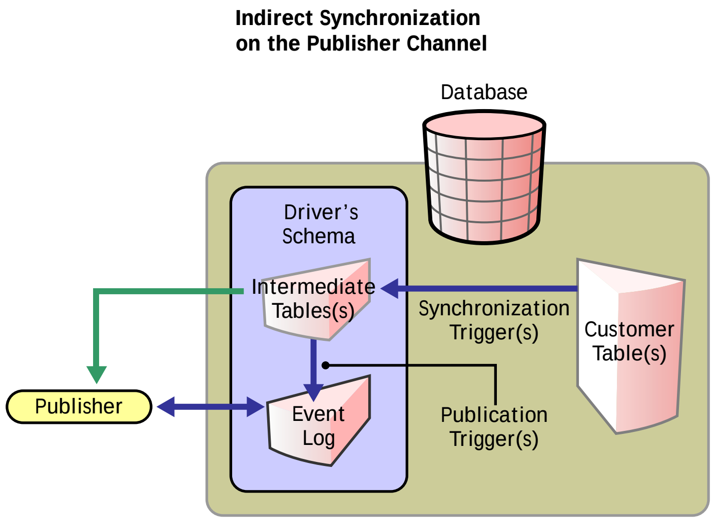
The following diagrams illustrate how indirect synchronization works on the Subscriber and Publisher channels. In the following scenarios, you can have one or more customer tables and intermediate staging tables.

Subscriber Channel



The Subscriber channel updates the intermediate staging tables in the synchronization schema. The synchronization triggers then update customer tables elsewhere in the database.

Publisher Channel



When customer tables are updated, synchronization triggers update the intermediate staging tables. Publication triggers then insert one or more rows into the event log table. The Publisher then reads the inserted rows and updates the Identity Vault.

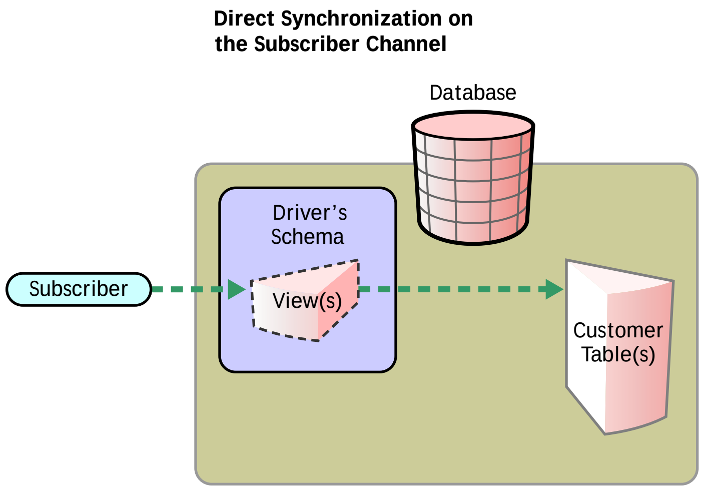
Depending on the contents of the rows read from the event log table, the Publisher channel might need to retrieve additional information from the intermediate tables before updating the Identity Vault. After updating the Identity Vault, the Publisher channel then deletes or marks the rows as processed.

Direct Synchronization

Direct synchronization typically uses views to synchronize data between Identity Manager and a database. Tables can be used if they conform to the structure required by the driver.

The following diagrams illustrate how direct synchronization works on the Subscriber and Publisher channels. In the following scenarios, you can have one or more customer tables or views.

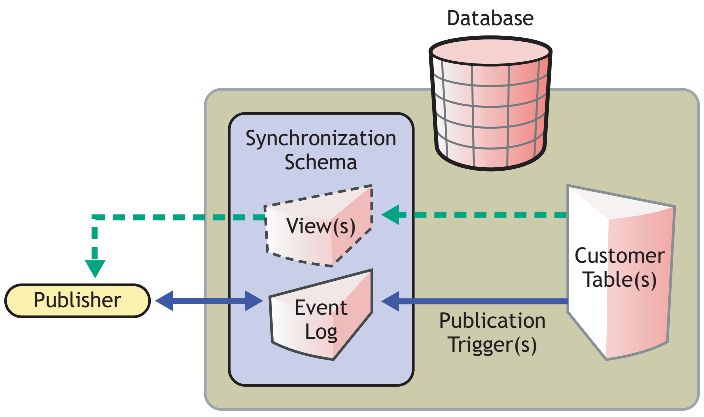
Subscriber Channel



The Subscriber channel updates existing customer tables through a view in the synchronization schema.

NOTE: Direct synchronization without a view is only possible if customer tables match the structure that the driver requires. For additional information, refer to "Indirect Synchronization" on page 69.

Publisher Channel



When a customer 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 Publication

Triggers are no longer required to log publication events. 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.

Triggerless vs. Triggered Publication

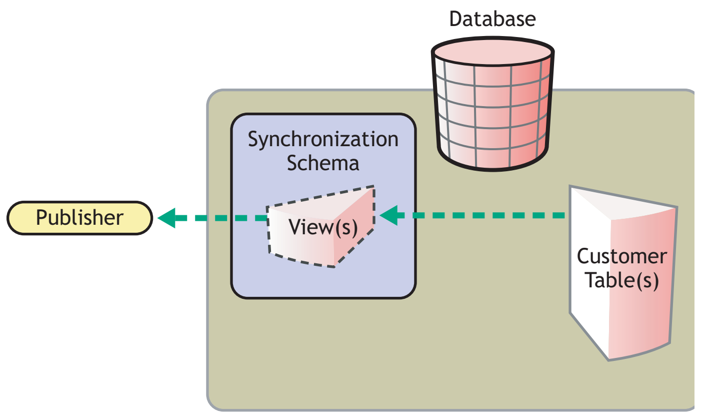
Triggerless publication is less efficient than triggered publication. With triggered publication, what changed is already known. With triggerless publication, change calculation must occur before events can be processed.

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 will be in sync.

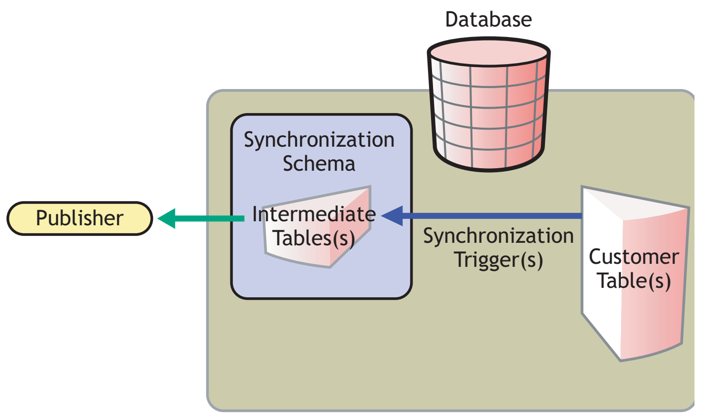
Triggerless publication, unlike triggered publication, does not provide historical data such as old values. It provides information on the current state of an object only, not the previous state.

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.

Direct Triggerless Publication



Indirect Triggerless Publication



2 Understanding Driver Prerequisites

- "Driver Prerequisites" on page 23
- "Supported Platforms" on page 23
- "Supported Databases" on page 23
- "Supported Third-Party JDBC Drivers" on page 23
- "Known Issues" on page 23
- "Limitations" on page 24

Driver Prerequisites

The Identity Manager Driver for JDBC requires the following:

- ❑ Novell iManager
- ❑ Novell Nsure Identity Manager 2
- ❑ Java Virtual Machine (JVM*) 1.4 or higher
- ❑ A supported third-party JDBC driver

Supported Platforms

The driver runs on all Identity Manager-enabled platforms, including Windows* NT*/2000, NetWare®, Solaris*, Linux*, and AIX*.

Supported Databases

Refer to "Database Interoperability" on page 113.

Supported Third-Party JDBC Drivers

Refer to "Third-Party JDBC Driver Interoperability" on page 99.

Known Issues

- eDirectory™ 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. Refer to "Time Syntax" on page 40 for more information.

- The driver for JDBC 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 driver for JDBC, by default, assumes time stamp values placed in the event log table are in ODBC canonical format (that is, `yyyy-mm-dd hh:mm:ss.ffffffff`).

  The recommended method for enabling the driver for JDBC to handle proprietary database time stamp formats is to implement a custom `DBTimestampTranslator` class. This interface is documented in the javadocs that ship with the driver. Using this approach avoids the problem of reformatting time stamps in the database before they are inserted into the event log table or reformatting them in style sheets. The driver for JDBC 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.

  Blocking can be caused by a myriad of factors. To mitigate the likelihood of blocking, we recommend that you do not set the parameter "Transaction Isolation Level" on page 48 to a level greater than `read committed`.

  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 50. The driver for JDBC ships with a default implementation for Oracle.

  The JDBC interface defines a method that allows a statement to timeout after a specified number of seconds, signature java.sql.Statement.setQueryTimeout(int):void (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html). 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.

Limitations

- The driver does not support the use of delimited (quoted) database identifiers.
- 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.

3 Installing or Upgrading the Driver for JDBC

- "Installing" on page 25
- "Upgrading" on page 32
- "Activating" on page 33

For information on uninstalling the driver, see Chapter 9, "Uninstalling the IDM Driver for JDBC," on page 127

IMPORTANT: We recommend installing or uninstalling driver configurations and database scripts as a unit. To prevent unintentional mismatching, database scripts and driver configurations contain headers with a version number, the target database name, and the database version.

Installing

The Identity Manager Driver for JDBC requires Identity Manager and database-side configuration. Identity Manager-side configuration consists of importing a driver configuration file. Database-side configuration consists of executing SQL scripts. We recommend that you execute database SQL scripts and test them before starting the driver.

The shipping configuration is a sample only. We recommend that you install the shipping configuration into a test environment before attempting to customize the configuration.

Identity Manager-Side Installation

- "Installing the Driver for JDBC" on page 25
- "Importing the Sample Driver Configuration File" on page 26

Installing the Driver for JDBC

If no previous installation exists for the driver, download the stand-alone installer and complete the following instructions:

1 Run the installer.

2 Copy the appropriate third-party JDBC driver files into the following directory by platform:

Platform	Directory Path
NetWare®	sys:\system\lib
Solaris, Linux, or AIX	/usr/lib/dirxml/classes
Windows NT/2000	Novell\NDS\lib

For information on third-party JDBC driver filenames and where to download them, refer to "Supported Third-Party JDBC Drivers" on page 100.

3 Restart eDirectory.

4 Start iManager.

Importing the Sample Driver Configuration File

The `JDBCv2.xml` configuration file creates and configures the Identity Manager objects needed for the sample driver to work properly. The configuration file also includes sample policies that you can customize.

1 In iManager, select DirXML Management > Create Driver.

2 Select a driver set, then click Next.

If you place this driver in a new driver set, specify a driver set name, context, and associated server.

3 Select Import a Driver Configuration from the Server (.XML File).

4 From the drop-down list, select the `JDBCv2.xml` file, then click Next.

The driver configuration file is installed on the Web server when you set up iManager.

5 When prompted to enter a name for the driver, specify the driver's name (for example, JDBC 2), then click Next.

6 Select the target database, select whether the driver is local or remote, then click Next.

7 Select a synchronization model, select a third-party JDBC implementation, then click Next.

8 Select a data flow (for example, bidirectional), specify a database host IP address, enter a port number, then click Next.

9 Specify the User container DN, the Group container DN, and the publication mode, then click Next.

10 (Optional) Click Define Security Equivalences.

10a Click Add, then select an object with Admin rights (or any other rights that you want the driver to have).

10b Click Apply, then click OK.

11 (Optional) To exclude objects from replication, click Exclude Administrative Roles.

11a Click Add, then select any users you want to exclude (such as the admin user).

11b Click Apply, then click OK.

12 To view the import summary, click Next.

13 Verify that the configuration is correct, then click Finish with Overview.

The installation has created the necessary Identity Manager driver objects. If you didn't define security equivalences or exclude administrative users at import time, you can complete these tasks by modifying the driver object's properties.

Configuration File Conventions

- Database usernames are the Surname of a User concatenated with the corresponding numeric primary key value (for example, John Doe's username could be Doe1).

- Initial passwords are the Surname of a User (for example, John Doe's password would be `Doe`).

  Sybase* passwords must be at least 6 characters long. When shorter than 6 characters, last names are padded with the character 'p' (for example, John Doe's password would be `Doeppp`). The padding character can be adjusted in the Subscriber Command Transformation policies.

Remote Loader Installation

If you want to run the driver remotely, you must install the Remote Loader on the target server. For more information, see Setting Up the Remote Loader in the *Identity Manager Administration* guide (http://www.novell.com/documentation/dirxml20/admin/data/bs35pip.html#bs35piq).

In the Remote Driver Configuration parameters, set the Driver parameter to

```
com.novell.nds.dirxml.driver.jdbc.JDBCDriverShim
```

Database-Side Installation

The following information explains how to install and configure database objects (for example, tables, triggers, and indexes) for synchronization with the sample driver configuration.

SQL scripts are located in the *install-dir*\jdbc\sql\abbreviated-database-name directory.

- "IBM DB2 Universal Database (UDB) Installation" on page 29
- "Informix Dynamic Server (IDS) Installation" on page 29
- "Microsoft SQL Server Installation" on page 30
- "MySQL Installation" on page 30
- "Oracle Installation" on page 30
- "PostgreSQL Installation" on page 31
- "Sybase Adaptive Server Enterprise (ASE) Installation" on page 31

SQL Script Conventions

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. Due to this length restriction, abbreviations are used. The following table summarizes identifier abbreviations and their meaning:

Abbreviation	Interpretation
proc_[1]	stored procedure/function
idx_	index
trg_	trigger
_i	on insert trigger
_u	on update trigger

Abbreviation	Interpretation
_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)

[1] 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 identifies identifier naming conventions for indexes, triggers, stored procedures, functions, and constraints:

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 precision 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, which can cause large gaps in identity column or sequence values.

IBM DB2 Universal Database (UDB) Installation

IMPORTANT: For IBM* DB2, you must manually create operating system user accounts 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 NT operating environment. If you rerun the SQL scripts, you should repeat only Steps 2 through 5.

The directory context for DB2 is *install-dir*\jdbc\sql\db2_udb\install.

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. Copy `idm_db2.jar` to your DB2 server.
3. Change the name of the administrator account name and password and adjust the path to `idm_db2.jar` in the installation scripts.
4. 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 the '\' line continuation character. Later versions of the Command Center don't recognize this character.

5. For versions 8 or later, execute the `2_install_8.sql` script.

   For example:
   ```
   db2 -f 2_install_8.sql
   ```

Informix Dynamic Server (IDS) Installation

IMPORTANT: 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 NT operating environment. If you rerun the SQL scripts, you should repeat only Steps 2 through 4.

The directory context for Informix SQL scripts is *install-dir*\jdbc\sql\informix_ids\install.

1. In Windows NT, 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.
3. Log in to your server as the `informix` user or another user with DBA (database administrator) privileges.

   By default, the password for `informix` is `informix`.

NOTE: 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.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.

Microsoft SQL Server Installation

The directory context for Microsoft* SQL Server scripts is *install-dir*\jdbc\sql\mssql\install.

1 Start a client such as Query Analyzer.

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 7, execute `1_install_7.sql`.

For version 8 (2000), execute `1_install_2k.sql`.

NOTE: The execute hotkey in Query Analyzer is F5.

MySQL Installation

The directory context for MySQL* SQL scripts is *install-dir*\jdbc\sql\mysql\install.

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

TIP: Don't use a semi-colon to terminate this statement.

Oracle Installation

The directory context for Oracle* SQL scripts is *install-dir*\jdbc\sql\oracle\install.

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

By default, the password for SYSTEM is MANAGER.

NOTE: 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
```

PostgreSQL Installation

The directory context for PostgreSQL scripts is *install-dir*\jdbc\sql\postgres\install. The directory context for executing Postgres commands is *postgres-install-dir*/pgsql/bin.

1 Create the database `idm`.

   For example, from the UNIX command line, execute the command createdb:
   ```
   ./createdb idm
   ```

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

   For example, from the UNIX command line, execute the command createlang:
   ```
   ./createlang plpgsql idm
   ```

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

   For example, from the UNIX command line, execute the command psql:
   ```
   ./psql -d idm postgres
   ```

   By default, the Postgres user has no password.

4 From inside psql, execute the script `1_install.sql`.

   For example:
   ```
   idm=# \i 1_install.sql
   ```

5 Update the `pg_hba.conf` file.

   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    255.255.255.255   255.255.255.0     password
   ```

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

Sybase Adaptive Server Enterprise (ASE) Installation

IMPORTANT: Ensure that you have JDBC metadata support installed on the database server. This is usually an issue for versions earlier than 12.5 only.

The directory context for Sybase SQL scripts is *install-dir*\jdbc\sql\sybase_ase\install.

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

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

Database	Test SQL Scripts Location
IBM DB2 Universal Database	*install-dir*\jdbc\sql\db2_udb\test

Database	Test SQL Scripts Location
Informix Dynamic Server	*install-dir*\jdbc\sql\informix_ids\log\test *install-dir*\jdbc\sql\informix_ids\no_log\test Informix ANSI test scripts are located in the log\test subdirectory.
Microsoft SQL Server	*install-dir*\jdbc\sql\mssql\test
MySQL	*install-dir*\jdbc\sql\mysql\test
Oracle	*install-dir*\jdbc\sql\oracle\test
PostgreSQL	*install-dir*\jdbc\sql\postgres\test
Sybase Adaptive Server Enterprise	*install-dir*\jdbc\sql\sybase_ase\test

We recommend that you try the test scripts before starting the sample driver.

Troubleshooting

- Publication events might not be recognized by the publisher unless you explicitly commit changes. For the commit keywords of supported database, see "Commit Keywords" on page 117.
- 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 64.

Upgrading

Upgrading from Versions Earlier than 1.5

For versions earlier than 1.5, you must first upgrade to version 1.5. Refer to the *DirXML Driver 1.5 for JDBC Implementation Guide* (http://www.novell.com/documentation/lg/dirxmldrivers/index.html).

Be sure to use the 2.0 association utility. It supersedes all previous versions.

Upgrading from 1.5 or Later to 2.0

Download the 2.0 driver, then run the stand-alone installer.

Backward Incompatibilities

- The driver now requires a minimum of two database connections for bidirectional synchronization. For additional information, refer to "Use Minimal Number of Connections?" on page 44.
- The driver now returns schema qualifiers (when available) for logical database class names (parent table or view names). This change doesn't affect existing configurations unless class

names are remapped in Schema Mapping policies. If class names are remapped, all references to class names in existing policy need to be schema-qualified.

- Slightly adjust configurations that reference the `com.novell.nds.dirxml.driver.jdbc.util.MappingPolicy` class. Methods in this class no longer edit the source document. Instead, they return node sets that must be copied into the destination document. The sample driver configuration file `JDBCv2.xml` includes examples of how to do this.
- Slightly alter configurations deployed against DB2/AS400 or other legacy databases that do not have a notion of column position. Add and set the Sort Column Names By parameter. See "Sort Column Names By" on page 53 to sort column names by string collation order. The default behavior has been changed to sort column names by hexadecimal value.

Activating

Activate the driver within 90 days of installation. Otherwise, the driver will not run.

For activation information, see Activating Novell Identity Management Products (http://www.novell.com/documentation/lg/dirxml20/admin/data/afbx4oc.html).

4 Configuring the Driver for JDBC

- "Smart Configuration" on page 35
- "Deprecated Parameters" on page 38
- "Authentication Parameters" on page 38
- "Driver Parameters" on page 39
- "Subscription Parameters" on page 54
- "Publication Parameters" on page 61
- "Trace Levels" on page 67
- "Configuring Third-Party JDBC Drivers" on page 68

Smart Configuration

The Identity Manager Driver for JDBC 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. These features alleviate the need for the end user to understand and explicitly set the parameters.

These features are implemented via four types of XML descriptor files that describe a third-party JDBC driver or database to the Driver for JDBC.

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

Reserved Filenames

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 will be issued. These limitations

- 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) that resides in the runtime classpath.

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

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 Driver for JDBC is updated, place custom descriptors in a different jar file.

Precedence

Parameters explicitly specified through a management console, such as iManager, 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 that 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.

Between 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 appendices contain DTDs for all descriptor file types. These DTDs can help you construct custom descriptor files.

Descriptor Type	Appendix
Third-party JDBC driver	Appendix F, "Third-Party JDBC Driver Descriptor DTD," on page 147
Third-party JDBC driver import	Appendix G, "Third-Party JDBC Driver Descriptor Import DTD," on page 149
Database	Appendix H, "Database Descriptor DTD," on page 151
Database import	Appendix I, "Database Descriptor Import DTD," on page 153

Configuration Parameters

- "Authentication Parameters" on page 38
- "Driver Parameters" on page 39
    - "Uncategorized Parameters" on page 40
    - "Database Scoping Parameters" on page 42
    - "Connectivity Parameters" on page 44
    - "Compatibility Parameters" on page 46
- "Subscription Parameters" on page 54
    - "Uncategorized Parameters" on page 54
    - "Primary Key Parameters" on page 56
- "Publication Parameters" on page 61
    - "Uncategorized Parameters" on page 61
    - "Triggered Publication Parameters" on page 63
    - "Triggerless Publication Parameters" on page 65
    - "Polling Parameters" on page 65

Viewing Driver Parameters

1. In iManager, click DirXML Management > Overview.
2. Locate the driver set containing the driver, then click the driver's icon.
3. From the Driver Overview, click the driver object.

   iManager displays the driver's configuration parameters.

Deprecated Parameters

The following parameters have been deprecated since version 1.6:

Tag Name	Justification
connection-tester-class[1]	The driver now dynamically creates a connection tester class at runtime, based upon information in XML descriptor files.
connection-test-stmt[1]	The driver now dynamically creates a connection tester class at runtime, based upon information in XML descriptor files.
reconnect-interval	The reconnect interval is now fixed at 30 seconds on both channels.

[1] These parameters are still operable to ensure backwards compatibility. Their continued use, however, is discouraged.

Authentication Parameters

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

- "Authentication ID" on page 38
- "Authentication Context" on page 38
- "Application Password" on page 38

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*\tools\sql\abbreviated-database-name\install directory.

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 between 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 101.

IMPORTANT: Changing anything in this value other than URL properties will force a resync 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`.

Driver Parameters

The following table is a summary of all driver-level parameters and their properties:

Display Name	Tag Name	Sample Value	Default Value	Required
Third-party JDBC driver class name	jdbc-class	oracle.jdbc.driver.OracleDriver	(none)	yes
Schema name	sync-schema	indirect	(none)	yes[1]
Table/view name(s)	sync-tables	usr	(none)	yes[1]
Include filter expression	include-table-filter	IDM_.*	(none)	no
Exclude filter expression	exclude-table-filter	BIN\$.{22}==\$0	(none)	no
Connection initialization statements	connection-init	USE idm	(none)	no
Time syntax	time-syntax	1 (integer)	1 (integer)	no
State directory	state-dir	. (current directory)	. (current directory)	no
JDBC driver descriptor filename	jdbc-driver-descriptor	ora_client_thin.xml	(none)	no
Database descriptor filename	database-descriptor	ora_10g.xml	(none)	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
Lock statement generator class	lock-generator-class	com.novell.nds.dirxml.driver.jdbc .db.lock.OraLockGenerator	(dynamic[3])	no
Enable referential attribute support?	enable-refs	1 (yes)	1 (yes)	no
Force username case	force-username-case	upper (to upper case)	(none)	no
Left outer-join operator	left-outer-join-operator	(+)	(dynamic[3])	no
Retrieve minimal metadata?	minimal-metadata	0 (no)	(dynamic[3])	no
Use minimal number of connections?	use-single-connection	0 (no)	(dynamic[3])	no
Function return method	function-return-method	result set	(dynamic[3])	no

Display Name	Tag Name	Sample Value	Default Value	Required
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

[1] These parameters are mutually exclusive.
[2] This default is derived dynamically at runtime from descriptor files and database metadata.
[3] These defaults are derived dynamically from descriptor files at runtime.

Driver parameters fall into the following subcategories:

- "Uncategorized Parameters" on page 40
- "Database Scoping Parameters" on page 42
- "Connectivity Parameters" on page 44
- "Compatibility Parameters" on page 46

Uncategorized Parameters

- "Third-party JDBC Driver Class Name" on page 40
- "Time Syntax" on page 40
- "State Directory" on page 41

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:

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 101.

Time Syntax

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

- Return database Time, Date, and Timestamp values as 32-bit signed integers and map them to eDirectory attributes of type Time or Timestamp

  This is the default.

  This option has two problems:

  - eDirectory Time and Timestamp syntaxes cannot express as large a date range as database Date or Timestamp syntaxes (approximately 136 years).
  - eDirectory Time and Timestamp syntaxes are granular to the second. Database Timestamp syntaxes are often granular to the nanosecond.

  The second option overcomes these two limitations.

- Return database Time, Date, and Timestamp values as canonical strings and map them to attributes of type Numeric String.

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

JDBC Data Type	**Canonical String Format:**[1]
java.sql.Time	HHMMSS
java.sql.Date	CCYYMMDD
java.sql.Timestamp	CCYYMMDDHHMMSSNNNNNNNNN

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

These fixed-length formats collate in chronological order on any platform in any locale.

The following table lists the properties of this parameter:

Property	**Value**
Tag Name	time-syntax
Required?	no
Default Value	1 (integer)
Legal Values	1 (integer) 2 (string)

State Directory

The State Directory parameter specifies where a driver instance should store state data. State data is currently used for triggerless publication, although it maybe used to store additional state information in the future.

Each driver instance has two state files. State filenames follow the format jdbc_*driver-instance-guid*.db and jdbc_*driver-instance-guid*.lg. For example, jdbc_bd2a3dd5-d571-4171-a195-28869577b87e.db and jdbc_bd2a3dd5-d571-4171-a195-28869577b87e.lg are state filenames. If you need to manually identify and delete a driver instance’s state files, each driver instance’s GUID is traced on startup. Defunct state files (those belonging to deleted drivers) in the current state directory are deleted each time a driver instance with the same state directory is started.‘

The following table lists the properties of this parameter:

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

This section describes the following driver parameters:

- "Schema Name" on page 42
- "Table/View Name" on page 43
- "Include Filter Expression" on page 43
- "Exclude Filter Expression" on page 44

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.

When using this parameter instead of "Table/View Name" on page 43, you don't need to explicitly schema-qualify other parameters that reference stored procedure, function, or table names unless they reside in a schema other than this schema name. Such names are implicitly schema-qualified by the driver with this schema name. In particular, "Method and Timing (Table-Local)" on page 57 and "Event Log Table Name" on page 63 are affected.

The following table lists the properties of this parameter:

Property	Value
Tag Name	sync-schema
Required?	yes[1]
Case-Sensitive?	See "Undelimited Identifier Case-Sensitivity" on page 116.
Sample Value	indirect
Default Value:	(none)

[1]When this parameter is used, the Table/View Name parameter must be left empty or omitted from a configuration. See "Table/View Name" on page 43.

IMPORTANT: Changing this value forces a resync of all objects when triggerless publication is used.

Table/View Name

The Table/View Name 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 an error 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/ views of which only a few are of interest. Reducing the number of table/view definitions cached by the driver can shorten start-up time as well as reduce runtime memory utilization.

When using this parameter instead of "Schema Name" on page 42, you'll likely need to schema-qualify other parameters that reference stored procedure, function, or table names. In particular, parameters "Method and Timing (Table-Local)" on page 57 and "Event Log Table Name" on page 63 are affected.

The following table lists the properties of this parameter:

Property	**Value**
Tag Name	sync-tables
Required?	yes[1]
Case-Sensitive?	See "Undelimited Identifier Case-Sensitivity" on page 116.
Delimiters	semicolon, white space, comma
Sample Value	indirect.usr; indirect.grp
Default Value	(none)

[1]When this parameter is used, leave the Schema Name parameter empty or omitted from a configuration. See "Schema Name" on page 42.

Include Filter Expression

The Include Filter Expression parameter is operative only when the Schema Name parameter is used. See "Schema Name" on page 42.

The following table lists the properties of this parameter:

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 operative only when the Schema Name parameter is used. See "Schema Name" on page 42.

The following table lists the properties of this parameter:

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)

Connectivity Parameters

- "Connection Initialization Statements" on page 45
- "Use Minimal Number of Connections?" on page 44
- "Connection Properties" on page 45

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, 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 set to Boolean True, the number of required database connections is reduced to two. One 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.

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

[1]This default is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean False.

NOTE: 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:

Property	Value
Tag Name	connection-init
Required?	no
Case-Sensitive?	See "Undelimited Identifier Case-Sensitivity" on page 116.
Delimiters	semicolon
Sample Value	USE idm; SET CHAINED OFF
Default Value	(none)

Connection Properties

The Connection Properties parameter specifies authentication properties. This parameter is useful for specifying properties that cannot be specified via the JDBC URL specified in the Authentication Context parameter. See "Authentication Context" on page 38.

The primary purpose of this parameter is to enable interoperability with the "Sybase Adaptive Server Enterprise JConnect JDBC Driver" on page 110 when using a custom SSL socket implementation.

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 they must be delimited by the ';' character.

When using connection properties, authentication information may be passed via the JDBC URL specified in the Authentication Context parameter or here. See "Authentication Context" on page 38.

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 38 and "Application Password" on page 38. For username, the token is `{$username}`. For password, the token is `{$password}`.

The following table lists the properties of this parameter:

Property	Value
Tag Name	connection-properties
Required?	no
Case-Sensitive?	third-party JDBC driver-dependent
Delimiters	semicolon
Sample Value	USER={$username}; PASSWORD={$password}; SYBSOCKET_FACTORY=DEFAULT
Default Value	(none)

Compatibility Parameters

- "JDBC Driver Descriptor Filename" on page 46
- "Database Descriptor Filename" on page 47
- "Use Manual Transactions?" on page 47
- "Transaction Isolation Level" on page 48
- "Reuse Statements?" on page 48
- "Number of Returned Result Sets" on page 49
- "Enable Statement-Level Locking?" on page 50
- "Lock Statement Generator Class" on page 50
- "Enable Referential Attribute Support?" on page 50
- "Force Username Case" on page 51
- "Left Outer Join Operator" on page 51
- "Retrieve Minimal Metadata?" on page 52
- "Function Return Method" on page 52
- "Supports Schemas in Metadata Retrieval?" on page 52
- "Sort Column Names By" on page 53

JDBC Driver Descriptor Filename

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

The following table lists the properties of this parameter:

Property	Value
Tag Name	jdbc-driver-descriptor

Property	Value
Required?	no
Case-Sensitive?	platform-dependent
Sample Value	my_custom_jdbc_driver_descriptor.xml
Default Value	(none)

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:

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

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:

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

[1]This default is derived dynamically from descriptor files and database metadata at runtime.

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

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

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. I

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 116.

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:

Property	**Value**
Tag Name	transaction-isolation-level
Required?	no
Case-Sensitive?	no
Default Value	(dynamic[1])
Legal Values	unsupported none read uncommitted read committed repeatable read serializable

[1]This default 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 SQL Server 2000 Driver for JDBC.

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:

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

[1]This default is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean True.

NOTE: 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 Drivers" on page 108 when evaluating the results of arbitrary SQL statements.

The following table lists the properties of this parameter:

Property	Value
Tag Name	handle-stmt-results
Required?	no
Sample Value	one
Default Value	(dynamic[1])
Legal Values	none, no (none) single, one (one) multiple, many, yes (multiple)

[1]This default 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:

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

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. The `DBLockStatementGenerator` interface is documented in the java documents that ship with the driver.

The following table lists the properties of this parameter:

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

[1]This default 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:

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

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 105.

The following table lists the properties of this parameter:

Property	Value
Tag Name	force-username-case
Required?	no
Default Value	(don't force)
Legal Values	lower (to lower case) mixed (to mixed case) upper (to upper case)

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:

Property	Value
Tag Name	left-outer-join-operator
Required?	no
Default Value	(dynamic[1])
Legal Values	*= (+) LEFT OUTER JOIN

[1]This default 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 D, "java.sql.DatabaseMetaData Methods," on page 139. Optional metadata methods are required for multivalue and referential attribute synchronization.

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

[1]This default is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean False.

NOTE: 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" on page 105.

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.

Property	Value
Tag Name	function-return-method
Required?	no
Default Value	(dynamic[1])
Legal Values	result set return value (scalar return value)

[1]This default 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 105.

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

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

[1]This default 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.

Property	Value	
Tag Name	column-position-comparator	
Required?	no	
Default Value	(dynamic[1])	
Legal Values	com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringComparator (any java.util.Comparator that accepts java.lang.String arguments)	(hexadecimal value) (string collation order)

[1]This default 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.

Subscription Parameters

The following table summarizes subscriber-level parameters and their 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[1])	no

[1]This default is derived dynamically from descriptor files at runtime.

Subscription parameters are in two subcategories:

- "Uncategorized Parameters" on page 54
- "Primary Key Parameters" on page 56

Uncategorized Parameters

- "Disable Subscriber?" on page 54
- "Disable Statement-Level Locking?" on page 55
- "Check Update Counts?" on page 55
- "Add Default Values on Insert?" on page 55

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.

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

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?” on page 50 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.

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

Check Update Counts?

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

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

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.

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

[1]This default is derived dynamically from descriptor files at runtime.

Primary Key Parameters

- "Generation/Retrieval Method (Table-Global)" on page 56
- "Retrieval Timing (Table-Global)" on page 57
- "Method and Timing (Table-Local)" on page 57

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 is indicated.

The following table identifies the supported methods and timings:

	Timing: before (row insertion)	**Timing: after (row insertion)**
Method: none (subscription event)	X	0[1]
Method: driver (SELECT MAX())	X	X
Method: auto (auto-generated/identity column)	0[2]	X
Method: (stored procedure/function)	X	X

[1]The Subscriber channel automatically overrides this timing to before.
[2]The Subscriber channel automatically overrides this timing 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. See "Method and Timing (Table-Local)" on page 57.

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:

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

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 107.

Property	Value
Tag Name	key-gen-method
Required?	no
Default Value	none (subscription event)
Legal Values	none (subscription event) driver (SELECT MAX()) auto (auto-generated/identity column)

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 parameter Method and Timing (Table-Local) overrides this parameter. See "Method and Timing (Table-Local)" on page 57.

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.

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)

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 "Table/View Name" on page 43, you'll probably need to explicitly schema-qualify any tables, views, stored procedures or functions referenced in this parameter's value. When you use parameter "Schema Name" on page 42, 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 other than the implicit schema, they must be schema-qualified.

BNF

The BNF (Backus Naur Form (http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html)) 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 look something like:

```
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 database 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 can 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 parameter "Transaction Isolation Level" on page 48.

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 93.

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 random alpha character sequence. 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 look something like:

```
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 107.

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 look something like:

```
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 database 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 look something like:

```
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

Retrieval timing 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.

IMPORTANT: 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.

IMPORTANT: 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:

Property	Value
Tag Name	key-gen
Required?	no
Case-Sensitive?	See "Undelimited Identifier Case-Sensitivity" on page 116.
Sample Value	`usr("?=proc_idu()", before)`
Default Value	(none)
Legal Values	(any string adhering to the BNF)

Publication Parameters

The following table summarizes publisher-level parameters and their 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)	no
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]
Time of day	time-of-day	15:30:00	(none)	no[2]
Post polling statements	post-poll-stmt	DELETE FROM direct.direct_process	(none)	no
Batch size	batch-size	16	1	no
Heartbeat interval (in minutes)	pub-heartbeat-interval	10	0	no

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

Publication parameters fall into four major subcategories:

- "Uncategorized Parameters" on page 54
- "Triggered Publication Parameters" on page 63
- "Triggerless Publication Parameters" on page 65
- "Polling Parameters" on page 65

Uncategorized Parameters

- "Disable Publisher?" on page 61
- "Publication Mode" on page 62

Disable Publisher?

The Disable Publisher parameter specifies whether the Publisher channel is disabled. If it is disabled, the Publisher channel does not establish a connection to the target database.

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.

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

Disable Statement-Level Locking?

This parameter specifies whether database resources should be explicitly locked on this channel before each SQL statement is executed. This parameter is active only if the parameter "Enable Statement-Level Locking?" on page 50 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.

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

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 dredges all tables/views in the synchronization schema for changes, and synthesizes events.

The following table lists the properties of this parameter:

Property	Value
Tag Name	publication-mode
Required?	no
Default Value	1 (triggered)
Legal Values	1 (triggered) 2 (triggerless)

Triggered Publication Parameters

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 "The Event Log Table" on page 81.

When using "Table/View Name" on page 43, you'll probably need to explicitly schema-qualify this table name. When you use "Schema Name" on page 42, 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:

Property	**Value**
Tag Name	log-table
Required?	no[1]
Case-Sensitive?	See "Undelimited Identifier Case-Sensitivity" on page 116.
Sample Value	eventlog
Default Value	(none)

[1]This parameter is required if "Publication Mode" on page 62 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 66.
- Place a before-delete trigger on the event log table to intercept delete events executed against the event log table and 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:

Property	**Value**
Tag Name	delete-from-log
Required?	no
Default Value	0 (no)

Property	Value
Legal Values	1, yes, true (yes) 0, no, false (no)

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

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:

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

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

Enable future event processing?

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

The following table lists the properties of this parameter:

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

Triggerless Publication Parameters

Startup Option

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

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

The following table lists the properties of this parameter:

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

Polling Parameters

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:

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

NOTE: We recommend setting 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).

The following table lists the properties of this parameter:

Property	Value
Tag Name	time-of-day
Required?	no
Sample Value	13:00:00 (1PM)
Default Value	(none)
Legal Values	hh:mm:ss (h = hour, m = minute, s = second)

NOTE: This parameter overrides the parameter Polling Interval (In Seconds). See "Polling Interval (In Seconds)" on page 65.

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'll probably need to 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:

Property	Value
Tag Name	post-poll-stmt
Required?	no
Case-Sensitive?	See "Undelimited Identifier Case-Sensitivity" on page 116.
Delimiters	semicolon
Sample Value	DELETE FROM direct.direct_process
Default Value	(none)
Legal Values	(any set of legal SQL statements)

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:

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

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:

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)

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 the Identity Manager Administration Guide (http://www.novell.com/documentation).

The driver supports the following seven trace levels:

Level	Description
1	Minimal tracing

Level	Description
2	Database properties
3	Connection status, SQL statements, event log records
4	Verbose output
5	Database resource allocation/deallocation, triggerless publication state
6	JDBC API (invoked methods, passed arguments, returned values, etc.)
7	Third-party JDBC 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 have to 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.

Parameter Name	Compatibility Value
Table/view name(s)	*delimited-list-of-table-names*
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

5 Advanced Configuration

After installing the sample driver configuration, customize it for specialized use.

- "Schema Mapping" on page 69
- "The Event Log Table" on page 81
- "XDS Event to SQL Statement Mapping" on page 80
- "Embedding SQL Statements in XDS Events" on page 89

Schema Mapping

The following table shows a high-level view of how the driver maps Novell® 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 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:

eDirectory Object	Database Object
Classes	Tables
Attributes	Columns

eDirectory Object	Database Object
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 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 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`

NOTE: 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 eDirectory attributes, such as mapping the single-value eDirectory attribute Password Minimum Length to the single-valued 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)
)
```

NOTE: 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 `usrphone.phoneno`).

The following table contains sample data for `indirect.usr_phone`.

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` as `usr.phoneno`, not `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>
```

NOTE: 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>
```

The interpretation of the above example is that 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)
)
```

NOTE: 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 aren’t aware 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 53.

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 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 single user is usually more efficient than synchronizing all members of a single group.

Direct Synchronization

In a direct synchronization model, the driver maps the following:

Identity Vault Object	Database Object
Classes	Views
Attributes	View Columns
1 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. If views are strictly 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 updated. For a list of databases that support instead-of-triggers, see “Database Features” on page 114.

View Column Meta-Identifiers

A view is a logical table. Unlike tables, views do not physically exist in the database. As such, views cannot have traditional primary key/foreign key constraints. To simulate these constructs, the driver for JDBC 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.

Constraint Prefixes (case-insensitive)	Interpretation
pk_	primary key
fk_	foreign key
sv_	single-value
mv_	multi-value

The following example views contain all of these constraint prefixes:

```
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 (http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html)) 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 driver for JDBC 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:

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

For the driver, view column name equivalency is determined without respect to meta information. 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.

Primary Key Columns

Primary key column names must be unique between all views in the synchronization schema.

Schema Mapping

Schema mapping for views and view columns is 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 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 eDirectory schema, it has no way of knowing whether an eDirectory attribute has a single value or multivalue. 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 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 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 eDirectory attributes Given Name and Surname to the single-value columns `fname` and `lname` respectively.

XDS Event to SQL Statement Mapping

The following table summarizes how the Subscriber channel maps XDS events to DML SQL statements 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

XML Event	SQL Equivalent
<query>	1 parent table select statement 1 select statement for each child table
<move> <rename> <modify-password> <check-object-password>	0 statements unless bound to embedded SQL statements

The following table summarizes how the Subscriber channel maps XDS events to DML SQL statements 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 statements 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

The Event Log Table

The event log table stores Publication events. This section discusses the structure and limitations 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 don't use column position, order is determined by the Sort Column Names By parameter. See "Sort Column Names By" on page 53.

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 64.

Event Log Columns

This section describes columns in the event log table. Columns are ordered by position.

1. `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.

2. `status`

   The `status` column indicates the state of a given row. The following table lists permitted values:

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.

   NOTE: Status values are case-sensitive.

3. `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:

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:

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 time, space, and complexity tradeoffs.

4. `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 64.

5. `perpetrator`

   This column identifies the database user who instigated the event. A `NULL` value is interpreted as a user other than the driver user. As such, 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 Publisher parameter Allow Loopback is set to Boolean True. See "Allow Loopback?" on page 64.

6. `table_name`

   The name of the table or view where the event occurred.

7. `table_key`

   Format values for this column exactly the same in all triggers for a logical database class. The BNF or Backus Naur Form (http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html) 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 used throughout this chapter, this column's value might be `pk_empno=1`.

   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").

NOTE: If primary key values placed in the `table_key` field contain the following characters, delimit (double-quote) the values: , ; ' + = \ " < >.

8. `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 will not start.

9. `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 will not start.

10. `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 will not start.

Event Types

The following table shows the basic correlation between publication event types and the XDS XML generated by the publisher.

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:

idu	fname	lname	photo
1	Jack	Frost	0xAAAA

The following table shows the current contents of `usr` after the row has been updated:

idu	fname	lname	photo
1	John	Doe	0xBBBB

1. 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=.

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>
```

2. 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=.

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>
```

3. 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.

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>
```

4. Delete Row

   The table below shows the contents of the event log table after the row in table `usr` has been deleted.

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>
```

5. Insert Row (Query-Back)

   The following table shows the contents of the event log table after a new row is inserted into table `usr`.

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>
```

6. Update Row (Query-Back)

   The table below shows the contents of the event log table after the row in table `usr` has been updated.

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>
```

7. 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.

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>
```

8. 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.

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>
```

Embedding SQL Statements in XDS Events

The following section includes information to help you embed SQL in XDS events.

All examples reference table `usr` below. The primary key generation method used to obtain primary key values is irrelevant to the examples in this section.

```
CREATE TABLE usr
(
    idu   INTEGER  NOT NULL,
    fname VARCHAR2(64),
    lname VARCHAR2(64),

    CONSTRAINT pk_usr_idu PRIMARY KEY(idu)
);
```

You can use embedded SQL in XDS events. In the same way that you can install database triggers on a table and cause side effects in a database, embedded SQL in XDS events acts as a virtual trigger with similar capabilities.

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.

NOTE: The namespace prefix `jdbc` used throughout this section is implicitly bound to the namespace `urn:dirxml:jdbc` when referenced outside of an XML document.

The following XML example shows an embedded SQL statement.

```
<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>
```

IMPORTANT: Use namespace-prefixed elements and attributes to embed SQL. Otherwise, the driver will not recognize them. In the above example, the namespace is urn:dirxml:jdbc. The prefix is the identifier to the right of the xmlns identifier. In the above example, the prefix is jdbc. In practice, the prefix can be whatever you want it to be, as long as it is bound to the correct namespace.

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
```

Variable Substitution

Rather than require you to parse field values from an association, the Subscriber channel supports variable substitution in embedded SQL statements. For example:

```
<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>
```

Variable placeholders must adhere to the XSLT attribute value template syntax {$*field-name*}. Also, the association element must precede the `<jdbc:statement>` element in the XDS document, or must be present as a child of the `<jdbc:statement> element.`

The *field-name* variable must refer to one of the naming RDN attribute names in the association value. The above example has 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 variable substitution because the association has not been created yet. Additionally, any embedded SQL statements using variable substitution 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 `{$$password}` to refer to the contents of the immediately preceding `<password>` element within the same document.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr">
          <password>Doe{$idu}</password>
          <add-attr name="lname">
              <value>Doe</value>
          </add-attr>
     </add>
     <jdbc:statement>
          <jdbc:sql>CREATE USER Doe 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 38.

Statement Placement

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.

```
<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:

```
<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. For example:

```
<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:

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`adsf
- `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:

```
<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:

idu	fname	lname
1	John	Doe

The XML document below would result 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 are visible to the driver. However, XDS queries do.

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="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>CREATE USER indirect IDENTIFIED BY novell
        </jdbc:sql>
    </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
INSERT INTO indirect.usr(lname) VALUES('Doe');
COMMIT; -- explicit commit
SET AUTOCOMMIT ON
CREATE USER indirect 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 fail and the create user statement succeed. The Subscriber channel 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 so grouped, 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="lname">
            <value>Doe</value>
        </add-attr>
        <password>Doe{$idu}</password>
    </add>
    <jdbc:statement jdbc:op-id="0">
        <jdbc:sql>CREATE USER Doe 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.

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/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>
```

Embedded Multiple 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/functions varies by database. For additional information, see “Stored Procedure and Function JDBC Call Syntaxes” on page 115.

6 Third-Party JDBC Drivers

- "Third-Party JDBC Driver Interoperability" on page 99
- "JDBC Drivers: Four Types" on page 99
- "Supported Third-Party JDBC Drivers" on page 100
- "Using Unsupported Third-Party JDBC Drivers" on page 111
- "Security Issues" on page 111

Third-Party JDBC Driver Interoperability

The Identity Manager Driver for JDBC 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 Driver for JDBC works against any given database. As a general rule, if the Driver for JDBC interoperates well with a given third-party JDBC driver, it will interoperate 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 this section. They are usually free, mature, and known to interoperate well with the Driver for JDBC and the databases they target. You can use other third-party drivers, but Novell® does not support them.

In general, most third-party drivers are backward compatible. However, even if they are generally 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.

JDBC Drivers: Four Types

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.

IMPORTANT: If you choose to use a type 1 or type 2 driver with the Driver for JDBC, use the remote loader to ensure the integrity of the directory process.

Supported Third-Party JDBC Drivers

- "BEA Weblogic jDriver for Microsoft SQL Server" on page 102
- "IBM DB2 Universal Database JDBC Drivers" on page 103
- "Microsoft SQL Server 2000 Driver for JDBC" on page 106
- "MySQL Connector/J JDBC Driver" on page 107
- "Oracle Thin Client JDBC Drivers" on page 108
- "PostgreSQL JDBC Driver" on page 109
- "Sybase Adaptive Server Enterprise JConnect JDBC Driver" on page 110

Third-Party JDBC Driver Features

The following table summarizes third-party JDBC driver features:

Driver	Supports Encrypted Transport?	Supports Retrieval of Auto-Generated Keys?
BEA Weblogic jDriver	No	No
IBM DB2 UDB Type 3	No	No
IBM DB2 UDB Type 4	No	No
Informix	No	No
Microsoft 2000	No	No
MySQL Connector/J	Yes	Yes
Oracle Client Thin	Yes	No

Driver	Supports Encrypted Transport?	Supports Retrieval of Auto-Generated Keys?
PostgreSQL	Yes*	No
Sybase jConnect	Yes	No

* For versions JDBC 3 (Java 1.4) and later.

JDBC URL Syntaxes

The following table lists URL syntaxes for supported third-party JDBC drivers:

Third-Party JDBC Driver	JDBC URL Syntax
Oracle Thin Client	jdbc:oracle:thin:@*ip-address*:1521:*sid*
IBM DB2 UDB Type 3	jdbc:db2://*ip-address*:6789/*database-name*
IBM DB2 UDB Type 4, Universal	jdbc:db2://*ip-address:50000*/*database-name*
BEA Weblogic* jDriver	jdbc:weblogic:mssqlserver4:*database-name@ip-address*:1433
Microsoft SQL Server	jdbc:microsoft:sqlserver://*ip-address-or-dns-name*:1433;DatabaseName=*database-name*
Sybase jConnect	jdbc:sybase:Tds:*ip-address*:2048/*database-name*
MySQL Connector/J	jdbc:mysql://*ip-address*:3306/*database-name*
Informix	jdbc:informix-sqli://*ip-address*:1526/*database-name*:informixserver=*server-id*
PostgreSQL	jdbc:postgresql://*ip-address*:5432/*database-name*

JDBC Driver Class Names

The following table lists the fully-qualified Java class names of supported third-party JDBC drivers:

Third-party JDBC Driver	Class Name
BEA Weblogic jDriver	weblogic.jdbc.mssqlserver4.Driver
IBM DB2 UDB Type 3	COM.ibm.db2.jdbc.net.DB2Driver
IBM DB2 UDB Type 4, Universal	com.ibm.db2.jcc.DB2Driver
Informix	com.informix.jdbc.IfxDriver
Microsoft 2000	com.microsoft.jdbc.sqlserver.SQLServerDriver
MySQL Connector/J	org.gjt.mm.mysql.Driver
Oracle Thin Client	oracle.jdbc.driver.OracleDriver
PostgreSQL	org.postgresql.Driver

Third-party JDBC Driver	Class Name
Sybase jConnect 5.5	com.sybase.jdbc2.jdbc.SybDriver

BEA Weblogic jDriver for Microsoft SQL Server

Supported Database Version:	Microsoft SQL Server 6.5, 7.x, 8.x (2000)
Class Name	weblogic.jdbc.mssqlserver4.Driver
Type	4
URL Syntax	jdbc:weblogic:mssqlserver4:*database-name@ip-address*:1433
Download Instructions	Register for free and download the latest version of Weblogic server. Run the installer. The weblogic.jar file is installed in the *install-dir*/ server/lib directory. BEA Download Center (http://commerce.bea.com/ showallversions.jsp?family=WLS)
Filename	weblogic.jar
Documentation URLs	jDriver Documentation (http://e-docs.bea.com/wls/docs81/ mssqlserver4/)

NOTE: The BEA Weblogic is included in the supported third-party driver listing to provide JDBC access to Microsoft SQL server 7. Microsoft's driver supports only version 8 (2000).

Compatibility

The BEA Weblogic driver is backward compatible. Database server and driver updates are infrequent.

Security

The BEA Weblogic driver does not support encrypted transport.

Known Issues

- The BEA Weblogic driver is not free. It must be purchased and properly licensed.
- Association values that contain `UNIQUEIDENTIFIER` columns are inconsistent between driver versions.

  Earlier versions of the BEA Weblogic driver returned a non-standard java.sql.Types (http:// java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) value for native `UNIQUEIDENTIFIER` columns. To compensate, the Driver for JDBC mapped that non-standard type to the standard type java.sql.Types.BINARY (http://java.sun.com/j2se/1.5.0/ docs/api/java/sql/Types.html) because it best mirrored the native database type -- a 16 byte value. This mapping results in a Base64-encoded association value.

  Later versions of the BEA Weblogic driver return a standard type java.sql.CHAR (http:// java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html). This mapping results in a non-Base64-encoded association value, effectively invalidating all associations generated by using earlier versions of the BEA Weblogic driver. This change effectively breaks backward compatibility.

The best solution to this problem is to continue using the earlier version of the BEA Weblogic driver. If you must upgrade, you'll have to remove all invalidated associations and reassociate all previously-associated objects.

- The BEA Weblogic driver throws a java.lang.IllegalMonitorStateException (http://java.sun.com/j2se/1.5.0/docs/api/java/lang/IllegalMonitorStateException.html) when method `java.sql.Connection.getConnection(String url, String username, String password)` is called on AIX.

IBM DB2 Universal Database JDBC Drivers

Type 3

Supported Database Versions:	7.x
Class Name:	COM.ibm.db2.jdbc.net.DB2Driver
Type	3
URL Syntax:	jdbc:db2://*ip-address*:6789/*database-name*
Download Instructions:	Copy the file from the database server. file:///*database-installation-directory*/java
File Name:	db2java.zip
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)

IMPORTANT: The type 3 driver is deprecated for version 8.

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.

IMPORTANT: 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 prove time intensive 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.

Type 4: Universal Drivers

Supported Database Versions	8.x
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)
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)

NOTE: Unlike the type 3 driver, the type 4 driver has only a minimal set of defined error codes. This absence inhibits the Driver for JDBC's ability to distinguish between connectivity, retry, authentication, and fatal error conditions.

Compatibility

The IBM DB2 driver is backward compatible, although it doesn't work with database version 7. 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 prove time intensive 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

Supported Database Versions	Dynamic Server 7.x, 9.x
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	ifxjdbc.jar, ifxjdbcx.jar (optional)
Documentation URLs	Informix Information Center (http://publib.boulder.ibm.com/infocenter/ids9help/index.jsp) Informix JDBC Driver (http://www-306.ibm.com/software/data/informix/pubs/library/jdbc_2.html)

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 Driver for JDBC to interoperate with the Informix driver against ANSI-compliant databases.

Display Name	Tag Name	Value
Supports schemas in metadata retrieval?	supports-schemas-in-metadata-retrieval	false
Force username case:	force-username-case	upper

Dynamic Parameter Defaults

The following table lists driver compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly override these settings.

Display Name	Tag Name	Value
Function return method:	function-return-method	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 52 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 51 to `upper`.

Microsoft SQL Server 2000 Driver for JDBC

Supported Database Versions:	8 (2000)
Class Name	com.microsoft.jdbc.sqlserver.SQLServerDriver
Type	4
URL Syntax	jdbc:microsoft:sqlserver://*ip-address-or-dns-name*:1433;DatabaseName=*database-name*
Download Instructions	Microsoft JDBC Downloads (http://www.microsoft.com/downloads/results.aspx?sortCriteria=date&OSID=&productID=&CategoryID=&freetext=jdbc&DisplayLang=en&DisplayEnglishAlso=)
Filenames	msbase.jar, mssqlserver.jar, msutil.jar

Compatibility

The SQL Server 2000 driver is backward compatible, although it doesn't work with database version 7. Database server and driver updates are infrequent.

Security

The SQL Server 2000 driver does not support encrypted transport.

URL Properties

Delimit URL properties by using a ';' character.

The following table lists values for the SelectMethod URL property for the SQL Server 2000 driver.

Legal Value	Description
`direct`	The default value; doesn't allow for multiple active statements on a single connection
`cursor`	Allows for multiple active statements on a single connection

Dynamic Parameter Defaults

The following table lists driver compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly override these settings.

Display Name	Tag Name	Value
Reuse Statements?	reuse-statements	false

Known Issues

- Can't start manual transaction because of cloned connections.

  An implementation anomaly that doesn't allow concurrent statements to be active on the same connection causes the most common problem experienced with the SQL Server 2000 driver. Unlike other third-party implementations, the SQL Server 2000 driver can have only one java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) object active at a time on a given connection.

  If you attempt to use more than one statement object, the following error is issued: "Can't start manual transaction mode because there are cloned connections." This error can occur only if driver compatibility parameter "Reuse Statements?" on page 48 is set to Boolean True. As a best practice, never explicitly set this parameter. Instead, defer to the dynamic default value.

  An alternative is to place the delimited property `;SelectMethod=cursor` at the end of the URL string. For additional information on this issue, consult the following support articles:

  - Document 30096 (http://knowledgebase.datadirect.com/kbase.nsf/SupportLink+Online/30096?OpenDocument) by DataDirect Technologies*
  - Article 313181 (http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B313181) by Microsoft

- Association values that contain `UNIQUEIDENTIFIER` columns are inconsistent between driver versions.

  Earlier versions of the SQL Server 2000 driver returned a non-standard java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) value for native `UNIQUEIDENTIFIER` columns. To compensate, the Driver for JDBC mapped that non-standard type to the standard type java.sql.Types.BINARY (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) because it best mirrored the native database type -- a 16 byte value. This mapping results in a Base64-encoded association value.

  Later versions of the SQL Server 2000 driver return a standard type java.sql.CHAR (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html). This mapping results in a non-Base64-encoded association value, effectively invalidating all associations generated by using earlier versions of the SQL Server 2000 driver. This change effectively breaks backward compatibility.

  The best solution to this problem is to continue using the earlier version of the SQL Server 2000 driver. If you must upgrade, you'll have to remove all invalidated associations and reassociate all previously-associated objects.

MySQL Connector/J JDBC Driver

Supported Database Versions	3.x, 4.x

Class Name	org.gjt.mm.mysql.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
Documentation URLs	MySQL Connector/J Documentation (http://dev.mysql.com/doc/connector/j/en/) Connecting Securely Using SSL (http://dev.mysql.com/doc/refman/5.0/en/cj-using-ssl.html)

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 Driver for JDBC can interoperate with the Connector/J driver against MyISAM tables.

Display Name	Tag Name	Value
Use manual transactions?	use-manual-transactions	false

Oracle Thin Client JDBC Drivers

Supported Database Versions	8i, 9i, 10g
Class Name	oracle.jdbc.driver.OracleDriver
Type	4
URL Syntax	jdbc:oracle:thin:@*ip-address*:1521:*sid*
Download Instructions	Register for free and download. Oracle Technology Network (http://otn.oracle.com/software/tech/java/sqlj_jdbc/content.html)
1.1 Filenames	classes111.zip, nls_charset11.zip (optional)
1.2-3 Filenames	classes12.zip, ocrs12.zip (optional), nls_charset12.zip (optional)
1.4 Filenames	ojdbc14.jar, ocrs12.zip (optional)

Documentation URLs	Oracle Advanced Security (http://www.oracle.com/technology/sample_code/deploy/security/files/secure_thin_driver/readme.html)

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 you use the 1.4 version.

Security

The Thin Client driver supports Oracle Advanced Security encrypted transport.

Dynamic Parameter Defaults

The following table lists driver compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly override these settings.

Display Name	Tag Name	Value
Number of returned result sets:	handle-stmt-results	single

Known Issues

- High CPU utilization triggered by execution of embedded SQL statements.

  The most common problem experienced with this driver is high CPU utilitization, because of 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 driver compatibility parameter "Number of Returned Result Sets" on page 49 other than `single`, `no` or `one` and an embedded SQL statement is being executed and 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.

  - Always place a `jdbc:type` attribute on any embedded `<jdbc:statement>` elements.

  NOTE: The jdbc namespace prefix maps to urn:dirxml:jdbc.

PostgreSQL JDBC Driver

Supported Database Versions	6.x, 7.x, 8.x
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)

Documentation URLs	JDBC Driver Documentation (http://jdbc.postgresql.org/documentation/docs.html) Using SSL (http://jdbc.postgresql.org/documentation/80/ssl.html)

NOTE: The filename of the PostgreSQL varies by database version.

Compatibility

The latest builds of the PostgreSQL driver are backward compatible through server version 7.2. 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

Supported Database Versions	Adaptive Server Enterprise 11.x, 12.x
Class Name	com.sybase.jdbc2.jdbc.SybDriver (for jconn2.jar) com.sybase.jdbc3.jdbc.SybDriver (for jconn3.jar)
Type	4
URL Syntax	jdbc:sybase:Tds:*ip-address*:2048/*database-name*
Download Instructions	Sybase Downloads (http://www.sybase.com/detail?id=1009796)
Filenames	jconn2.jar or jconn3.jar
Documentation URLs	jConnect Documentation (http://sybooks.sybase.com/onlinebooks/group-jc/jcg0600e/prjdbc)

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 45.

Connection Properties

The following table lists an important connection property for this driver.

Property	Significance
SYBSOCKET_FACTORY	Can be used to specify the class name of a custom socket implementation that supports encrypted transport

Using Unsupported Third-Party JDBC Drivers

Minimum Third-Party JDBC Driver Requirements

The Driver for JDBC 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 Driver for JDBC makes, see Appendix D, "java.sql.DatabaseMetaData Methods," on page 139.

- Support other required JDBC methods

  For a list of required JDBC methods that the Driver for JDBC uses, refer to Appendix E, "Utilized JDBC Methods," on page 141. 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 Driver for JDBC 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 Driver for JDBC 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'll probably need to write a custom driver descriptor file.

  Specifically, you'll 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 this product and a third-party driver, we recommend the following:

- Run this product remotely on the database server.
- Use SSL to encrypt communications between the Identity Manager server and the database server.

If you cannot run the Driver for JDBC 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.

7 Supported Databases

- "Database Interoperability" on page 113
- "Supported Databases" on page 113
- "Database Characteristics" on page 114

Database Interoperability

The Identity Manager Driver for JDBC is designed to interoperate 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/database versions have been tested and are recommended for use with this product:

Database	Minor Version
IBM* DB2 Universal Database (UDB) 7	7.2 or higher
IBM* DB2 Universal Database (UDB) 8	8.1 or higher
Informix* Dynamic Server (IDS)	9.40 or higher
Microsoft SQL Server 7	7.5, Service Pack 4 or higher
Microsoft* SQL Server 8 (2000)	Service Pack 3a or higher
MySQL* 3	3.23.50 or higher
MySQL* 4	4.1 or higher
Oracle 8i	Release 3 (8.1.7) or higher
Oracle 9i	Release 2 (9.2.0.1) or higher
Oracle 10g	Release 1 (10.0.2.1) or higher
PostgreSQL 7	7.4.6 or higher
Sybase* Adaptive Server Enterprise (ASE) 12	12.5 or higher

You can use the Driver for JDBC with other databases/database versions. However, Novell® does not support them. To interoperate with this product, a database must:

- Support the SQL-92 entry level grammar.
- Be JDBC-accessible.

Database Characteristics

- "Database Features" on page 114
- "IBM DB2 Universal Database (UDB)" on page 117
- "Informix Dynamic Server (IDS)" on page 118
- "Microsoft SQL Server" on page 118
- "MySQL" on page 119
- "Oracle" on page 120
- "PosgreSQL" on page 120
- "Sybase Adaptive Server Enterprise (ASE)" on page 121

Database Features

The following table is a summary of database features:

Database	Schemas	Views	Identity Columns	Sequences	Stored Procedures	Functions	Triggers	Instead-Of-Triggers
IBM DB2 UDB 7	X	X	X	0	X[1]	X[1]	X	0
IBM DB2 UDB 8	X	X	X	0	X[1]	X[1]	X	X
Informix IDS 9	X	X	X[2]	0	X[3]	X	X	0
MS SQL 7	X	X	X	0	X	0	X	0
MS SQL 8	X	X	X	0	X	X	X	X
MySQL 4	0	0	X[4]	0	0	0	0	0
Oracle 8i, 9i, 10g	X	X	0	X	X	X	X	X
Postgres 7	X	X	X[5]	X	X	X	X[6]	X[6]
Sybase ASE 12	X	X	X	0	X	0	X	0

DB2 natively supports stored procedures/functions written in Java. To write procedures using the native SQL procedural language, install a C compiler on the database server.

The Informix identity column keyword is `SERIAL8`.

Informix stored procedures cannot return values.

The MySQL identity column keyword is `AUTO_INCREMENT`.

You can use a Postgres sequence object to provide default values for primary key columns, effectively simulating an identity column.

Postgres 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/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:

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	SELECT (CURRENT_TIMESTAMP)	Yes
Oracle	SELECT (SYSDATE) FROM SYS.DUAL	No
PostgreSQL	SELECT (CURRENT_TIMESTAMP)	Yes
Sybase ASE	SELECT GETDATE()	No

Stored Procedure and Function JDBC Call Syntaxes

The following table lists the SQL syntax for calling a stored procedure or function. This is useful for formatting procedure and function calls in embedded SQL statements.

Database	**Stored Procedure/Function JDBC Call Syntax**
IBM DB2 UDB	{call *schema-name*.*procedure-name*(*parameter-list*)}
Informix IDS	EXECUTE [PROCEDURE | FUNCTION] *schema-name*.*procedure-name*(*parameter-list*)
MSSQL	EXECUTE *schema-name*.*procedure-name*(*parameter-list*)
MySQL	(NA)
Oracle	CALL *schema-name*.*procedure-name*(*parameter-list*)
PostgreSQL	SELECT *schema-name*.*procedure-name*(*parameter-list*)
Sybase ASE	EXECUTE *schema-name*.*procedure-name*(*parameter-list*)

Left Outer Join Operators

The following table lists outer join operators by database.

Database	**Left Outer Join Operator**	**Ansi-Compliant**
IBM DB2 UDB	LEFT OUTER JOIN	Yes
Informix IDS	LEFT OUTER JOIN	Yes

Database	Left Outer Join Operator	Ansi-Compliant
MSSQL	*=	No
MySQL	LEFT OUTER JOIN	Yes
Oracle	(+)	No
PostgreSQL	LEFT OUTER JOIN	Yes
Sybase ASE	*=	No

NOTE: Oracle supports the ANSI-compliant left outer join operator LEFT OUTER JOIN as of version 10g.

Undelimited Identifier Case-Sensitivity

Database	Case-Sensitive?
IBM DB2 UDB	No
Informix IDS	No
MSSQL	No
MySQL	Yes
Oracle	No
PostgreSQL	No
Sybase ASE	Yes

Supported Transaction Isolation Levels

Database	None	Read Uncommitted	Read Committed	Repeatable Read	Serializable	URL
IBM DB2 UDB	0	X	X[1]	X	X	Setting JDBC Transaction Isolation Levels (http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/tjvjdiso.htm)
MySQL (InnoDB Table Type)	0	X	X	X[1]	X	InnoDB Transaction Isolation Levels (http://dev.mysql.com/doc/mysql/en/innodb-transaction-isolation.html)
Oracle	0	0	X[1]	0	X	JDBC Transaction Optimization (http://www.oracle.com/technology/oramag/oracle/02-jul/o42special_jdbc.html)
PostgreSQL	0	0[2]	X[1]	0[2]	X	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:

Database	Commit Keyword
IBM DB2 UDB	COMMIT
Informix IDS	COMMIT WORK[1]
MSSQL	GO
MySQL	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)

The following table lists properties for this database.

Property	Value
Current Timestamp Statement	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
Stored Procedure/ Function Call Syntax	{call *schema-name*.*procedure-name*(*parameter-list*)}
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Dynamic Defaults

The following table lists database compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly override these 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 121.

Informix Dynamic Server (IDS)

The following table lists properties for this database.

Property	Value
Current Timestamp Statement	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES
Stored Procedure/ Function Call Syntax	EXECUTE [PROCEDURE | FUNCTION] *schema-name*.*procedure-name*(*parameter-list*)
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 Driver for JDBC implicitly sets at runtime. Do not explicitly overwrite these 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.

Microsoft SQL Server

The following table lists properties for this database:

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Stored Procedure/Function Call Syntax	EXECUTE *schema-name*.*procedure-name*(*parameter-list*)
Case-Sensitive?	No
Commit Keyword	GO

Property	Value
Left Outer Join Operator	*=

Dynamic Defaults

The following table lists database compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly overwrite these settings.

Display Name	Tag Name	Value
Add default values on insert?	add-default-values-on-view-insert	true
Left outer-join operator:	left-outer-join-operator	*=

MySQL

The following table lists properties for this database.

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Stored Procedure/Function Call Syntax	(NA)
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.

Display Name	Tag Name	Value
Supports schemas in metadata retrieval?	supports-schemas-in-metadata-retrieval	false

Known Issues

- `TIMESTAMP` columns, when 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

The following table lists properties for this database:

Property	Value
Current Timestamp Statement	SELECT (SYSDATE) FROM SYS.DUAL
Stored Procedure/Function Call Syntax	CALL *schema-name.procedure-name*(*parameter-list*)
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	(+)

Dynamic Defaults

The following table lists database compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly overwrite these 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

NOTE: The default exclusion filter is intended to omit from the synchronization schema dropped tables visible in Oracle 10g (database objects visible to the Driver for JDBC at runtime).

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.

PosgreSQL

The following table lists properties for this database:

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Stored Procedure/Function Call Syntax	SELECT *schema-name.procedure-name*(*parameter-list*)
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)

The following table lists properties for this database:

Property	Value
Current Timestamp Statement	SELECT GETDATE()
Stored Procedure/Function Call Syntax	EXECUTE *schema-name*.*procedure-name*(*parameter-list*)
Case-Sensitive?	Yes
Commit Keyword	GO
Left Outer Join Operator	*=

Dynamic Defaults

The following table lists database compatibility parameters that the Driver for JDBC implicitly sets at runtime. Do not explicitly overwrite these 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, will be truncated. If binary columns exceed the size of their respective eDirectory attributes, extra 0s will be 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.

  See "Known Issues" on page 121

8 Using the Association Utility

This section contains information on using the Association Utility. The utility normalizes associations of objects associated under the 1.0 or later versions of the Driver for JDBC. 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 Driver for JDBC, and supersedes all previous versions.

Understanding the Association Utility

The Association Utility supports seven independent operations:

Operation	Description	Read-Write Functionality
1	List objects associated with a driver (default).	Read-only
2	List objects that have multiple associations to a driver.	Read-only
3	List 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	List 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
5	Normalize object associations listed during operation 4.	Write

Operation	Description	Read-Write Functionality
6	List 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 "Editing Associations" on page 125.	Read-only
7	Modify object associations listed during operation 6. This operation requires two parameters (oldRDN and newRDN). See "Editing Associations" on page 125.	Write

Before You Begin

Modifying associations can potentially 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 database identifiers are undelimited (unquoted and contain no special characters).
- Update all object associations related to a driver together.

  IMPORTANT: It is extremely important that you update all object associations related to a driver together.

  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.

  NOTE: 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 be insecure.

  NOTE: 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*\jdbc\util directory, a batch file `association.bat` or shell script `association.sh` (depending upon your platform) starts the utility.

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

Database	Properties Filename
IBM DB2 Universal Database	properties_db2.txt
Informix Dynamic Server	properties_ifx_ansi.txt[1] 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

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

NOTE: For more information on how to run the utility from the command line, refer to run.bat in the *install-dir*\tools\util directory.

1 Stop the driver.

2 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.

4 Run the Association Utility to normalize associations (operations 4 and 5).

Editing Associations

The Association Utility requires two parameters (*oldRDN* and *newRDN*) for operations 6 and 7, which 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:

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=*`**

9 Uninstalling the IDM Driver for JDBC

- "Deleting IDM Driver Objects" on page 127
- "Running the Product Uninstaller" on page 127
- "Executing Database Uninstallation Scripts" on page 127

IMPORTANT: We recommend that you 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 IDM Driver Objects

When deleting Novell® eDirectory™ 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 Novell iManager, click DirXML Management > Overview.
2. From Overview, locate the driver set where the driver exists, then click Delete Driver.
3. Click the Driver you want to delete, then click OK.

Running the Product Uninstaller

Uninstallation procedures vary by platform.

To uninstall the Identity Manager Driver for JDBC on Windows, use Add or Remove Programs in the Control Panel.

Executing Database Uninstallation Scripts

This section helps you execute database uninstallation SQL scripts.

- "IBM DB2 Universal Database (UDB) Installation" on page 29
- "Informix Dynamic Server (IDS) Installation" on page 29
- "Microsoft SQL Server Installation" on page 30
- "MySQL Uninstallation" on page 128
- "Oracle Installation" on page 30
- "PostgreSQL Installation" on page 31
- "Sybase Adaptive Server Enterprise (ASE) Installation" on page 31

IBM DB2 Universal Database (UDB) Uninstallation

The directory context for DB2 is *install-dir*\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 script `uninstall.sql`.

   For example:
   ```
   db2 -f uninstall.sql
   ```

   IMPORTANT: This script won't execute in the Command Center interface beyond version 7. It 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*\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.`

   NOTE: 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 if you haven't already done so.
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*\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`.

   NOTE: The execute hotkey in Query Analyzer is F5.

MySQL Uninstallation

The directory context for MySQL SQL scripts is *install-dir*\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 uninstallation script `uninstall.sql`.

For example:

```
mysql> \. c:\uninstall.sql
```

TIP: Don't use a semicolon to terminate this statement.

Oracle Uninstallation

The directory context for Oracle SQL scripts is *install-dir*\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.`

NOTE: 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*\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 from 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*\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.

A Best Practices

The following section lists important best practices for using the Driver for JDBC. You can find additional information in Chapter 4, “Configuring the Driver for JDBC,” on page 35 and Chapter 5, “Advanced Configuration,” on page 69.

- 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 eDirectory™ 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.
- If foreign key columns link child and parent tables, omit the columns from publication triggers.
- If primary key columns are static (they do not change), do not include them in publication triggers.
- In a production environment, turn off tracing.
- 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.
- For performance and security reasons, run the driver remotely whenever possible.

B FAQ

- "Why can't the driver see my tables or views?" on page 133
- "How do I synchronize with tables located in multiple schemas?" on page 133
- "Why isn't the driver processing rows in the event log table?" on page 134
- "Can the driver manage database user accounts?" on page 134
- "Can the driver synchronize large binary and string data types?" on page 134
- "Why is publication so slow?" on page 134
- "Can the driver synchronize multiple classes?" on page 134
- "Does the driver support encrypted transport?" on page 135
- "How do I map multivalue attributes to single-value database fields?" on page 135
- "Why is the driver synchronizing garbage strings?" on page 135

Why can't the driver see my tables or views?

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 `SELECT` privilege.

How do I synchronize with tables located in multiple schemas?

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 Name parameter.

  See "Table/View Name" on page 43.

Why isn’t the driver processing rows in the event log table?

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 64.

  When 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 using the Authentication ID parameter. See “Authentication ID” on page 38.

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.

Can the driver manage database user accounts?

Yes. You can manage database accounts by using embedded SQL. For more information, see “Embedding SQL Statements in XDS Events” on page 89.

Can the driver synchronize large binary and string data types?

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 84.

Why is publication so slow?

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 63.

Can the driver synchronize multiple classes?

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.

Does the driver support encrypted transport?

No. How the driver communicates with a given database depends upon the third-party driver being used. Some third-party drivers support encrypted transport, while 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 Driver for JDBC and your third-party driver. This method allows both the Driver for JDBC and the third-party driver to run locally on the database server. Then all data traveling across the network between the metadirectory engine and the Driver for JDBC 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.

How do I map multivalue attributes to single-value database fields?

See "Mapping Multivalue Attributes to Single-Value Database Fields" on page 79.

Why is the driver synchronizing garbage strings?

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 the Character Encoding Values (http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html) defined by Sun.

C Supported Data Types

The Identity Manager Driver for JDBC 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`

D 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.

Required JDBC 1 methods:

- 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
- 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

E Utilized JDBC 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 driver for JDBC 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.

1. java.sql.DriverManager (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html)
2. java.sql.CallableStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html)
3. java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html)
4. java.sql.PreparedStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html)
5. java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html)
6. java.sql.ResultSetMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html)
7. java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html)
8. 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 Driver for JDBC uses:

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]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 Driver for JDBC uses:

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 driver for JDBC uses:

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 Driver for JDBC uses:

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
setInt(int parameterIndex, int x):void	1	yes

Method Signature	JDBC Version	Required?
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 Driver for JDBC uses:

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
getMetaData():java.sql.ResultSetMetaData	1	no
getShort(int columnIndex):short	1	yes

Method Signature	JDBC Version	Required?
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 Driver for JDBC uses:

Method Signature	JDBC Version	Required?
getColumnCount():int	1	no
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 Driver for JDBC uses:

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
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

Method Signature	JDBC Version	Required?
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 Driver for JDBC uses:

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

F 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 (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 (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 (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 retry (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)>
```

G 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 (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 (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 (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 retry (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)>
```

H 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)>
```

I 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)>
```

J Documentation Updates

This section contains new or updated information on the Identity Manager Driver for JDBC.

The documentation is provided on the Web in two formats: HTML and PDF. The HTML and PDF documentation are both kept up-to-date with the documentation changes listed in this section.

If you need to know whether a copy of the PDF documentation you are using is the most recent, check the date that the PDF file was published. The date is on the title page.

New or updated documentation was published on the following dates:

- "May 3, 2006" on page 155

May 3, 2006

Updated three broken links found in the document.