

Driver for JDBC Implementation Guide

Novell® Identity Manager

4.0

October 15, 2010

www.novell.com



Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. For more information on exporting Novell software, see the [Novell International Trade Services Web page \(http://www.novell.com/info/exports/\)](http://www.novell.com/info/exports/). Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 2008-2010 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

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

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

Novell Trademarks

For a list of Novell trademarks, see [Trademarks \(http://www.novell.com/company/legal/trademarks/tmlist.html\)](http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide	11
1 Introducing the Identity Manager Driver for JDBC	13
1.1 Driver Concepts	13
1.1.1 JDBC	13
1.1.2 Identity Manager JDBC driver	14
1.1.3 Third-Party JDBC Driver	14
1.1.4 Identity Vault	14
1.1.5 Directory Schema	14
1.1.6 Application Schema	15
1.1.7 Database Schema	15
1.1.8 Synchronization Schema	15
1.1.9 Logical Database Class	15
1.1.10 XDS	15
1.2 Database Concepts	15
1.2.1 Structured Query Language	16
1.2.2 Data Manipulation Language	16
1.2.3 Data Definition Language	16
1.2.4 View	16
1.2.5 Identity Columns/Sequences	17
1.2.6 Transaction	17
1.2.7 Stored Procedures or Functions	18
1.2.8 Trigger	18
1.2.9 Instead-Of-Trigger	19
1.3 Driver Features	20
1.3.1 Local and Remote Platforms	20
1.3.2 Password Synchronization	21
1.3.3 Data Synchronization Models	21
1.3.4 Triggerless vs. Triggered Publication	24
2 Installing the Driver Files	27
2.1 Installing the Driver Files	27
2.2 Installing JDBC Driver Jar Files	27
3 Installing and Configuring Database Objects	29
3.1 SQL Script Conventions	29
3.2 Installing IBM DB2 Universal Database (UDB)	31
3.3 Installing Informix Dynamic Server (IDS)	32
3.4 Installing Microsoft SQL Server	32
3.5 Installing MySQL	33
3.6 Installing Oracle	33
3.7 Installing PostgreSQL 7	33
3.8 Installing PostgreSQL 8	34
3.9 Installing Sybase Adaptive Server Enterprise (ASE)	34
3.10 Testing the Database Object Installation	35

4	Upgrading an Existing Driver	37
4.1	Supported Upgrade Paths	37
4.2	What's New in Version 4.0	37
4.3	Upgrade Procedure	37
5	Importing an Example JDBC Configuration File	39
5.1	Using Designer to Import	39
5.2	Using iManager to Import	39
5.3	JDBC Driver Settings	40
6	Configuring the JDBC Driver	43
6.1	Smart Configuration	43
6.1.1	Specifying Custom Descriptor Files	43
6.1.2	Reserved Filenames for Descriptor Files	44
6.1.3	Import Descriptor Files	44
6.1.4	Descriptor File Locations	44
6.1.5	Precedence	45
6.1.6	Custom Descriptor Best Practices	45
6.1.7	Descriptor File DTDs	45
6.2	Configuration Parameters	45
6.2.1	Viewing Driver Parameters	45
6.2.2	Deprecated Parameters	46
6.2.3	Authentication Parameters	46
6.3	Driver Parameters	47
6.3.1	Uncategorized Parameters	49
6.3.2	Database Scoping Parameters	52
6.3.3	Connectivity Parameters	57
6.3.4	Compatibility Parameters	59
6.4	Subscription Parameters	68
6.4.1	Uncategorized Parameters	69
6.4.2	Primary Key Parameters	71
6.5	Publication Parameters	77
6.5.1	Uncategorized Parameters	77
6.5.2	Triggered Publication Parameters	80
6.5.3	Triggerless Publication Parameters	82
6.5.4	Polling Parameters	83
6.6	Trace Levels	86
6.7	Configuring Third-Party JDBC Drivers	86
6.8	Configuring jTDS Support for the JDBC Driver	87
7	Managing the Driver	89
8	Schema Mapping	91
8.1	High-Level View	91
8.2	Logical Database Classes	91
8.3	Indirect Synchronization	91
8.3.1	Mapping eDirectory Classes to Logical Database Classes	92
8.3.2	Parent Tables	94
8.3.3	Parent Table Columns	94
8.3.4	Child Tables	95
8.3.5	Referential Attributes	96

8.3.6	Single-Value Referential Attributes	96
8.3.7	Multivalue Referential Attributes	97
8.4	Direct Synchronization	99
8.4.1	View Column Meta-Identifiers	99
8.4.2	Primary Key Columns	102
8.4.3	Schema Mapping	102
8.5	Synchronizing Primary Key Columns	102
8.6	Synchronizing Multiple Classes	102
8.7	Mapping Multivalue Attributes to Single-Value Database Fields	103
9	Mapping XDS Events to SQL Statements	105
9.1	Mapping XDS Events for Indirect Synchronization	105
9.2	Mapping XDS Events for Direct Synchronization	106
10	The Event Log Table	107
10.1	Event Log Columns	107
10.1.1	record_id	107
10.1.2	table_key	107
10.1.3	status	108
10.1.4	event_type	108
10.1.5	event_time	109
10.1.6	perpetrator	109
10.1.7	table_name	109
10.1.8	column_name	109
10.1.9	old_value	109
10.1.10	new_value	110
10.2	Event Types	110
11	Embedded SQL Statements in XDS Events	119
11.1	Common Uses of Embedded SQL	120
11.2	Embedded SQL Basics	120
11.2.1	Elements	120
11.2.2	Namespaces	120
11.2.3	Embedded SQL Example	120
11.3	Token Substitution	121
11.4	Virtual Triggers	124
11.5	Manual vs. Automatic Transactions	125
11.6	Transaction Isolation Level	126
11.7	Statement Type	126
11.8	SQL Queries	128
11.9	Data Definition Language (DDL) Statements	129
11.10	Logical Operations	130
11.11	Implementing Password Set with Embedded SQL	130
11.12	Implementing Modify Password with Embedded SQL	131
11.13	Implementing Check Object Password	132
11.14	Calling Stored Procedures and Functions	132
11.14.1	Using Embedded SQL to Call Stored Procedures or Functions	132
11.14.2	Using the jdbc:call-procedure Element	133
11.14.3	Using the jdbc:call-function Element	137
11.15	Best Practices	141

12 Supported Databases	143
12.1 Database Interoperability	143
12.2 Supported Databases	143
12.3 Database Characteristics	144
12.3.1 Database Features	145
12.3.2 Current Time Stamp Statements	145
12.3.3 Syntaxes for Calling Stored Procedures and Functions	146
12.3.4 Left Outer Join Operators	146
12.3.5 Undelimited Identifier Case Sensitivity	147
12.3.6 Supported Transaction Isolation Levels	147
12.3.7 Commit Keywords	148
12.3.8 IBM DB2 Universal Database (UDB)	148
12.3.9 Informix Dynamic Server (IDS)	149
12.3.10 Microsoft SQL Server	150
12.3.11 MySQL	151
12.3.12 Oracle	152
12.3.13 PostgreSQL	153
12.3.14 Sybase Adaptive Server Enterprise (ASE)	153
 13 Third-Party JDBC Drivers	 155
13.1 Third-Party JDBC Driver Interoperability	155
13.2 Third-Party JDBC Driver Types	155
13.2.1 Driver Types	155
13.2.2 Which Type To Use?	156
13.3 Third-Party Jar File Placement	156
13.4 Supported Third-Party JDBC Drivers (Recommended)	157
13.4.1 Third-Party JDBC Driver Features	157
13.4.2 JDBC URL Syntaxes	157
13.4.3 JDBC Driver Class Names	158
13.4.4 IBM DB2 Universal Database Type 4 JDBC Driver	158
13.4.5 Informix JDBC Driver	160
13.4.6 jTDS JDBC Driver	161
13.4.7 MySQL Connector/J JDBC Driver	163
13.4.8 Oracle Thin Client JDBC Driver	164
13.4.9 Oracle OCI JDBC Driver	166
13.4.10 PostgreSQL JDBC Driver	167
13.4.11 Sybase Adaptive Server Enterprise JConnect JDBC Driver	168
13.5 Supported Third-Party JDBC Drivers (Not Recommended)	169
13.5.1 Third-Party JDBC Driver Features	169
13.5.2 JDBC URL Syntaxes	169
13.5.3 JDBC Driver Class Names	170
13.5.4 IBM DB2 Universal Database JDBC Driver	170
13.5.5 Microsoft SQL Server 2000 Driver for JDBC	172
13.5.6 Microsoft SQL Server 2005 JDBC Driver	174
13.6 Deprecated Third-Party JDBC Drivers	175
13.7 Other Third-Party JDBC Drivers	175
13.7.1 IBM Toolbox for Java/JTOpen	175
13.7.2 Minimum Third-Party JDBC Driver Requirements	175
13.7.3 Considerations When Using Other Third-Party JDBC Drivers	176
13.8 Security Issues	176
 14 The Association Utility	 177
14.1 Independent Operations	177
14.2 Before You Begin	178

14.3	Using the Association Utility	178
14.4	Parameters for Searching and Replacing	179
15	Troubleshooting the JDBC Driver	181
15.1	Recognizing Publication Events	181
15.2	Executing Test Scripts	181
15.3	Troubleshooting Driver Processes	181
A	Uninstalling the Driver	183
A.1	Deleting Identity Manager Driver Objects	183
A.2	Running the Product Uninstaller	183
A.3	Executing Database Uninstallation Scripts	183
A.3.1	IBM DB2 Universal Database (UDB) Uninstallation.	184
A.3.2	Informix Dynamic Server (IDS) Uninstallation	184
A.3.3	Microsoft SQL Server Uninstallation	184
A.3.4	MySQL Uninstallation	185
A.3.5	Oracle Uninstallation	185
A.3.6	PostgreSQL Uninstallation.	185
A.3.7	Sybase Adaptive Server Enterprise (ASE) Uninstallation	186
B	Known Issues and Limitations	187
B.1	Known Issues.	187
B.2	Limitations	187
C	Best Practices	189
D	FAQ	191
D.1	Can't See Tables or Views.	191
D.2	Synchronizing with Tables	191
D.3	Processing Rows in the Event Log Table	191
D.4	Managing Database User Accounts.	192
D.5	Synchronizing Large Data Types.	192
D.6	Slow Publication.	192
D.7	Synchronizing Multiple Classes	192
D.8	Encrypted Transport.	193
D.9	Mapping Multivalued Attributes	193
D.10	Synchronizing Garbage Strings	193
D.11	Running Multiple JDBC Driver Instances	193

E	Supported Data Types	195
F	java.sql.DatabaseMetaData Methods	197
G	JDBC Interface Methods	199
H	Third-Party JDBC Driver Descriptor DTD	205
I	Third-Party JDBC Driver Descriptor Import DTD	207
J	Database Descriptor DTD	209
K	Database Descriptor Import DTD	211
L	Policy Example: Triggerless Future Event Processing	213
M	Setting Up an OCI Client on Linux	215
M.1	Downloading the Instant Client	215
M.2	Setting Up the OCI Client.	215
M.3	Configuring the OCI Driver.	216
N	Sybase Chain Modes and the Identity Manager JDBC driver	217
N.1	Error Codes	217
N.2	Procedures and Modes	218
N.2.1	Using Stored Procedure sp_proxmode	218
N.2.2	Chained and Unchained Modes	218
N.2.3	Managing Transactions in a Policy	219
N.2.4	Useful Links	219

About This Guide

The 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.

- ♦ [Chapter 1, “Introducing the Identity Manager Driver for JDBC,” on page 13](#)
- ♦ [Chapter 2, “Installing the Driver Files,” on page 27](#)
- ♦ [Chapter 3, “Installing and Configuring Database Objects,” on page 29](#)
- ♦ [Chapter 4, “Upgrading an Existing Driver,” on page 37](#)
- ♦ [Chapter 5, “Importing an Example JDBC Configuration File,” on page 39](#)
- ♦ [Chapter 6, “Configuring the JDBC Driver,” on page 43](#)
- ♦ [Chapter 7, “Managing the Driver,” on page 89](#)
- ♦ [Chapter 8, “Schema Mapping,” on page 91](#)
- ♦ [Chapter 9, “Mapping XDS Events to SQL Statements,” on page 105](#)
- ♦ [Chapter 10, “The Event Log Table,” on page 107](#)
- ♦ [Chapter 11, “Embedded SQL Statements in XDS Events,” on page 119](#)
- ♦ [Chapter 12, “Supported Databases,” on page 143](#)
- ♦ [Chapter 13, “Third-Party JDBC Drivers,” on page 155](#)
- ♦ [Chapter 14, “The Association Utility,” on page 177](#)
- ♦ [Chapter 15, “Troubleshooting the JDBC Driver,” on page 181](#)
- ♦ [Appendix A, “Uninstalling the Driver,” on page 183](#)
- ♦ [Appendix B, “Known Issues and Limitations,” on page 187](#)
- ♦ [Appendix C, “Best Practices,” on page 189](#)
- ♦ [Appendix D, “FAQ,” on page 191](#)
- ♦ [Appendix E, “Supported Data Types,” on page 195](#)
- ♦ [Appendix F, “java.sql.DatabaseMetaData Methods,” on page 197](#)
- ♦ [Appendix G, “JDBC Interface Methods,” on page 199](#)
- ♦ [Appendix H, “Third-Party JDBC Driver Descriptor DTD,” on page 205](#)
- ♦ [Appendix I, “Third-Party JDBC Driver Descriptor Import DTD,” on page 207](#)
- ♦ [Appendix J, “Database Descriptor DTD,” on page 209](#)
- ♦ [Appendix K, “Database Descriptor Import DTD,” on page 211](#)
- ♦ [Appendix L, “Policy Example: Triggerless Future Event Processing,” on page 213](#)
- ♦ [Appendix M, “Setting Up an OCI Client on Linux,” on page 215](#)
- ♦ [Appendix N, “Sybase Chain Modes and the Identity Manager JDBC driver,” on page 217](#)

Audience

This guide is for Novell eDirectory and Identity Manager administrators who are using the Identity Manager Driver for JDBC.

Feedback

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.

Documentation Updates

For the most recent version of this document, see the [Identity Manager 4.0 Drivers Documentation Web site \(http://www.novell.com/documentation/idm40drivers/index.html\)](http://www.novell.com/documentation/idm40drivers/index.html).

Additional Documentation

For documentation on using Identity Manager and the other drivers, see the [Identity Manager 4.0 Documentation Web site \(http://www.novell.com/documentation/idm40drivers/index.html\)](http://www.novell.com/documentation/idm40drivers/index.html).

Introducing the Identity Manager Driver for JDBC

1

The Identity Manager Driver for Java DataBase Connectivity (JDBC) provides a generic solution for synchronizing data between Identity Manager and JDBC-accessible relational databases.

The principal value of this driver resides in its generic nature. Unlike most drivers that interface with a single application, this driver can interface with most relational databases and database-hosted applications.

- ♦ [Section 1.1, “Driver Concepts,” on page 13](#)
- ♦ [Section 1.2, “Database Concepts,” on page 15](#)
- ♦ [Section 1.3, “Driver Features,” on page 20](#)

1.1 Driver Concepts

- ♦ [Section 1.1.1, “JDBC,” on page 13](#)
- ♦ [Section 1.1.2, “Identity Manager JDBC driver,” on page 14](#)
- ♦ [Section 1.1.3, “Third-Party JDBC Driver,” on page 14](#)
- ♦ [Section 1.1.4, “Identity Vault,” on page 14](#)
- ♦ [Section 1.1.5, “Directory Schema,” on page 14](#)
- ♦ [Section 1.1.6, “Application Schema,” on page 15](#)
- ♦ [Section 1.1.7, “Database Schema,” on page 15](#)
- ♦ [Section 1.1.8, “Synchronization Schema,” on page 15](#)
- ♦ [Section 1.1.9, “Logical Database Class,” on page 15](#)
- ♦ [Section 1.1.10, “XDS,” on page 15](#)

1.1.1 JDBC

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

Most enterprise database vendors provide a unique implementation of the JDBC interface. 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 JDBC driver primarily uses the JDBC 1 interface. It uses a small subset of JDBC 2 or JDBC 3 methods when supported by third-party JDBC drivers.

1.1.2 Identity Manager JDBC driver

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

The driver consists of 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.

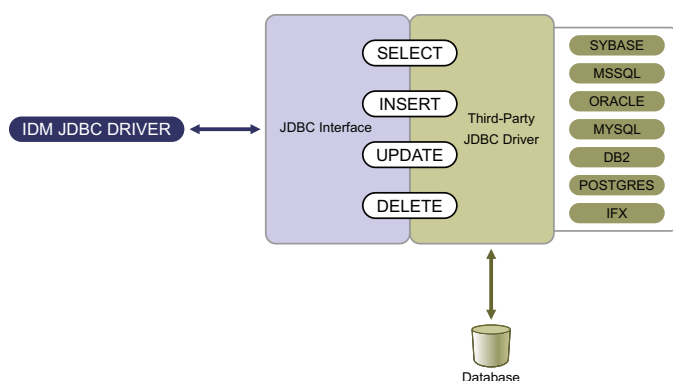
1.1.3 Third-Party JDBC Driver

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

For example, `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 JDBC driver and third-party JDBC drivers.

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



1.1.4 Identity Vault

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

1.1.5 Directory Schema

The directory schema is 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.

1.1.6 Application Schema

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

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

1.1.7 Database Schema

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

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

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

1.1.8 Synchronization Schema

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

1.1.9 Logical Database Class

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

1.1.10 XDS

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

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

1.2 Database Concepts

- ♦ [Section 1.2.1, “Structured Query Language,” on page 16](#)
- ♦ [Section 1.2.2, “Data Manipulation Language,” on page 16](#)
- ♦ [Section 1.2.3, “Data Definition Language,” on page 16](#)
- ♦ [Section 1.2.4, “View,” on page 16](#)
- ♦ [Section 1.2.5, “Identity Columns/Sequences,” on page 17](#)
- ♦ [Section 1.2.6, “Transaction,” on page 17](#)
- ♦ [Section 1.2.7, “Stored Procedures or Functions,” on page 18](#)
- ♦ [Section 1.2.8, “Trigger,” on page 18](#)
- ♦ [Section 1.2.9, “Instead-Of-Trigger,” on page 19](#)

1.2.1 Structured Query Language

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

1.2.2 Data Manipulation Language

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

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

The following example shows several DML statements:

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

1.2.3 Data Definition Language

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

DDL statements are proprietary and differ substantially between databases. Even though the JDBC driver is DML-based, you can embed DDL statements in XDS events. For additional information, refer to [Chapter 11, “Embedded SQL Statements in XDS Events,”](#) on page 119,

The following examples show several DDL statements:

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

CREATE USER idm IDENTIFIED BY novell;
```

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

1.2.4 View

A view is a logical table.

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


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

1.2.5 Identity Columns/Sequences

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

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

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

The following is an example of a sequence object:

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

1.2.6 Transaction

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

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

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

- ♦ [“Manual \(User-Defined\) Transactions” on page 17](#)
- ♦ [“Automatic Transactions” on page 18](#)

Manual (User-Defined) Transactions

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

The following example illustrates a manual transaction:

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

Automatic Transactions

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

The following example illustrates an automatic transaction:

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

1.2.7 Stored Procedures or Functions

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

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

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

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

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

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

1.2.8 Trigger

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

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

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

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

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

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

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

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

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

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

1.2.9 Instead-Of-Trigger

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

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

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

```

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

```

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

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

```

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

```

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

```

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

```

In this example, the statements happen to be equivalent.

1.3 Driver Features

- ♦ [Section 1.3.1, “Local and Remote Platforms,” on page 20](#)
- ♦ [Section 1.3.2, “Password Synchronization,” on page 21](#)
- ♦ [Section 1.3.3, “Data Synchronization Models,” on page 21](#)
- ♦ [Section 1.3.4, “Triggerless vs. Triggered Publication,” on page 24](#)

1.3.1 Local and Remote Platforms

The driver runs on all platforms supported for Identity Manager 4.0, including any local installation (Metadirectory server) or remote installation (Remote Loader). For information about supported platforms for Identity Manager 4.0, see [“System Requirements”](#) in the *Identity Manager 4.0 Integrated Installation Guide*.

For information on supported databases, see [“Database Interoperability” on page 143](#).

For information on supported third-party JDBC drivers, see [“Third-Party JDBC Driver Interoperability” on page 155](#).

1.3.2 Password Synchronization

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

1.3.3 Data Synchronization Models

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

Model	Association	Description
Direct	Usually associated with views	Views provide the abstraction mechanism that best facilitates integration with existing customer tables.
Indirect	Usually associated with tables	Customer tables probably don't match the structure required by the driver. Therefore, it's usually necessary to create intermediate staging tables that do match the structure that the driver requires. Although the structures might match, it is highly unlikely.

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

- ♦ [“Indirect Synchronization” on page 21](#)
- ♦ [“Direct Synchronization” on page 22](#)

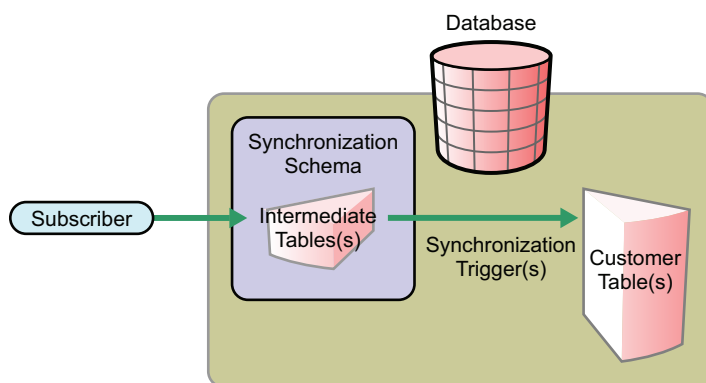
Indirect Synchronization

Indirect synchronization uses intermediate staging tables to synchronize data between the 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

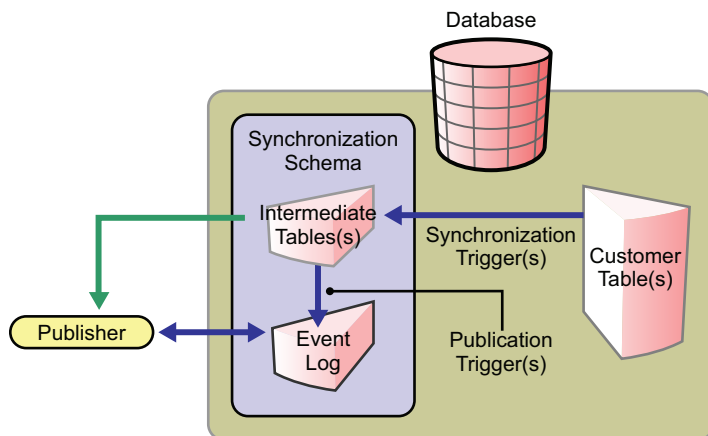
Figure 1-2 *Indirect Synchronization on the Subscriber Channel*



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

Figure 1-3 *Indirect Synchronization on the 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 channel then reads the inserted rows and updates the Identity Vault.

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

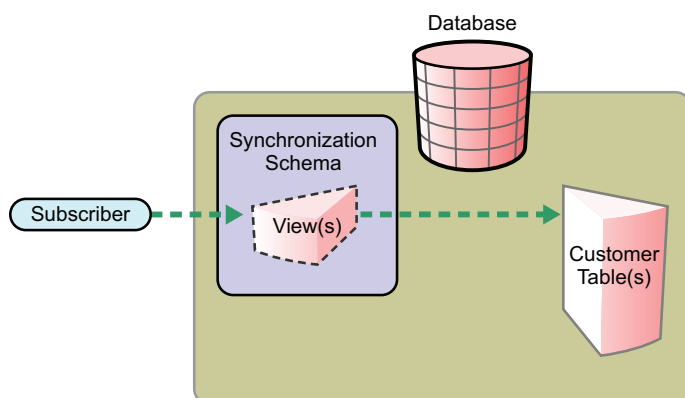
Direct Synchronization

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

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 views or tables.

Subscriber Channel

Figure 1-4 Direct Synchronization on the Subscriber Channel

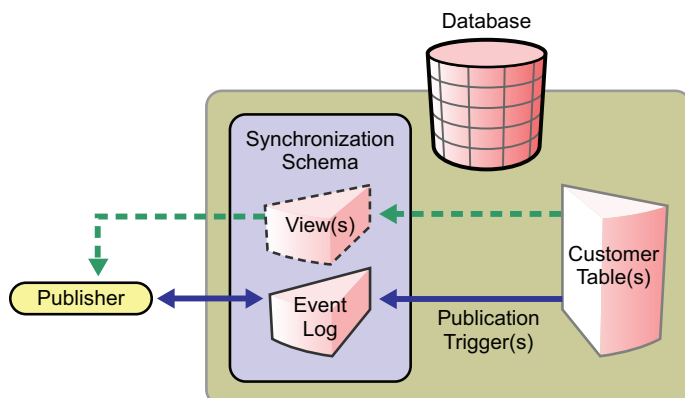


The Subscriber channel updates existing customer tables through a view in the synchronization schema.

Direct synchronization without a view is possible only if customer tables match the structure that the JDBC driver requires. For additional information, see [Section 8.3, “Indirect Synchronization,” on page 91](#).

Publisher Channel

Figure 1-5 Direct Synchronization on the 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.

1.3.4 Triggerless vs. Triggered Publication

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

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

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

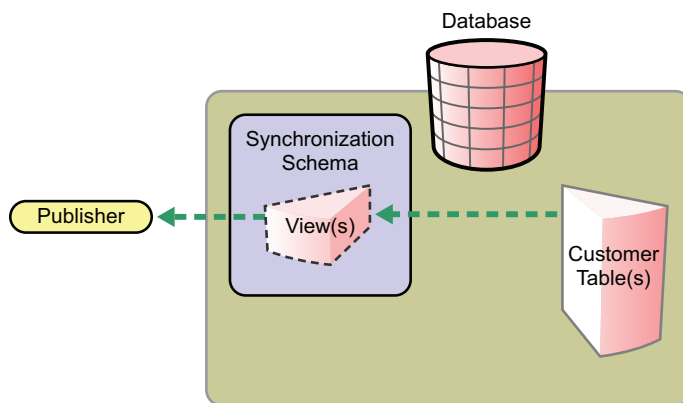
Triggerless publication, unlike triggered publication, does not preserve event order. It only guarantees that by the end of a polling cycle, objects in the database and the Identity Vault are in sync.

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

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

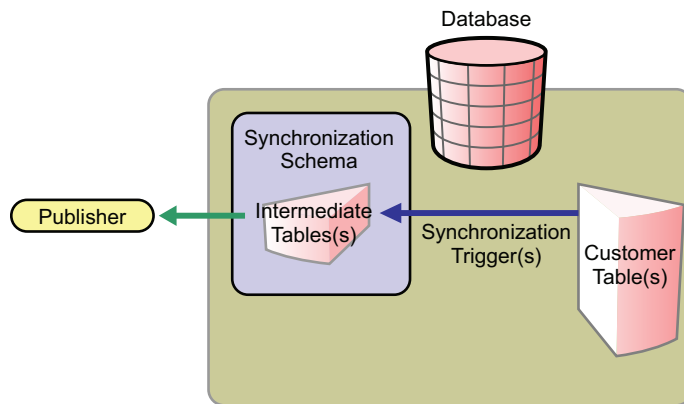
The following figure illustrates direct triggerless publication:

Figure 1-6 *Direct Triggerless Publication*



The following figure illustrates indirect triggerless publication:

Figure 1-7 *Indirect Triggerless Publication*



If you move the driver without moving the state files, the driver must build up new state files by resynchronizing. For information on this situation, see [“State Directory” on page 51](#).

Installing the Driver Files

By default, the JDBC driver files are installed on the Metadirectory server at the same time as the Metadirectory engine. The installation program extends the Identity Vault's schema and installs the driver shim and a driver configuration file. It does not create the driver in the Identity Vault (see [Chapter 5, “Importing an Example JDBC Configuration File,” on page 39](#)) or upgrade an existing driver's configuration (see [Chapter 4, “Upgrading an Existing Driver,” on page 37](#)).

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

- ♦ [Section 2.1, “Installing the Driver Files,” on page 27](#)
- ♦ [Section 2.2, “Installing JDBC Driver Jar Files,” on page 27](#)

For information about uninstalling the driver, see [Chapter A, “Uninstalling the Driver,” on page 183](#).

2.1 Installing the Driver Files

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

- ♦ On a local machine: Install the JDBC driver files on the Metadirectory server and connect to the database by using the Provider URL (Connection Properties). For information on installing the Metadirectory server, see [“Installing Identity Manager” in the *Identity Manager 4.0 Integrated Installation Guide*](#).
- ♦ On a remote machine: Install the JDBC driver files on the Remote Loader. For information on installing the Remote Loader, see [“Installing Identity Manager” in the *Identity Manager 4.0 Integrated Installation Guide*](#).

2.2 Installing JDBC Driver Jar Files

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

- 1 Locate the appropriate JDBC driver jar files.

Information about the jar files you need and where to download them from is found in [Section 13.4, “Supported Third-Party JDBC Drivers \(Recommended\),” on page 157](#).

- 2 Place the files in the appropriate location.

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

Table 2-1 *Locations for JAR Files: Metadirectory Server*

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

Table 2-2 *Locations for JAR Files: Remote Loader*

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

Installing and Configuring Database Objects

3

You need to install and configure database objects (for example, tables, triggers, and indexes) for synchronization with the sample driver configuration. If you don't configure database objects, the sample configuration file won't work.

- ♦ [Section 3.1, “SQL Script Conventions,” on page 29](#)
- ♦ [Section 3.2, “Installing IBM DB2 Universal Database \(UDB\),” on page 31](#)
- ♦ [Section 3.3, “Installing Informix Dynamic Server \(IDS\),” on page 32](#)
- ♦ [Section 3.4, “Installing Microsoft SQL Server,” on page 32](#)
- ♦ [Section 3.5, “Installing MySQL,” on page 33](#)
- ♦ [Section 3.6, “Installing Oracle,” on page 33](#)
- ♦ [Section 3.7, “Installing PostgreSQL 7,” on page 33](#)
- ♦ [Section 3.8, “Installing PostgreSQL 8,” on page 34](#)
- ♦ [Section 3.9, “Installing Sybase Adaptive Server Enterprise \(ASE\),” on page 34](#)
- ♦ [Section 3.10, “Testing the Database Object Installation,” on page 35](#)

3.1 SQL Script Conventions

The following table lists default locations for SQL scripts:

Table 3-1 *Default Locations for SQL Scripts*

Platform	Default Location
Windows	<ul style="list-style-type: none">♦ If the Metadirectory server is installed through Identity Manager integrated installer, the default location is <code>c:\Novell\IdentityManager\NDS\DirXMLUtilities\jdbc\sql\databaseabbreviation.</code>♦ If the Metadirectory server is installed through Identity Manager framework installer, the default location is <code>c:\novell\NDS\DirXMLUtilities\jdbc\sql\databaseabbreviation.</code>
UNIX or Linux	<code>/opt/novell/eDirectory/lib/dirxml/rules/jdbc/sql/databaseabbreviation</code>

For example, when the scripts are installed on a SUSE Linux Enterprise Server with eDirectory, the DB2 scripts are found in `opt/novell/eDirectory/lib/dirxml/rules/jdbc/db2/*`.

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

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

Table 3-2 *Identifier Abbreviations and Meanings*

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

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

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

Table 3-3 *Identifier Naming Conventions*

Database Object	Naming Convention	Examples
stored procedure/ function	<code>proc_procedure-or-function-name</code>	<code>proc_idu</code>
index	<code>idx_unqualified-table-name_sequence-number</code>	<code>idx_indirectlog_1</code>
trigger	<code>tgr_unqualified-table-name_triggering-statement-type_sequence-number</code>	<code>tgr_usr_i_1</code>
primary key constraint	<code>pk_unqualified-table-name_column-name</code>	<code>pk_usr_idu</code>
foreign key constraint	<code>fk_unqualified-table-name_column-name</code>	<code>fk_usr_idu</code>
check constraint	<code>chk_unqualified-table-name_column-name</code>	<code>chk_usr_idu</code>

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. This action can cause large gaps in identity column or sequence values.

3.2 Installing IBM DB2 Universal Database (UDB)

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

Because the process to create user accounts differs between operating systems, Step 1 below is OS-specific. These instructions are for a Windows NT operating environment. If you rerun the SQL scripts, repeat only Steps 2 through 5.

The directory context for DB2 is `install-dir\DirXMLUtilities\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 Adjust the file path to `idm_db2.jar` in the `1_install.sql` installation script. The file path to `idm_db2.jar` should reflect the location of this file on your client machine.
- 3 Execute the `1_install.sql` script from the Command Line Processor (CLP.)

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

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

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

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

3.3 Installing Informix Dynamic Server (IDS)

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

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

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

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

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

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

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

NOTE: The remaining instructions are OS-independent.

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

By default, the password for the `informix` user is `informix`. If you execute scripts as a user other than `informix`, change all references to `informix` in the scripts prior to execution.
- 4 Open and execute `1_install_9.sql` from either the `ansi` (transactional, ANSI-compliant), `log` (transactional, non-ANSI-compliant), or `no_log` (non-transactional, non-ANSI-compliant) subdirectory, depending upon which type of database you want to create.
- 5 For version 10 or later, open and execute `2_install_10.sql` from either the `ansi` (transactional, ANSI-compliant), `log` (transactional, non-ANSI-compliant), or `no_log` (non-transactional, non-ANSI-compliant) subdirectory, depending upon which type of database you want to create.

3.4 Installing Microsoft SQL Server

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

- 1 Start a client such as Query Analyzer (7, 2000) or Microsoft SQL Server Management Studio (2005).
- 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 2000 (8), execute `1_install_2k.sql`.
For version 2005 (9), execute `1_install_2005.sql`.

NOTE: The execute hotkey in Query Analyzer is F5.

3.5 Installing MySQL

The directory context for MySQL SQL scripts is *install-dir\DirXMLUtilities\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 MySQL 3 or 4, use the scripts in subdirectory `3or4`. For version 5.0.27 use the scripts in subdirectory 5.

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

Don't use a semicolon to terminate this statement.

3.6 Installing Oracle

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

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

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

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

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

3.7 Installing PostgreSQL 7

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

- 1 Create the database `idm`.

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

```
./createdb idm
```

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

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

```
./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 following command:

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

By default, the Postgres user has no password.

- 4 From inside `psql`, execute the script `1_install_7.sql`. For example:

```
idm=# \i 1_install_7.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.

3.8 Installing PostgreSQL 8

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

1 Create the `idm` database.

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

```
./createdb idm
```

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

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

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

By default, the Postgres user has no password.

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

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

4 Update the `pg_hba.conf` file.

As of version 8, this can be done through pgAdminIII. After you start, go to *Tools > Connect* to connect to the server, select the IDM database, then go to *Tools > Server Configuration > pg_hba.conf*. In the pgAdminIII `pg_hba.conf` editor, the IP-ADDRESS and IP-MASK columns in the file are combined into a single field: IP-Address. Place both the IP-ADDRESS and IP-MASK values in that field, separated by a single whitespace character.

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

```
# TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD# allow
driver user idm to connect to database idm
host idm idm 255.255.255.255 255.255.255.0 password
```

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

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

3.9 Installing Sybase Adaptive Server Enterprise (ASE)

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

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

- 1 From a Sybase client, such as isql, log 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.

3.10 Testing the Database Object Installation

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

Table 3-4 *Location of Database Scripts*

Database	Test SQL Scripts Location
IBM DB2 Universal Database	<i>install-dir\DirXMLUtilities\jdbc\sql\db2_udb\test</i>
Informix Dynamic Server	<i>install-dir\DirXMLUtilities\jdbc\sql\informix_ids\log\test</i> <i>install-dir\DirXMLUtilities\jdbc\sql\informix_ids\no_log\test</i> Informix ANSI test scripts are located in the <i>log\test</i> subdirectory.
Microsoft SQL Server	<i>install-dir\DirXMLUtilities\jdbc\sql\mssql\test</i>
MySQL	<i>install-dir\DirXMLUtilities\jdbc\sql\mysql\test</i>
Oracle	<i>install-dir\DirXMLUtilities\jdbc\sql\oracle\test</i>
PostgreSQL	<i>install-dir\DirXMLUtilities\jdbc\sql\postgres\test</i>
Sybase Adaptive Server Enterprise	<i>install-dir\DirXMLUtilities\jdbc\sql\sybase_ase\test</i>

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

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

- ♦ [Section 15.1, “Recognizing Publication Events,” on page 181.](#)
- ♦ [Section 15.2, “Executing Test Scripts,” on page 181.](#)

Upgrading an Existing Driver

4

The following sections provide information to help you upgrade an existing driver to version 4.0:

- ♦ [Section 4.1, “Supported Upgrade Paths,” on page 37](#)
- ♦ [Section 4.2, “What’s New in Version 4.0,” on page 37](#)
- ♦ [Section 4.3, “Upgrade Procedure,” on page 37](#)

4.1 Supported Upgrade Paths

You can upgrade from any Identity Manager 3.x version of the JDBC driver. Upgrading a pre-3.x version of the driver directly to version 4.0 is not supported.

4.2 What’s New in Version 4.0

- ♦ The Microsoft SQL 2008 R2 support has been added with 4.0 version.

4.3 Upgrade Procedure

The process for upgrading the JDBC driver is the same as for other Identity Manager drivers. For detailed instructions, see “[Overlaying the Existing Driver Configuration File with the New Driver \(Deprecated\)](#)” in the *Identity Manager 4.0 Framework Installation Guide*.

Importing an Example JDBC Configuration File

5

The JDBC driver includes an example configuration file that you can use as a starting point for creating the Driver object. When you import this file, Designer for Identity Manager or iManager creates and configures the objects and policies needed to make the driver work properly.

- [Section 5.1, “Using Designer to Import,” on page 39](#)
- [Section 5.2, “Using iManager to Import,” on page 39](#)
- [Section 5.3, “JDBC Driver Settings,” on page 40](#)

5.1 Using Designer to Import

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

The following procedure explains one of several ways to import the example configuration file:

- 1 Open a project in Designer.
- 2 In the Modeler, right-click the Driver Set object, then select *New > Driver*.
- 3 From the list, select *IDM Driver for JDBC 3.6.1*, then click *Run*.
- 4 Configure the driver by filling in the fields.
Provide information specific to your environment. See [Table 5-1 on page 40](#).
- 5 After specifying parameters, click *Finish* to import the driver.
- 6 Test the driver.
- 7 Deploy the driver into the Identity Vault.

See “[Deploying a Driver to an Identity Vault](#)” in the *Designer 4.0 for Identity Manager 4.0 Administration Guide*.

5.2 Using iManager to Import

Identity Manager provides an example configuration file. You installed this file when you installed the Identity Manager Web components on an iManager server. Think of the example configuration file as a template that you import and customize or configure for your environment.

- 1 In iManager, select *Identity Manager > Identity Manager Overview*.
- 2 Select *Driver Set > Drivers > Add Drivers*.

Where do you want to place the new drivers?

- ☒ In an existing driver set

snati_drset.novell  

- ☐ In a new driver set

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

- 3** Select *JDBC-IDM3_6_0-v4.xml*, then click *Next*.

- 4** Configure the driver by filling in the configuration parameters.

For information on the settings, see [Table 5-1 on page 40](#).

- 5** Define security equivalences by using a User object that has the rights that the driver needs to have on the server

The Admin user object is most often used for this task. However, you might want to create a DriversUser (for example) and assign security equivalence to that user. Whatever rights that the driver needs to have on the server, the DriversUser object must have the same security rights.

- 6** Identify all objects that represent administrative roles and exclude them from replication.

Exclude the security-equivalence object (for example, DriversUser) that you specified in Step 2. If you delete the security-equivalence object, you have removed the rights from the driver. Therefore, the driver can't make changes to Identity Manager.

- 7** Click *Finish*.

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. Because Sybase passwords must be at least 6 characters long, any Sybase passwords that are shorter than 6 characters are padded with the character "p." For example, John Doe's password would be Doepppp. The padding character can be adjusted in the Subscriber Command Transformation policies.

5.3 JDBC Driver Settings

Table 5-1 JDBC Driver Settings

Setting	Description
<i>Driver name</i>	The name that you want to display in the driver set.
<i>Target database</i>	That database that the driver writes to.
<i>Driver is local/remote</i>	Specifies whether the driver runs locally or remotely on a Remote Loader.

Setting	Description
<i>Synchronization model</i>	Specifies whether the driver uses views to synchronize directly to existing tables of arbitrary structure or synchronize to intermediate staging tables of a particular structure.
<i>Third-party JDBC implementation</i>	The third-party implementation that the driver uses.
<i>Data flow</i>	Specifies whether the authoritative source of data is the database, Identity Manager, or bidirectional (both the database and Identity Manager).
<i>Database host IP address</i>	The IP address of the database host.
<i>Database port</i>	Specifies the port that the driver uses to communicate with the database. If you don't provide a port number, the Driver Configuration Wizard provides a default port number for the database that you selected at install time.
<i>User container DN</i>	The Distinguished Name (complete context) of the container where the database users are published.
<i>Group container DN</i>	The Distinguished Name (complete context) of the container where the database groups are published.
<i>Publication mode</i>	Specifies whether publication is triggered (default) or triggerless

- ♦ [Section 6.1, “Smart Configuration,” on page 43](#)
- ♦ [Section 6.2, “Configuration Parameters,” on page 45](#)
- ♦ [Section 6.3, “Driver Parameters,” on page 47](#)
- ♦ [Section 6.4, “Subscription Parameters,” on page 68](#)
- ♦ [Section 6.5, “Publication Parameters,” on page 77](#)
- ♦ [Section 6.6, “Trace Levels,” on page 86](#)
- ♦ [Section 6.7, “Configuring Third-Party JDBC Drivers,” on page 86](#)
- ♦ [Section 6.8, “Configuring jTDS Support for the JDBC Driver,” on page 87](#)

6.1 Smart Configuration

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

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

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

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

- ♦ [Section 6.1.1, “Specifying Custom Descriptor Files,” on page 43](#)
- ♦ [Section 6.1.2, “Reserved Filenames for Descriptor Files,” on page 44](#)
- ♦ [Section 6.1.3, “Import Descriptor Files,” on page 44](#)
- ♦ [Section 6.1.4, “Descriptor File Locations,” on page 44](#)
- ♦ [Section 6.1.5, “Precedence,” on page 45](#)
- ♦ [Section 6.1.6, “Custom Descriptor Best Practices,” on page 45](#)
- ♦ [Section 6.1.7, “Descriptor File DTDs,” on page 45](#)

6.1.1 Specifying Custom Descriptor Files

You can force the driver to use a custom descriptor file for a database or third-party JDBC driver. To specify a custom database descriptor file, see [“Database Descriptor Filename” on page 60](#). To specify a custom third-party driver descriptor file, see [“JDBC Driver Descriptor Filename” on page 60](#). This is useful when multiple descriptor files exist for the same database or third-party JDBC driver. For the custom descriptor file to take effect, set the driver parameter as the jdbc-driver-descriptor.

6.1.2 Reserved Filenames for Descriptor Files

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

6.1.3 Import Descriptor Files

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

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

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

6.1.4 Descriptor File Locations

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

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

Table 6-1 *Where to Place Descriptors*

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

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

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

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

6.1.6 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).

6.1.7 Descriptor File DTDs

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

Table 6-2 *Where to Find Descriptor DTDs*

Descriptor Type	Appendix
Third-party JDBC driver	Appendix H, “Third-Party JDBC Driver Descriptor DTD,” on page 205
Third-party JDBC driver import	Appendix I, “Third-Party JDBC Driver Descriptor Import DTD,” on page 207
Database	Appendix J, “Database Descriptor DTD,” on page 209
Database import	Appendix K, “Database Descriptor Import DTD,” on page 211

6.2 Configuration Parameters

- ♦ [Section 6.2.1, “Viewing Driver Parameters,” on page 45](#)
- ♦ [Section 6.2.2, “Deprecated Parameters,” on page 46](#)
- ♦ [Section 6.2.3, “Authentication Parameters,” on page 46](#)

6.2.1 Viewing Driver Parameters

- 1 In iManager, click *Identity Manager* > *Identity Manager Overview*.

- 2 Locate the driver set containing the driver, then click the driver's icon and edit properties. iManager displays the driver's configuration parameters.

6.2.2 Deprecated Parameters

The following parameters have been deprecated since version 1.6:

Table 6-3 *Deprecated Parameters*

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

6.2.3 Authentication Parameters

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

- ♦ [“Authentication ID” on page 46](#)
- ♦ [“Authentication Context” on page 46](#)
- ♦ [“Application Password” on page 47](#)

Authentication ID

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

This value can be referenced in the Connection Properties parameter value via the token `{username}`. See [“Connection Properties” on page 58](#).

The default value for the sample configuration is `idm`.

Authentication Context

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

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

For a list of JDBC URL syntaxes for supported third-party drivers, see [“JDBC URL Syntaxes” on page 157](#).

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

Application Password

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

This value can be referenced in the Connection Properties parameter value via the token `{ $password }`. See [“Connection Properties” on page 58](#).

6.3 Driver Parameters

The following table summarizes all driver-level parameters and their properties:

Table 6-4 *Driver Parameters and Properties*

Display Name	Tag Name	Sample Value	Default Value	Required
Third-Party JDBC Driver Class Name	jdbc-class	oracle.jdbc.driver.OracleDriver	(none)	yes
Time Syntax	time-syntax	1 (integer)	1 (integer)	no
Synchronization Filter	sync-filter	schema (include by schema membership)	(none)	no
Schema Name	sync-schema	indirect	(none)	yes ¹
Include Filter Expression	include-table-filter	IDM_*	(none)	no
Exclude Filter Expression	exclude-table-filter	BIN\\${22}==\\${0}	(none)	no
Table/View Names	sync-tables	usr	(none)	yes ¹
Connection Initialization Statements	connection-init	USE idm	(none)	no
Use Minimal Number of Connections?	use-single-connection	0 (no)	(dynamic ³)	no
Connection Properties	connection-properties	USER={ \$username }; PASSWORD={ \$password }	(dynamic ³)	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

Display Name	Tag Name	Sample Value	Default Value	Required
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
Enable Meta-Identifier Support?	enable-meta-identifiers	1 (yes)	1 (yes)	no
Force Username Case	force-username-case	upper (to uppercase)	(none)	no
Left Outer Join Operator	left-outer-join-operator	(+)	(dynamic ³)	no
Retrieve Minimal Metadata	minimal-metadata	0 (no)	(dynamic ³)	no
Function Return Method	function-return-method	result set	(dynamic ³)	no
Supports Schemas in Metadata Retrieval?	supports-schemas-in-metadata-retrieval	1 (yes)	(dynamic ³)	no
Sort Column Names By	column-position-comparator	com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator (hexadecimal value)	(dynamic ³)	no

¹ One of these mutually exclusive parameters must be present if the Synchronization Filter parameter is not present. See [“Synchronization Filter” on page 53](#). ² This default is derived dynamically at runtime from descriptor files and database metadata. ³ This default is derived dynamically from descriptor files at runtime.

Driver parameters fall into the following subcategories:

- ♦ [Section 6.3.1, “Uncategorized Parameters,” on page 49](#)
- ♦ [Section 6.3.2, “Database Scoping Parameters,” on page 52](#)
- ♦ [Section 6.3.3, “Connectivity Parameters,” on page 57](#)
- ♦ [Section 6.3.4, “Compatibility Parameters,” on page 59](#)

6.3.1 Uncategorized Parameters

- ♦ [“Third-Party JDBC Driver Class Name” on page 49](#)
- ♦ [“Time Syntax” on page 49](#)
- ♦ [“State Directory” on page 51](#)

Third-Party JDBC Driver Class Name

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

The following table lists the properties of this parameter:

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

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

For a list of supported third-party JDBC driver classnames, see [“JDBC Driver Class Names” on page 158](#).

Time Syntax

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

- ♦ [“Return Database Time, Date, and Timestamp Values as 32-Bit Integers” on page 49](#)
- ♦ [“Return Database Time, Date, and Timestamp Values as Canonical Strings” on page 50](#)
- ♦ [“Return database Time, Date, and Timestamp Values in their Java String Representation as Returned by the Method toString\(\):java.lang.String” on page 50](#)

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

This is the default.

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

This option has two problems:

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

The second and third options overcome these two limitations.

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

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

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

Table 6-6 Database Types and Canonical String Representations

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

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

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

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

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

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

Table 6-7 Database Types and Java String Formats

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

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

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

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

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

Table 6-8 *Time Syntax: Properties*

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

State Directory

The State Directory parameter specifies where a driver instance should store state data. State data is currently used for triggerless publication. See [“Triggerless Publication Parameters” on page 82](#). State data might be used to store additional state information in the future.

Each driver instance has two state files. State filenames follow the formats `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.

State files are named to be unique. These names are not intuitive. The names begin with `jdbc_` and end in `.lg` or `.db`. The rest of the filename is a GUID value that must be looked up by using a directory browser that can display it.

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

Changes That Can Force Triggerless Publisher Resynchronization

If you delete state files, the triggerless publisher will build new state files by resynchronizing. If you move the JDBC driver without moving the state files, the triggerless publisher builds new state files by resynchronizing. Changing to and from the Remote Loader is a move. Therefore, if you move the JDBC driver using triggerless publication and want to avoid a full resync, also move all `jdbc_*.lg` and `jdbc_*.db` files in the state directory.

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

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

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

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

Table 6-9 *Default Directories*

Platform or Environment	Default Directory
Windows, for the Remote Loader	novell\remoteloader
Windows, for Identity Manager (local; not on the Remote Loader)	c:\novell\nds\dibfiles

The current directory might be different for a custom installation.

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

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

Properties

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

Table 6-10 *State Directory: Properties*

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

6.3.2 Database Scoping Parameters

- ♦ [“Synchronization Filter” on page 53](#)
- ♦ [“Schema Name” on page 55](#)
- ♦ [“Include Filter Expression” on page 55](#)
- ♦ [“Exclude Filter Expression” on page 56](#)
- ♦ [“Table/View Names” on page 56](#)

Synchronization Filter

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

Schema-Unaware Mode

When the Synchronization Filter parameter is present and set to empty (exclude all tables/views), the driver is schema-unaware. It does not retrieve table/view metadata on startup. Therefore, no metadata methods are required. See [Appendix F, “java.sql.DatabaseMetaData Methods,” on page 197](#).

When it is schema-unaware, the synchronization schema can be empty. Both the Schema Name and Sync Tables/Views parameters are completely ignored. Neither is required. Both can be absent, present, valued or valueless. See [“Schema Name” on page 55](#) and [“Table/View Names” on page 56](#).

In schema-unaware mode, the driver acts as a passthrough agent for embedded SQL. In this state, standard XDS events (for example, Add, Modify, and Delete) are ignored. See [Chapter 11, “Embedded SQL Statements in XDS Events,” on page 119](#). Also, triggered or triggerless publication no longer work.

Schema-Aware Mode

When the Synchronization Filter parameter is not present or set to a value other than empty (exclude all tables/views), the driver is schema-aware. It retrieves table/view metadata on a limited number of tables/views to facilitate data synchronization. You can cache metadata on all tables/views owned by a single database user (include by schema membership), or cache metadata on an explicit list of table/view names (include by table/view name). When schema-aware, the driver retrieves database table/view metadata on startup. For a list of required metadata methods, see [Appendix F, “java.sql.DatabaseMetaData Methods,” on page 197](#).

When schema-aware, parameter Schema Name or Table/View Names must be present and have a value. Because these two parameters are mutually exclusive, only one parameter can have a value. See [“Schema Name” on page 55](#) and [“Table/View Names” on page 56](#).

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

Table 6-11 *Schema-Dependent Parameters*

Parameter
Lock Statement Generator Class
Enable Referential Attribute Support?
Enable Meta-Identifier Support?
Left Outer Join Operator
Retrieve Minimal Metadata
Supports Schemas in Metadata Retrieval?

Parameter
Sort Column Names By
Disable Statement-Level Locking
Check Update Counts?
Add Default Values on Insert?
Generation/Retrieval Method (Table-Global)
Retrieval Timing (Table-Global)
Retrieval Timing
Disable Publisher?
Disable Statement-Level Locking?
Publication Mode
Enable Future Event Processing?
Event Log Table Name
Delete Processed Rows?
Allow Loopback?
Startup Option
Polling Interval (In Seconds)
Publication Time of Day
Post Polling Statements
Batch Size

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

Table 6-12 *Synchronization Filter: Properties*

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

Schema Name

The Schema Name parameter identifies the database schema being synchronized. A database schema is analogous to the name of the owner of the tables or views being synchronized. For example, to synchronize two tables, `usr` and `grp`, each belonging to database user `idm`, you enter `idm` as this parameter's value.

When using this parameter instead of Table/View Names, names of database objects are implicitly schema-qualified by the driver. As such, parameters referencing stored procedure, function, or table names do not need to be schema-qualified unless they reside in a schema other than the one specified here. In particular, Method and Timing (Table-Local) and Event Log Table Name are affected. See [“Table/View Names” on page 56](#), [“Method and Timing \(Table-Local\)” on page 73](#), and [“Event Log Table Name” on page 80](#).

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

Table 6-13 *Schema Name: Properties*

Property	Value
Tag Name	sync-schema
Required?	yes
Case-Sensitive?	See “Undelimited Identifier Case Sensitivity” on page 147 .
Sample Value	indirect
Default Value:	(none)

When the Schema Name parameter is used without the Synchronization Filter parameter, the Table/View Names parameter must be left empty or omitted from a configuration. See [“Synchronization Filter” on page 53](#) and [“Table/View Names” on page 56](#).

Changing the value of the Schema Name parameter forces a resync of all objects when triggerless publication is used.

Include Filter Expression

The Include Filter Expression parameter is only operative when the Schema Name parameter is used. See [“Schema Name” on page 55](#).

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

Table 6-14 *Include Filter Expression: Properties*

Property	Value
Tag Name	include-table-filter
Required?	no
Case-Sensitive?	yes
Sample Value	<code>idm_.*</code> (all table/view names starting with “ <code>idm_</code> ”)

Property	Value
Default Value	(none)
Legal Values	(any legal Java regular expression)

Exclude Filter Expression

This parameter is only operative when the Schema Name parameter is used. See [“Schema Name” on page 55](#).

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

Table 6-15 *Exclude Filter Expression: Properties*

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

Table/View Names

The Table/View Names parameter allows you to create a logical database schema by listing the names of the logical database classes to synchronize. Logical database class names are the names of parent tables and views. It is 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 or views of which only a few are of interest. Reducing the number of table/view definitions cached by the driver can shorten startup time as well as reduce runtime memory utilization.

When using this parameter instead of Schema Name, you probably need to schema-qualify other parameters that reference stored procedure, function, or table names. In particular, the Method and Timing (Table-Local) and Event Log Table Name parameters are affected. See [“Schema Name” on page 55](#), [“Method and Timing \(Table-Local\)” on page 73](#) and [“Event Log Table Name” on page 80](#).

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

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

Property	Value
Tag Name	sync-tables
Required?	yes

Property	Value
Case-Sensitive?	See “Undelimited Identifier Case Sensitivity” on page 147 .
Delimiters	semicolon, white space, comma
Sample Value	indirect.usr; indirect.grp
Default Value	(none)

When this parameter is used without the Synchronization Filter parameter, the Schema Name parameter must be left empty or omitted from a configuration. See [“Synchronization Filter” on page 53](#) and [“Schema Name” on page 55](#).

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

6.3.3 Connectivity Parameters

- ♦ [“Use Minimal Number of Connections?” on page 57](#)
- ♦ [“Connection Initialization Statements” on page 58](#)
- ♦ [“Connection Properties” on page 58](#)

Use Minimal Number of Connections?

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

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

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

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

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

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

Property	Value
Schema-Dependent	False

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

Setting this parameter to Boolean True reduces performance.

Connection Initialization Statements

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

The following table lists the properties of this parameter:

Table 6-18 *Connection Initialization Statements: Properties*

Property	Value
Tag Name	connection-init
Required?	no
Case-Sensitive?	See “Undelimited Identifier Case Sensitivity” on page 147 .
Delimiters	semicolon
Sample Value	USE idm; SET CHAINED OFF
Default Value	(none)
Schema-Dependent	False

Connection Properties

The Connection Properties parameter specifies authentication properties. This parameter is useful for specifying properties that cannot be set via the JDBC URL specified in the Authentication Context parameter. See [“Authentication Context” on page 46](#).

The primary purpose of this parameter is to enable encrypted transport for third-party JDBC drivers. For a list of relevant connection properties, see [“Sybase Adaptive Server Enterprise JConnect JDBC Driver” on page 168](#) and [“Oracle Thin Client JDBC Driver” on page 164](#).

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

When you use the Connection Properties parameter, authentication information can be passed via the JDBC URL specified in the Authentication Context parameter or here. See [“Authentication Context” on page 46](#).

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 46](#) and [“Application Password” on page 47](#). For username, the token is {\$username}. For password, the token is {\$password}.

The following table lists the properties of this parameter:

Table 6-19 *Connection Properties: Properties*

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

6.3.4 Compatibility Parameters

- ♦ [“JDBC Driver Descriptor Filename” on page 60](#)
- ♦ [“Database Descriptor Filename” on page 60](#)
- ♦ [“Use Manual Transactions?” on page 60](#)
- ♦ [“Transaction Isolation Level” on page 61](#)
- ♦ [“Reuse Statements?” on page 62](#)
- ♦ [“Number of Returned Result Sets” on page 63](#)
- ♦ [“Enable Statement-Level Locking?” on page 63](#)
- ♦ [“Lock Statement Generator Class” on page 64](#)
- ♦ [“Enable Referential Attribute Support?” on page 64](#)
- ♦ [“Enable Meta-Identifier Support?” on page 65](#)
- ♦ [“Force Username Case” on page 65](#)
- ♦ [“Left Outer Join Operator” on page 66](#)
- ♦ [“Retrieve Minimal Metadata” on page 66](#)
- ♦ [“Function Return Method” on page 67](#)
- ♦ [“Supports Schemas in Metadata Retrieval?” on page 67](#)
- ♦ [“Sort Column Names By” on page 68](#)

JDBC Driver Descriptor Filename

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

The following table lists the properties of this parameter:

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

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

Database Descriptor Filename

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

The following table lists the properties of this parameter:

Table 6-21 *Database Descriptor Filename: Properties*

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

Use Manual Transactions?

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

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

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

The following table lists the properties of this parameter:

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

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

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

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

Transaction Isolation Level

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

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

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

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

IMPORTANT: The list of supported isolation levels varies by database. For a list of supported transaction isolation levels for supported databases, see [“Supported Transaction Isolation Levels” on page 147](#).

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

The following table lists the properties of this parameter:

Table 6-23 *Transaction Isolation Level: Properties*

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

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

Reuse Statements?

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

Table 6-24 *Reuse Statements?: Properties*

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

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

Setting this parameter to Boolean False degrades performance.

Number of Returned Result Sets

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

This parameter is primarily used to avoid infinite loop conditions in “Oracle Thin Client JDBC Driver” on page 164 when evaluating the results of arbitrary SQL statements.

The following table lists the properties of this parameter:

Table 6-25 *Number of Returned Result Sets: Properties*

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

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

Enable Statement-Level Locking?

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

The following table lists the properties of this parameter:

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

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

Lock Statement Generator Class

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

The following table lists the properties of this parameter:

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

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

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

Enable Referential Attribute Support?

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

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

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

The following table lists the properties of this parameter:

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

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

Enable Meta-Identifier Support?

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

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

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

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

The following table lists the properties of this parameter:

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

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

Force Username Case

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

The primary purpose of this parameter is to enable interoperability with the Informix JDBC Driver when used against ANSI-compliant databases. See [“Informix JDBC Driver” on page 160](#).

The following table lists the properties of this parameter:

Table 6-30 *Force Username Case: Properties*

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

Property	Value
Schema-Dependent	False

Left Outer Join Operator

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

The following table lists the properties of this parameter:

Table 6-31 *Left Outer Join Operator: Properties*

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

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

Retrieve Minimal Metadata

When set to Boolean True, the driver calls only required metadata methods. When set to Boolean False, the driver calls required and optional metadata methods. For a list of required and optional metadata methods, refer to [Appendix F, “java.sql.DatabaseMetaData Methods,” on page 197](#). Optional metadata methods are required for multivalue and referential attribute synchronization.

Table 6-32 *Retrieve Minimal Metadata: Properties*

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

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

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

Function Return Method

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

The primary purpose of this parameter is to enable interoperability with the Informix JDBC driver. See [“Informix JDBC Driver” on page 160](#).

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

Table 6-33 *Function Return Method: Properties*

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

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

Supports Schemas in Metadata Retrieval?

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

The primary purpose of this parameter is to enable interoperability with the Informix JDBC Driver when used against ANSI-compliant databases. See [“Informix JDBC Driver” on page 160](#).

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

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

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

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

Sort Column Names By

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

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

Sorting columns names by hexadecimal value ensures that if a driver instance is relocated to a different server, it continues to function without modification. Sorting column names by platform or locale string collation order is more intuitive, but might require configuration changes if a driver instance is relocated to a different server. In particular, log table column order and compound column name order might change. In the case of the latter, Schema-Mapping policies and object association values might need to be updated. In the case of the former, log table columns might have to be renamed.

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

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

Table 6-35 Sort Column Names By: Properties

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

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

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

6.4 Subscription Parameters

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

Table 6-36 *Subscriber-Level Parameters and Properties*

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

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

Subscription parameters are in two subcategories:

- ♦ [Section 6.4.1, “Uncategorized Parameters,” on page 69](#)
- ♦ [Section 6.4.2, “Primary Key Parameters,” on page 71](#)

6.4.1 Uncategorized Parameters

- ♦ [“Disable Subscriber?” on page 69](#)
- ♦ [“Disable Statement-Level Locking?” on page 70](#)
- ♦ [“Check Update Counts?” on page 70](#)
- ♦ [“Add Default Values on Insert?” on page 71](#)

Disable Subscriber?

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

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

Table 6-37 *Disable Subscriber?: Properties*

Property	Value
Tag Name	disable

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

Disable Statement-Level Locking?

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

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

Table 6-38 *Disable Statement-Level Locking?: Properties*

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

Check Update Counts?

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

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

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

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

Table 6-39 *Check Update Counts?: Properties*

Property	Value
Tag Name	check-update-count
Required?	no
Default Value	1 (yes)

Property	Value
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Add Default Values on Insert?

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

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

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

Table 6-40 *Add Default Values on Insert?: Properties*

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

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

6.4.2 Primary Key Parameters

- ♦ [“Generation/Retrieval Method \(Table-Global\)” on page 72](#)
- ♦ [“Retrieval Timing \(Table-Global\)” on page 73](#)
- ♦ [“Method and Timing \(Table-Local\)” on page 73](#)

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

The following table identifies the supported methods and timings:

Table 6-41 *Supported Methods and Timings*

Method	Timing: before (row insertion)	Timing: after (row insertion)
None (subscription event)	X	0 ¹
Driver (Subscriber-generated)	X	X
Auto (auto-generated/identity column)	0 ²	X
(stored procedure/function)	X	X

¹ The Subscriber channel automatically overrides this timing and changes it to *before*. ² The Subscriber channel automatically overrides this timing and changes it to *after*.

Generation/Retrieval Method (Table-Global)

The Generation/Retrieval Method (Table-Global) parameter specifies how primary key values are generated or retrieved for all parent tables and views. The Method and Timing parameter overrides this parameter on a per-table/view basis. See [“Method and Timing \(Table-Local\)” on page 73](#).

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

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

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

When this parameter is set to *auto*, primary key values are retrieved via the `java.sql.Statement.getGeneratedKeys() : java.sql.ResultSet` method. The MySQL Connector/J JDBC driver is the only supported third-party JDBC driver that currently implements this method. See [“MySQL Connector/J JDBC Driver” on page 163](#).

Table 6-42 *Generation/Retrieval Method (Table-Global): Properties*

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

Retrieval Timing (Table-Global)

The Retrieval Timing (Table-Global) parameter specifies when the Subscriber channel retrieves primary key values for all parent tables and views. The Method and Timing (Table-Local) parameter overrides this parameter. See “[Method and Timing \(Table-Local\)](#)” on page 73.

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

Table 6-43 Retrieval Timing (Table-Global): Properties

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

Method and Timing (Table-Local)

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

When using the [Table/View Names](#) parameter, you probably need to explicitly schema-qualify any tables, views, stored procedures or functions referenced in this parameter’s value. When you use the [Schema Name](#) parameter, tables, views, stored procedures, or functions referenced in this parameter’s value are implicitly schema-qualified with that schema name. If tables, views, stored procedures, or functions referenced in this parameter’s value are located in a different schema than the implicit schema, they must be schema-qualified.

- ♦ “[BNF](#)” on page 73
- ♦ “[Generation or Retrieval Method](#)” on page 74
- ♦ “[Retrieval Timing](#)” on page 76

BNF

The BNF ([Backus Naur Form](http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html) (<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 be similar to the following:

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

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

Driver: This method assumes that the 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 against databases that do not support stored procedures or functions. When you use this method in a production environment, however, all SQL statements generated by an <add> event should be contained in a serializable transaction. For additional information, refer to [“Transaction Isolation Level” on page 61](#).

Instead of making all transactions serializable, you can also set individual transaction isolation levels by using embedded SQL attributes. For additional information, refer to [Section 11.6, “Transaction Isolation Level,” on page 126](#).

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

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

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

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

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

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

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

Some databases support identity columns that automatically generate primary key values for inserted rows. This method retrieves auto-generated primary key values through the JDBC 3 interface method `java.sql.Statement.getGeneratedKeys():java.sql.ResultSet`. The MySQL Connector/J JDBC driver is the only supported third-party JDBC driver that currently implements this method. See [“MySQL Connector/J JDBC Driver” on page 163](#).

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

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

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

Stored-Procedure/Function: This method assumes that the 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 be similar to the following:

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

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

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

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

Retrieval Timing

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

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

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

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

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

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

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

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

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

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

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

The following table lists the properties of this parameter:

Table 6-44 Retrieval Timing: Properties

Property	Value
Tag Name	key-gen
Required?	no
Case-Sensitive?	See “Unlimited Identifier Case Sensitivity” on page 147 .
Sample Value	usr("?=proc_idu()", before)
Default Value	(none)
Legal Values	(any string adhering to the BNF)
Schema-Dependent	True

6.5 Publication Parameters

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

Table 6-45 *Publisher-Level Parameters and Properties*

Display Name	Tag Name	Sample Value	Default Value	Required
Disable Publisher?	disable	1 (yes)	0 (no)	no
Disable Statement-Level Locking?	disable-locking	1 (yes)	0 (no)	no
Publication Mode	publication-mode	2 (triggerless)	1 (triggered)	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
Publication Time of Day	time-of-day	15:30:00	(none)	no ²
Post Polling Statements	post-poll-stmt	DELETE FROM direct.direct_process	(none)	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:

- ♦ [Section 6.5.1, “Uncategorized Parameters,” on page 77](#)
- ♦ [Section 6.5.2, “Triggered Publication Parameters,” on page 80](#)
- ♦ [Section 6.5.3, “Triggerless Publication Parameters,” on page 82](#)
- ♦ [Section 6.5.4, “Polling Parameters,” on page 83](#)

6.5.1 Uncategorized Parameters

- ♦ [“Disable Publisher?” on page 78](#)
- ♦ [“Disable Statement-Level Locking?” on page 78](#)
- ♦ [“Publication Mode” on page 78](#)
- ♦ [“Enable Future Event Processing?” on page 79](#)

Disable Publisher?

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

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

Table 6-46 *Disable Publisher?: Properties*

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

Disable Statement-Level Locking?

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

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

Table 6-47 *Disable Statement-Level Locking?: Properties*

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

Publication Mode

The Publication Mode parameter specifies which publication algorithm is used.

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

The following table lists the properties of this parameter:

Table 6-48 *Publication Mode: Properties*

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

Enable Future Event Processing?

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

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

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

Database local time is published as an attribute on each XDS event (for example, add, modify, delete). The attribute name is `jdbc:database-local-time`, where the `jdbc` namespace prefix is bound to `urn:dirxml:jdbc`. The format is the Java string representation of a `java.sql.Timestamp`: `yyyy-mm-dd hh:mm:ss.ffffffffff`. Depending upon the value of the Time Syntax parameter, the value indicating when an event should be processed can be published as an integer, as a canonical string, or as a Java string. See “Time Syntax” on page 49.

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

Table 6-49 *Mapping Time Syntax to Parse Methods*

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

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

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

An example policy is provided in [Appendix L, “Policy Example: Triggerless Future Event Processing,” on page 213](#).

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

The following table lists the properties of this parameter:

Table 6-50 *Enable Future Event Processing?: Properties*

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

6.5.2 Triggered Publication Parameters

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

- ♦ [“Event Log Table Name” on page 80](#)
- ♦ [“Delete Processed Rows?” on page 81](#)
- ♦ [“Allow Loopback?” on page 82](#)

Event Log Table Name

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

The table specified here must conform to the definition of [Chapter 10, “The Event Log Table,” on page 107](#).

When using [“Table/View Names” on page 56](#), you should explicitly schema-qualify this table name. When you use [“Schema Name” on page 55](#), this table name is implicitly schema-qualified with that schema name. If this table is located in a schema other than the implicit schema, it must be schema-qualified.

The following table lists the properties of this parameter:

Table 6-51 *Event Log Table Name: Properties*

Property	Value
Tag Name	log-table
Required?	no
Case-Sensitive?	See “Undelimited Identifier Case Sensitivity” on page 147 .
Sample Value	eventlog
Default Value	(none)
Schema-Dependent	True

This parameter is required if [“Publication Mode” on page 78](#) 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 84](#).
- ♦ Place a before-delete trigger on the event log table to intercept delete events executed against the event log table and to move deleted rows to a history table before they are deleted from the event log table.

The following table lists the properties of this parameter:

Table 6-52 *Delete Processed Rows?: Properties*

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

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

Allow Loopback?

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

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

The following table lists the properties of this parameter:

Table 6-53 *Allow Loopback?: Properties*

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

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

6.5.3 Triggerless Publication Parameters

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

Table 6-54 *Startup Option: Settings and Results*

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

The following table lists the properties of this parameter:

Table 6-55 *Startup Option: Properties*

Property	Value
Tag Name	startup-option
Required?	no
Default Value	1 (process all changes)

Property	Value
Legal Values	1 (resync all objects) 2 (process future changes only) 3 (process all changes)
Schema-Dependent	True

The following configuration changes can force a full resynchronization:

- ♦ Changing anything in the [Authentication Context](#) parameter other than URL properties forces a resynchronization of all objects when triggerless publication is used.
- ♦ Changing the value of the [Schema Name](#) parameter or the [Table/View Names](#) parameter forces a resynchronization of all objects when triggerless publication is used.
- ♦ Changing the [State Directory](#) parameter value.
- ♦ Moving or deleting state files. See [“Changes That Can Force Triggerless Publisher Resynchronization” on page 51](#).
- ♦ Changing the table/view structure in the database (in particular, changing the position or type of key columns).

6.5.4 Polling Parameters

- ♦ [“Polling Interval \(In Seconds\)” on page 83](#)
- ♦ [“Publication Time of Day” on page 84](#)
- ♦ [“Post Polling Statements” on page 84](#)
- ♦ [“Batch Size” on page 85](#)
- ♦ [“Heartbeat Interval \(In Minutes\)” on page 85](#)

Polling Interval (In Seconds)

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

The following table lists the properties of this parameter:

Table 6-56 *Polling Interval (In Seconds): Properties*

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

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

Publication Time of Day

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

The following table lists the properties of this parameter:

Table 6-57 *Publication Time of Day: Properties*

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

This parameter overrides the parameter Polling Interval (In Seconds). See [“Polling Interval \(In Seconds\)” on page 83](#).

Post Polling Statements

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

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

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

The following table lists the properties of this parameter:

Table 6-58 *Post Polling Statements: Properties*

Property	Value
Tag Name	post-poll-stmt
Required?	no
Case-Sensitive?	See “Unlimited Identifier Case Sensitivity” on page 147 .
Delimiters	semicolon
Sample Value	DELETE FROM direct.direct_process

Property	Value
Default Value	(none)
Legal Values	(any set of legal SQL statements)
Schema-Dependent	True

Batch Size

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

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

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

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

The following table lists the properties of this parameter:

Table 6-59 *Batch Size: Properties*

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

Heartbeat Interval (In Minutes)

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

The following table lists the properties of this parameter:

Table 6-60 *Heartbeat Interval (In Minutes): Properties*

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

6.6 Trace Levels

To see debugging output from the driver, add a DirXML-DriverTraceLevel attribute value from 1 to 7 on the driver set containing the driver instance. This attribute is commonly confused with the DirXML-XSL TraceLevel attribute. For more information on driver set trace levels, refer to [“Viewing Identity Manager Processes”](#) in the *Identity Manager 4.0 Common Driver Administration Guide*.

The driver supports the following seven trace levels:

Table 6-61 *Supported Trace Levels*

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

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

6.7 Configuring Third-Party JDBC Drivers

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

- ♦ Use the latest version of the driver.
- ♦ Third-party driver behavior might be configurable.
In many cases, incompatibility issues can be resolved by adjusting the driver’s JDBC URL properties.

- ♦ When you work with international characters, you often must explicitly specify to third-party drivers the character encoding that the database uses.

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

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

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

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

Table 6-62 *Recommended Settings for Third-Party JDBC Drivers*

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

6.8 Configuring jTDS Support for the JDBC Driver

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

Database Name	Class Name	Connection URL	Jar File Name
MS SQL version 2000/2005	<code>net.sourceforge.jtds.jdbc.Driver</code>	<code>jdbc:jtds:sqlserver://<server>:<port>1433/>;DatabaseName=<database></code>	<code>jtds-1.2.2.jar</code>
Sybase	<code>net.sourceforge.jtds.jdbc.Driver</code>	<code>jdbc:jtds:sybase://<server>:<port>5000/>;DatabaseName=<database></code>	<code>jtds-1.2.2.jar</code>

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

Place the jar file in the specific directory path for the platform being used. For information on placing the jar files, refer to [Section 13.3, “Third-Party Jar File Placement,” on page 156](#).

Managing the Driver

7

As you work with the JDBC driver, there are a variety of management tasks you might need to perform, including the following:

- ♦ Starting, stopping, and restarting the driver
- ♦ Viewing driver version information
- ♦ Using Named Passwords to securely store passwords associated with the driver
- ♦ Monitoring the driver's health status
- ♦ Backing up the driver
- ♦ Inspecting the driver's cache files
- ♦ Viewing the driver's statistics
- ♦ Using the DirXML Command Line utility to perform management tasks through scripts
- ♦ Securing the driver and its information
- ♦ Synchronizing objects
- ♦ Migrating and resynchronizing data
- ♦ Activating the driver

Because these tasks, as well as several others, are common to all Identity Manager drivers, they are included in one reference, the [*Identity Manager 4.0 Common Driver Administration Guide*](#).

- ♦ [Section 8.1, “High-Level View,” on page 91](#)
- ♦ [Section 8.2, “Logical Database Classes,” on page 91](#)
- ♦ [Section 8.3, “Indirect Synchronization,” on page 91](#)
- ♦ [Section 8.4, “Direct Synchronization,” on page 99](#)
- ♦ [Section 8.5, “Synchronizing Primary Key Columns,” on page 102](#)
- ♦ [Section 8.6, “Synchronizing Multiple Classes,” on page 102](#)
- ♦ [Section 8.7, “Mapping Multivalue Attributes to Single-Value Database Fields,” on page 103](#)

8.1 High-Level View

The following table shows a high-level view of how the driver maps Novell Identity Vault objects to database objects.

Table 8-1 *Mapping Identity Vault Objects to Database Objects*

Identity Vault Object	Database Object
Tree	Schema
Class	Table/View
Attribute	Column
Association	Primary Key

8.2 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.

8.3 Indirect Synchronization

In an indirect synchronization model, the driver maps the following:

Table 8-2 *Mappings in Indirect Synchronization*

Identity Vault Object	Database Object
Classes	Tables
Attributes	Columns
1 Class	1 parent table and 0 or more child tables
Single-value attribute	Parent table column
Multivalue attribute	Parent table column (holding delimited values) or Child table column (preferred)

- ♦ [Section 8.3.1, “Mapping eDirectory Classes to Logical Database Classes,” on page 92](#)
- ♦ [Section 8.3.2, “Parent Tables,” on page 94](#)
- ♦ [Section 8.3.3, “Parent Table Columns,” on page 94](#)
- ♦ [Section 8.3.4, “Child Tables,” on page 95](#)
- ♦ [Section 8.3.5, “Referential Attributes,” on page 96](#)
- ♦ [Section 8.3.6, “Single-Value Referential Attributes,” on page 96](#)
- ♦ [Section 8.3.7, “Multivalue Referential Attributes,” on page 97](#)

8.3.1 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),
    pwdeptime    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-map>

```

```

        </attr-name>
        <attr-name class-name="User">
            <nds-name>Login Disabled</nds-name>
            <app-name>disabled</app-name>
        </attr-name>
    </attr-name-map>
</rule>

```

8.3.2 Parent Tables

Parent tables are tables with an explicit primary key constraint that contains one or more columns. In a parent table, an explicit primary key constraint is required so that the driver knows which fields to include in an association value.

```

CREATE TABLE indirect.usr
(
    idu INTEGER NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu PRIMARY KEY (idu)
)

```

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

idu	fname	lname
1	John	Doe

The resulting association for this row is

```
idu=1,table=usr,schema=indirect
```

The case of database identifiers in association values is determined dynamically from database metadata at runtime.

8.3.3 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-value parent table column `pwdminlen`.

Parent table columns are implicitly prefixed with the schema name and name of the parent table. It is not necessary to explicitly table-prefix parent table columns. For example, `indirect.usr.fname` is equivalent to `fname` for schema mapping purposes.

```

<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>Given Name</nds-name>
            <app-name>fname</app-name>
        </attr-name>
    </attr-name-map>
</rule>

```

Large binary and string data types should usually be mapped to parent table columns. To map to a child table column, data types must be comparable in SQL statements. Large data types usually cannot be compared in SQL statements.

Large binary and string data types can be mapped to child table columns if the following occur:

- ♦ Each `<remove-value>` event on these types is transformed in a policy into a `<remove-all-values>element`
- ♦ An `<add-value>` element follows each `<remove-value>` event

8.3.4 Child Tables

A child table is a table that has a foreign key constraint on its parent table's primary key, linking the two tables together. The columns that comprise the child table's foreign key can have different names than the columns in the parent table's primary key.

The following example shows the relationship between parent table `usr` and child tables `usr_phone` and `usr_faxno`:

```
CREATE TABLE indirect.usr
(
    idu    INTEGER    NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu    PRIMARY KEY (idu)
)

CREATE TABLE indirect.usr_phone
(
    idu            INTEGER            NOT NULL,
    phoneno        VARCHAR2(64)    NOT NULL,
    CONSTRAINT fk_phone_idu FOREIGN KEY (idu)            REFERENCES
indirect.usr(idu)
)

CREATE TABLE indirect.usr_fax
(
    idu            INTEGER            NOT NULL,
    faxno          VARCHAR2(64)    NOT NULL,
    CONSTRAINT fk_fax_idu FOREIGN KEY (idu)            REFERENCES
indirect.usr(idu)
)
```

In a child table, constrain all columns NOT NULL.

The first constrained column in a child table identifies the parent table. In the above example, the constrained column in child table `usr_phone` is `idu`. The only purpose of this column is to relate tables `usr_phone` and `usr`. Because constrained columns do not contain any useful information, omit them from publication triggers and Schema Mapping policies.

The unconstrained column is the column of interest. It represents a single, multivalued attribute. In the above example, the unconstrained columns are `phoneno` and `faxno`. Because unconstrained columns can hold multiple values, they are ideal for mapping multivalued eDirectory attributes (for example, mapping the multivalued eDirectory attribute Telephone Number to `usrphone.phoneno`).

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

Table 8-3 *Sample Data*

idu	phoneno
1	111-1111
1	222-2222

Like parent table columns, child table columns are implicitly schema-prefixed. Unlike parent table columns, however, a child table column name must be explicitly prefixed with the child table name (for example, `usr_phone.phoneno`). Otherwise, the driver implicitly interprets column `phoneno` (the parent table column) as `usr.phoneno`, not the child table column `usr_phone.phoneno`.

```
<rule name="Schema Mapping Rule">
  <attr-name-map>
    <class-name>
      <nds-name>User</nds-name>
      <app-name>indirect.usr</app-name>
    </class-name>
    <attr-name class-name="User">
      <nds-name>Facsimile Telephone Number</nds-name>
      <app-name>usr_fax.faxno</app-name>
    </attr-name>
    <attr-name class-name="User">
      <nds-name>Telephone Number</nds-name>
      <app-name>usr_phone.phoneno</app-name>
    </attr-name>
  </attr-name-map>
</rule>
```

Map each multivalue eDirectory attribute to a different child table.

8.3.5 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.

8.3.6 Single-Value Referential Attributes

You can relate two parent tables through a single-value parent table column. This column must have a foreign key constraint pointing to the other parent table's primary key. The following example relates a single parent table `usr` to itself:

```
CREATE TABLE indirect.usr
(
  idu          INTEGER NOT NULL,
  -- ...
  manager     INTEGER,
  CONSTRAINT pk_usr_idu      PRIMARY KEY (idu),
  CONSTRAINT fk_usr_manager FOREIGN KEY (manager) REFERENCES
indirect.usr(idu)
)
```

NOTE: Single-valued referential columns should be nullable.

```
<rule name="Schema Mapping Rule">
  <attr-name-map>
    <class-name>
      <nds-name>User</nds-name>
      <app-name>indirect.usr</app-name>
    </class-name>
    <attr-name class-name="User">
      <nds-name>manager</nds-name>
      <app-name>manager</app-name>
    </attr-name>
  </attr-name-map>
</rule>
```

In the above example, each user can have only one manager who himself is a user.

8.3.7 Multivalue Referential Attributes

You can relate two parent tables through a common child table. This child table must have a column constrained by a foreign key pointing to the other parent table's primary key. The following example relates two parent tables `usr` and `grp` through a common child table `member`.

```
CREATE TABLE indirect.usr
(
  idu  INTEGER  NOT NULL,
  -- ...
  CONSTRAINT pk_usr_idu PRIMARY KEY (idu)
)

CREATE TABLE indirect.grp
(
  idg  INTEGER  NOT NULL,
  -- ...
  CONSTRAINT pk_grp_idg PRIMARY KEY (idg)
)

CREATE TABLE indirect.grp_member
(
  idg  INTEGER  NOT NULL,
  idu  INTEGER  NOT NULL,
  CONSTRAINT fk_member_idg FOREIGN KEY (idg) REFERENCES
indirect.grp(idg),  CONSTRAINT fk_member_idu FOREIGN KEY (idu)
REFERENCES indirect.usr(idu)
)
```

Constrain all columns in a child table NOT NULL.

```
<rule name="Schema Mapping Rule">
  <attr-name-map>
    <class-name>
      <nds-name>Group</nds-name>
      <app-name>indirect.grp</app-name>
    </class-name>
    <class-name>
      <nds-name>User</nds-name>
      <app-name>indirect.usr</app-name>
    </class-name>
  </attr-name-map>
</rule>
```

```

        </class-name>
        <attr-name class-name="Group">
            <nds-name>Member</nds-name>
            <app-name>grp_member.idu</app-name>
        </attr-name>
    </attr-name-map>
</rule>

```

The first constrained column in a child table determines which logical database class the child table `grp_member` belongs to. In the above example, `grp_member` is considered to be part of logical database class `grp`. `grp_member` is said to be a proper child of `grp`. The second constrained column in a child table is the multivalue referential attribute.

In the following example, the order of the constrained columns has been reversed so that `grp_member` is part of class `usr`. To more accurately reflect the relationship, table `grp_member` has been renamed to `usr_mbr_of`.

```

CREATE TABLE indirect.usr
(
    idu  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu PRIMARY KEY (idu)
)

CREATE TABLE indirect.grp
(
    idg  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_grp_idg PRIMARY KEY (idg)
)

CREATE TABLE indirect.usr_mbr_of
(
    idu  INTEGER  NOT NULL,
    idg  INTEGER  NOT NULL,
    CONSTRAINT fk_mbr_of_idu FOREIGN KEY (idu)          REFERENCES
indirect.usr(idu) ON DELETE CASCADE,
    CONSTRAINT fk_mbr_of_idg FOREIGN KEY (idg)          REFERENCES
indirect.grp(idg) ON DELETE CASCADE
)

<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
            <nds-name>Group</nds-name>
            <app-name>indirect.grp</app-name>
        </class-name>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>Group Membership</nds-name>
            <app-name>usr_mbr_of.idg</app-name>
        </attr-name>
    </attr-name-map>
</rule>

```

In databases that have no awareness of column position (such as DB2/AS400), order is determined by sorting column names by string or hexadecimal value. For additional information, see [“Sort Column Names By” on page 68](#).

In general, it is necessary to synchronize only bidirectional, multivalued, 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 user is usually more efficient than synchronizing all members of a group.

8.4 Direct Synchronization

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

Table 8-4 Mappings in Direct Synchronization

Identity Vault Object	Database Object
Classes	Views
Attributes	View Columns
Class	View
Single-value attribute	View Column
Multivalued attribute	View Column

The update capabilities of views vary between databases. Most databases allow views to be updated when they are comprised of a single base table. (That is, they do not join multiple tables.) If views are read-only, they cannot be used for subscription. Some databases allow update logic to be defined on views in instead-of-triggers, which allow a view to join multiple base tables and still be updateable.

For a list of databases that support instead-of-triggers, see [“Database Features” on page 145](#). Instead-of-trigger logic can be simulated, regardless of database capability using embedded SQL. See [Section 11.4, “Virtual Triggers,” on page 124](#).

- ♦ [Section 8.4.1, “View Column Meta-Identifiers,” on page 99](#)
- ♦ [Section 8.4.2, “Primary Key Columns,” on page 102](#)
- ♦ [Section 8.4.3, “Schema Mapping,” on page 102](#)

8.4.1 View Column Meta-Identifiers

A view is a logical table. Unlike tables, views do not physically exist in the database. As such, views usually cannot have traditional primary key/foreign key constraints. To simulate these constructs, the JDBC driver embeds constraints and other metadata in view column names. The difference between these constraints and traditional ones is that the former are not enforced at the database level. They are an application-level construct.

For example, to identify to the driver which fields to use when constructing association values, place a primary key constraint on a parent table. The corollary to this for a view is to prefix one or more column names with `pk_` (case-insensitive).

The following table lists the constraint prefixes that can be embedded in view column names.

Table 8-5 *Constraint Prefixes*

Constraint Prefixes (case-insensitive)	Interpretation
<code>pk_</code>	primary key
<code>fk_</code>	foreign key
<code>sv_</code>	single-value
<code>mv_</code>	multivalue

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 JDBC driver uses two normalized forms of view meta-identifiers:

- ♦ Database native form

Database native form is the column name as declared in the database. This form is usually much more verbose than schema mapping form, and contains all necessary meta information.

- ♦ Schema mapping form

Schema mapping form is returned when the driver returns the application schema. This form is much more concise than database native form because much of the meta information included in database native form is represented in XDS XML and not in the identifier.

The referential prefixes `pk_` and `fk_` are the only meta information preserved in schema mapping form. This limitation ensures backward compatibility.

The following table provides examples of each form:

Table 8-6 *Example Normalized Forms of View Meta-Identifiers*

Database Native Form	Schema Mapping Form
<code>pk_idu</code>	<code>pk_idu</code>
<code>sv_fname</code>	<code>fname</code>
<code>mv_phoneno</code>	<code>phoneno</code>
<code>fk_mv__idg__mbr_of</code>	<code>fk_mbr_of</code>

Equivalent Forms

A view column name without meta information is called its “effective” name, which is similar to a directory object’s “effective” rights. For the driver, view column name equivalency is determined without respect to meta information by default. For example, `pk_idu` is equivalent to `idu`, and `fk_mv_idg_mbr_of` is equivalent to `mbr_of`. Any variant form of a view meta column identifier can be passed to the driver at runtime. For backward compatibility reasons, meta information can be treated as part of the effective view column name. See [“Enable Meta-Identifier Support?” on page 65](#).

8.4.2 Primary Key Columns

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

8.4.3 Schema Mapping

Schema mapping conventions for views and view columns are equivalent to that used for parent tables and parent table columns.

8.5 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 multivalued attribute and can change. GUID has a single value and is static.

8.6 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)
);
```

8.7 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 has multiple values. Accordingly, multivalue and single-value attribute mappings are handled identically.

The driver implements the Most Recently Touched (MRT) algorithm with regard to single-value parent table or view columns. An MRT algorithm ensures that the most recently added attribute value or most recently deleted attribute value is stored in the database. The algorithm is adequate if the attribute in question has a single value.

If the attribute has multiple values, the algorithm has some undesirable consequences. When a value is deleted from a multivalue attribute, the database field it is mapped to is set to `NULL` and remains `NULL` until another value is added. The preferred solution to this undesirable behavior is to extend the 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.

Mapping XDS Events to SQL Statements

9

- ♦ [Section 9.1, “Mapping XDS Events for Indirect Synchronization,” on page 105](#)
- ♦ [Section 9.2, “Mapping XDS Events for Direct Synchronization,” on page 106](#)

9.1 Mapping XDS Events for Indirect Synchronization

The following table summarizes how the Subscriber channel maps XDS events to DML SQL statements for indirect synchronization:

Table 9-1 *Mapping XDS Events for Indirect Synchronization*

XML Event	SQL Equivalent
<add>	<ul style="list-style-type: none">♦ 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>	<ul style="list-style-type: none">♦ 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>	<ul style="list-style-type: none">♦ 1 parent table delete statement.♦ 1 delete statement for each child table.
<query>	<ul style="list-style-type: none">♦ 1 parent table select statement.♦ 1 select statement for each child table.
<move> <rename> <modify-password> <check-object-password>	<ul style="list-style-type: none">♦ 0 statements unless bound to embedded SQL statements.

9.2 Mapping XDS Events for Direct Synchronization

The following table summarizes how the Subscriber channel maps XDS events to DML SQL statements for direct synchronization:

Table 9-2 *Mapping XDS Events for Direct Synchronization*

XML Event	SQL Equivalent
<add>	<ul style="list-style-type: none">♦ 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 multivalued <add-attr> element.
<modify>	<ul style="list-style-type: none">♦ 1 view update statement for each single value <add-value> or <remove-value> element.♦ 1 view insert statement for each multivalued <add-value> element.♦ 1 view delete statement for each <remove-value> element.
<delete>	<ul style="list-style-type: none">♦ 1 view delete statement.
<query>	<ul style="list-style-type: none">♦ 1 view select statement.
<move> <rename> <modify-password> <check-object-password>	<ul style="list-style-type: none">♦ 0 statements unless bound to embedded SQL statements.

The Event Log Table

10

The event log table stores publication events. This section describes the structure and capabilities of the event log table.

You can customize the name of the event log table and its columns to avoid conflicts with reserved database keywords. The order, number, and data types of its columns, however, are fixed. In databases that are unaware of column position, order is determined by the Sort Column Names By parameter. See “Sort Column Names By” on page 68.

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 79.

- ♦ Section 10.1, “Event Log Columns,” on page 107
- ♦ Section 10.2, “Event Types,” on page 110

10.1 Event Log Columns

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

- ♦ Section 10.1.1, “`record_id`,” on page 107
- ♦ Section 10.1.2, “`table_key`,” on page 107
- ♦ Section 10.1.3, “`status`,” on page 108
- ♦ Section 10.1.4, “`event_type`,” on page 108
- ♦ Section 10.1.5, “`event_time`,” on page 109
- ♦ Section 10.1.6, “`perpetrator`,” on page 109
- ♦ Section 10.1.7, “`table_name`,” on page 109
- ♦ Section 10.1.8, “`column_name`,” on page 109
- ♦ Section 10.1.9, “`old_value`,” on page 109
- ♦ Section 10.1.10, “`new_value`,” on page 110

10.1.1 `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.

10.1.2 `table_key`

Format values for this column are exactly the same in all triggers for a logical database class. The BNF or Backus Naur Form (<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 referenced throughout this chapter, this column's value might be `pk_empno=1`.

For a hypothetical compound primary key (one containing multiple columns), this column's value might be `pkey1=value1+pkey2=value2`.

If primary key values placed in the `table_key` field contain any of the special characters `{, ; ' + " = \<>}`, where `{` and `}` contain the set of special characters, delimit the value with double quotes. You also need to escape the double quote character `"` as `\` and the literal escape character `\` as `\\` when they are contained inside a pair of double quotes.

For a hypothetical primary key containing special characters, this column's value might be `pkey=" ; ' + \" = \\ < > "`. (Note the double quotes and escaped characters.)

Differences in padding or formatting might result in out-of-order event processing. For performance reasons, remove any unnecessary white space from numeric values. For example, `idu=1` is preferred over `idu= 1`.

10.1.3 status

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

Table 10-1 Permitted Values for Status Columns

Character Value	Interpretation
N	new
S	success
W	warning
E	error
F	fatal

To be processed, all rows inserted into the event log table must have a `status` value of N. The remainder of the status characters are used solely by the Publisher channel to designate processed rows. All other characters are reserved for future use.

Status values are case sensitive.

10.1.4 event_type

Values in this column must be between 1 and 8. All other numbers are reserved for future use.

The following table describes each event type:

Table 10-2 *Event Types*

Event Type	Interpretation
1	insert field
2	update field
3	update field (remove all values)
4	delete row
5	insert row (query-back)
6	update row (query-back)
7	insert field (query-back)
8	update field (query-back)

For additional information on this field, see [Section 10.2, “Event Types,” on page 110](#).

10.1.5 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 79](#).

10.1.6 perpetrator

This column identifies the database user who instigated the event. A `NULL` value is interpreted as a user other than the driver user. Rows with a `NULL` value or value not equal to the driver’s database username are published. Rows with a value equal to the driver’s database username are not published unless the Allow Loopback Publisher parameter is set to Boolean True. See [“Allow Loopback?” on page 82](#).

10.1.7 table_name

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

10.1.8 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 cannot start.

10.1.9 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 cannot start.

10.1.10 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 cannot start.

10.2 Event Types

The following table describes each event type:

Table 10-3 *Event Types*

Event Type	Interpretation
1	insert field
2	update field
3	update field (remove all values)
4	delete row
5	insert row (query-back)
6	update row (query-back)
7	insert field (query-back)
8	update field (query-back)

Event types are in four major categories. Some categories overlap. The following table describes each category and indicates which event types are members:

Table 10-4 *Event Categories and Types*

Event Category	Event Types
Per-field (attribute)	1, 2, 3, 7, 8
Per-row (object)	4, 5, 6
Non-query-back	1, 2, 3, 4
Query-back	5, 6, 7, 8
Per-field, non-query-back	1, 2, 3
Per-field, query-back	7, 8
Per-row, non-query-back	4
Per-row, query-back	5, 6

In general, a combination of event types from each category yields the best trade-off in terms of space, time, implementation complexity, and performance.

Per-field event types are more granular, require more space, and are more complex to implement than per-row event types. Per-row events are less granular, require less space, and are easier to implement than per-field event types.

Query-back event types use less space but require more time to process than non-query-back event types. Non-query-back event types use more space but require less time to process than query-back event types.

Query-back event types take precedence over their non-query-back counterparts. Non-query-back events are ignored if a query-back event is logged for the same field or object. For example, if an event of type 2 (update-field, non-query-back) and 8 (update-field, query-back) are logged on the same field, the type 2 event is ignored in favor of the type 8 event.

Furthermore, query-back row event types take precedence over query-back field event types. For example, if an event type 8 (update field, query-back) and a event type 6 (update row query-back) are logged on the same object, the type 8 event is ignored in favor of the type 6 event.

Query-back events are ignored by the Publisher if the database object no longer exists. They are dependent upon the database object still being available at processing time. Therefore, logged query-back adds and modifies (event types 5, 6, 7, 8) have no effect once the database object they refer to is deleted.

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

Table 10-5 *Basic Correlation of Publication Event Types*

Event Type	Resulting XDS
insert	<add>
update	<modify>
delete	<delete>

The following example illustrates XML that the Publisher channel generates for events logged on the `usr` table for each possible event type.

```
CREATE TABLE indirect.usr
(
  idu    INTEGER NOT NULL,
  fname  VARCHAR2(64),
  photo  LONGRAW,
  --...
  CONSTRAINT pk_usr_idu PRIMARY KEY(idu)
);
```

The following table shows the initial contents of `usr` after a new row has been inserted:

Table 10-6 *An Inserted Row in the `usr` Table*

idu	fname	lname	photo
1	Jack	Frost	0xAAAA

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

Table 10-7 *An Updated Row in the `usr` Table*

idu	fname	lname	photo
1	John	Doe	0xB BBB

Insert Field

The table below shows the contents of the event log table after a new row is inserted into table `usr`. The value for column `photo` has been Base64-encoded. The Base64-encoded equivalent of `0xAAAA` is `qqo=`.

Table 10-8 *Event Log Table: Insert Field*

event_type	table	table_key	column_name	old_value	new_value
1	usr	idu=1	fname	NULL	Jack
1	usr	idu=1	lname	NULL	Frost
1	usr	idu=1	photo	NULL	qqo=

The Publisher channel generates the following XML:

```
<add class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <add-attr attr-name="fname">
    <value type="string">Jack</value>
  </add-attr>
  <add-attr attr-name="lname">
    <value type="string">Frost</value>
  </add-attr>
  <add-attr attr-name="photo">
    <value type="octet">qqo=</value>
  </add-attr>
</add>
```

Update Field

The following table shows the contents of the event log table after the row in table `usr` has been updated. The values for column `photo` has been Base64-encoded. The Base64-encoded equivalent of `0xB BBB` is `u7s=`.

Table 10-9 *Event Log Table: Update Field*

event_type	table	table_key	column_name	old_value	new_value
2	usr	idu=1	fname	Jack	John
2	usr	idu=1	lname	Frost	Doe

event_type	table	table_key	column_name	old_value	new_value
2	usr	idu=1	photo	qqo=	u7s=

The Publisher channel generates the following XML:

```
<modify class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <modify-attr attr-name="fname">
    <remove-value>
      <value type="string">Jack</value>
    </remove-value>
    <add-value>
      <value type="string">John</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="lname">
    <remove-value>
      <value type="string">Frost</value>
    </remove-value>
    <add-value>
      <value type="string">Doe</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="photo">
    <remove-value>
      <value type="octet">qqo=</value>
    </remove-value>
    <add-value>
      <value type="octet">u7s=</value>
    </add-value>
  </modify-attr>
</modify>
```

Update Field (Remove-All-Values)

The following table shows the contents of the event log table after the row in table `usr` has been updated. The value for column `photo` has been Base64-encoded.

Table 10-10 Event Log Table: Update Field (Remove-All-Values)

event_type	table	table_key	column_name	old_value	new_value
3	usr	idu=1	fname	Jack	John
3	usr	idu=1	lname	Frost	Doe
3	usr	idu=1	photo	qqo=	u7s=

The Publisher channel generates the following XML:

```

<modify class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <modify-attr attr-name="fname">
    <remove-all-values/>
    <add-value>
      <value type="string">John</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="lname">
    <remove-all-values/>
    <add-value>
      <value type="string">Doe</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="photo">
    <remove-all-values/>
    <add-value>
      <value type="octet">u7s=</value>
    </add-value>
  </modify-attr>
</modify>

```

Delete Row

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

Table 10-11 Event Log Table: Delete Row

event_type	table	table_key	column_name	old_value	new_value
4	usr	idu=1	NULL	NULL	NULL

The Publisher channel generates the following XML:

```

<delete class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
</delete>

```

Insert Row (Query-Back)

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

Table 10-12 Event Log Table: Insert Row (Query-Back)

event_type	table	table_key	column_name	old_value	new_value
5	usr	idu=1	NULL	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```

<add class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <add-attr attr-name="fname">
    <value type="string">John</value>
  </add-attr>
  <add-attr attr-name="lname">
    <value type="string">Doe</value>
  </add-attr>
  <add-attr attr-name="photo">
    <value type="octet">u7s=</value>
  </add-attr>
</add>

```

Update Row (Query-Back)

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

Table 10-13 Event Log Table: Update Row (Query-Back)

event_type	table	table_key	column_name	old_value	new_value
6	usr	idu=1	NULL	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```

<modify class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <modify-attr attr-name="fname">
    <remove-all-values/>
    <add-value>
      <value type="string">John</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="lname">
    <remove-all-values/>
    <add-value>
      <value type="string">Doe</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="photo">
    <remove-all-values/>
    <add-value>
      <value type="octet">u7s=</value>
    </add-value>
  </modify-attr>
</modify>

```

Insert Field (Query-Back)

The following table shows the contents of the event log table after a new row is inserted into table `usr`. Old and new values are omitted because they are not used.

Table 10-14 *Event Log Table: Insert Field (Query-Back)*

event_type	table	table_key	column_name	old_value	new_value
7	usr	idu=1	fname	NULL	NULL
7	usr	idu=1	lname	NULL	NULL
7	usr	idu=1	photo	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```
<add class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <add-attr attr-name="fname">
    <value type="string">John</value>
  </add-attr>
  <add-attr attr-name="lname">
    <value type="string">Doe</value>
  </add-attr>
  <add-attr attr-name="photo">
    <value type="octet">u7s</value>
  </add-attr>
</add>
```

Update Field (Query-Back)

The following table shows the contents of the event log table after the row in table `usr` has been updated. Old and new values are omitted because they are not used.

Table 10-15 *Event Log Table: Update Field (Query-Back)*

event_type	table	table_key	column_name	old_value	new_value
8	usr	idu=1	fname	NULL	NULL
8	usr	idu=1	lname	NULL	NULL
8	usr	idu=1	photo	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```
<modify class-name="usr">
  <association>idu=1,table=usr,schema=indirect
</association>
  <modify-attr attr-name="fname">
    <remove-all-values/>
    <add-value>
      <value type="string">John</value>
    </add-value>
  </modify-attr>
  <modify-attr attr-name="lname">
    <remove-all-values/>
```

```
        <add-value>
            <value type="string">Doe</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="photo">
        <remove-all-values/>
        <add-value>
            <value type="octet">u7s=</value>
        </add-value>
    </modify-attr>
</modify>
```


Embedded SQL Statements in XDS Events

11

Embedded SQL allows you to embed SQL statements in XDS-formatted XML documents. You can use embedded SQL statements along with XDS events or use them alone. When embedded SQL statements are used alone, embedded SQL processing does not require that the driver know anything about tables/view in the target database. Therefore, the driver can run in schema-unaware mode. See [“Synchronization Filter” on page 53](#). When using embedded SQL alone, you must establish associations manually. The driver won’t establish them for you.

When used in conjunction with XDS events, embedded SQL can act as a virtual database trigger. In the same way that you can install database triggers on a table and cause side effects in a database when certain SQL statements are executed, embedded SQL can cause side effects in a database in response to certain XDS events.

All examples in this section reference the following `indirect_usr` table.

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

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

- ♦ [Section 11.1, “Common Uses of Embedded SQL,” on page 120](#)
- ♦ [Section 11.2, “Embedded SQL Basics,” on page 120](#)
- ♦ [Section 11.3, “Token Substitution,” on page 121](#)
- ♦ [Section 11.4, “Virtual Triggers,” on page 124](#)
- ♦ [Section 11.5, “Manual vs. Automatic Transactions,” on page 125](#)
- ♦ [Section 11.6, “Transaction Isolation Level,” on page 126](#)
- ♦ [Section 11.7, “Statement Type,” on page 126](#)
- ♦ [Section 11.8, “SQL Queries,” on page 128](#)
- ♦ [Section 11.9, “Data Definition Language \(DDL\) Statements,” on page 129](#)
- ♦ [Section 11.10, “Logical Operations,” on page 130](#)
- ♦ [Section 11.11, “Implementing Password Set with Embedded SQL,” on page 130](#)
- ♦ [Section 11.12, “Implementing Modify Password with Embedded SQL,” on page 131](#)
- ♦ [Section 11.13, “Implementing Check Object Password,” on page 132](#)
- ♦ [Section 11.14, “Calling Stored Procedures and Functions,” on page 132](#)
- ♦ [Section 11.15, “Best Practices,” on page 141](#)

11.1 Common Uses of Embedded SQL

You can accomplish the following by embedding SQL in XDS events:

- ♦ Create database users or roles.
- ♦ Set, check, or modify user passwords.
- ♦ Manage database user or role privileges.

For examples of each, consult the `User DDL Command Transformation` style sheet on the Subscriber channel in the example driver configuration.

11.2 Embedded SQL Basics

- ♦ [Section 11.2.1, “Elements,” on page 120](#)
- ♦ [Section 11.2.2, “Namespaces,” on page 120](#)
- ♦ [Section 11.2.3, “Embedded SQL Example,” on page 120](#)

11.2.1 Elements

SQL is embedded in XDS events through the `<jdbc:statement>` and `<jdbc:sql>` elements. The `<jdbc:statement>` element can contain one or more `<jdbc:sql>` elements.

11.2.2 Namespaces

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

You must use namespace-prefixed embedded SQL elements and attributes. Otherwise, the driver does not recognize them. In all examples in this section, the prefix used is `jdbc`. In practice, the prefix can be whatever you want it to be, as long as it is bound to the namespace value `urn:dirxml:jdbc`.

The following XML example illustrates how to use and properly namespace-prefix embedded SQL elements. In the following example, the namespace declaration and namespace prefixes are bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr">
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' </jdbc:sql>
  </jdbc:statement>
</input>
```

11.2.3 Embedded SQL Example

The following XML example illustrates how to use the `<jdbc:statement>` and `<jdbc:sql>` elements and their interpretation. In the following example, embedded SQL elements are bolded:


```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr">
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John'           </jdbc:sql>
  </jdbc:statement>
</input>

```

Because the Subscriber channel resolves `<add>` events to one or more INSERT statements, the XML shown above resolves to:

```

SET AUTOCOMMIT OFF
INSERT INTO indirect.usr(lname) VALUES('Doe');
COMMIT; --explicit commit
UPDATE indirect.usr SET fname = 'John';
COMMIT; --explicit commit

```

11.3 Token Substitution

Rather than require you to parse field values from an association, the Subscriber channel supports token substitution in embedded SQL statements. In the following examples, tokens and the values they reference are **bolded**:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect</association>
    <modify-attr name="lname">
      <add-value>
        <value>DoeRaeMe</value>
      </add-value>
    </modify-attr>
  </modify>
  <jdbc:statement>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
      idu = { $idu }</jdbc:sql>
  </jdbc:statement>
</input>

```

Token placeholders must adhere to the XSLT attribute value template syntax `{ $field-name }`. Also, the referenced association element must precede the `<jdbc:statement>` element in the XDS document, or must be present as a child of the `<jdbc:statement>` element. Alternatively, instead of copying the association element as child of the `<jdbc:statement>` element, you could copy the `src-entry-id` of the element containing the association element onto the `<jdbc:statement>` element. Both approaches are **bolded** in the following examples:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect</association>
    <modify-attr name="lname">
      <add-value>
        <value>DoeRaeMe</value>
      </add-value>
    </modify-attr>
  </modify>
  <jdbc:statement>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
      idu = { $idu }</jdbc:sql>
  </jdbc:statement>
</input>

```

```

        </modify-attr>
    </modify>
    <jdbc:statement>
        <association>idu=1,table=usr,schema=indirect</association>
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
            idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>

<input xmlns:jdbc="urn:dirxml:jdbc">
    <modify class-name="usr" src-entry-id="0">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">
            <add-value>
                <value>DoeRaeMe</value>
            </add-value>
        </modify-attr>
    </modify>
    <jdbc:statement src-entry-id="0">
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
            idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>

```

The `{$field-name}` token must refer to one of the naming RDN attribute names in the association value. The above examples have only one naming attribute: `idu`.

An `<add>` event is the only event where an association element is not required to precede embedded SQL statements with tokens because the association has not been created yet. Additionally, any embedded SQL statements using tokens must follow, not precede, the `<add>` event. For example:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
            idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>

```

To prevent tracing of sensitive information, you can use the `$$$password` token to refer to the contents of the immediately preceding `<password>` element within the same document. In the following example, the password token and the value it refers to are bolded:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <password>some password</password>
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
    </add>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET fname = $$$password WHERE
            idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>

```

```

        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>CREATE USER jdoe IDENTIFIED BY
            {{password}}</jdbc:sql>
    </jdbc:statement>
</input>

```

Furthermore, you can also refer to the driver's database authentication password specified by the Application Password parameter as `{{driver-password}}`. See [“Application Password” on page 47](#). Named password substitution is not yet supported.

Just as with association elements, the referenced password element must precede the `<jdbc:statement>` element in the XDS document or must be present as a child of the `<jdbc:statement>` element. Alternatively, instead of copying the password element as child of the `<jdbc:statement>` element, you could copy the `src-entry-id` of the element containing the password element onto the `<jdbc:statement>` element. Both approaches are bolded in the following examples:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <password>some password</password>
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <password>some password</password>
        <jdbc:sql>CREATE USER jdoe IDENTIFIED BY
            {{password}}</jdbc:sql>
    </jdbc:statement>
</input>

```

```

<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr" src-entry-id="0">
        <password>some password</password>
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement src-entry-id="0">
        <jdbc:sql>CREATE USER jdoe IDENTIFIED BY
            {{password}}</jdbc:sql>
    </jdbc:statement>
</input>

```

11.4 Virtual Triggers

In the same way that database triggers can fire before or after a triggering statement, embedded SQL can be positioned before or after the triggering XDS event. The following examples show how you can embed SQL before or after an XDS event.

Virtual Before Trigger

```
<input xmlns:jdbc"urn:dirxml:jdbc">
  <jdbc:statement>
    <association>idu=1,table=usr,schema=indirect</association>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
idu = {$idu}</jdbc:SQL>
  </jdbc:statement>
  <modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect</association>
    <modify-attr name="lname">
      <remove-all-values/>
      <add-value>
        <value>Doe</value>
      </add-value>
    </modify-attr>
  </modify>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
COMMIT; --explicit commit
UPDATE indirect.usr SET lname = 'Doe' WHERE idu = 1;
COMMIT; --explicit commit
```

Virtual After Trigger

```
<input xmlns:jdbc"urn:dirxml:jdbc">
  <modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect</association>
    <modify-attr name="lname">
      <remove-all-values/>
      <add-value>
        <value>Doe</value>
      </add-value>
    </modify-attr>
  </modify>
  <jdbc:statement>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE idu =
{$idu}</jdbc:sql>
  </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
UPDATE indirect.usr SET lname = 'Doe' WHERE idu = 1;
COMMIT; --explicit commit
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
COMMIT; --explicit commit
```

11.5 Manual vs. Automatic Transactions

You can manually group embedded SQL and XDS events by using two custom attributes:

- ♦ `jdbc:transaction-type`
- ♦ `jdbc:transaction-id`

`jdbc:transaction-type`

This attribute has two values: `manual` and `auto`. By default, most XDS events of interest (`<add>`, `<modify>`, and `<delete>`) are implicitly set to the manual transaction type. The manual setting enables XDS events to resolve to a transaction consisting of one or more SQL statement.

By default, embedded SQL events are set to auto transaction type because some SQL statements, such as DDL statements, cannot usually be included in a manual transaction. In the following example, the attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr" jdbc:transaction-type="auto">
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement>
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
      idu = {$idu}</jdbc:sql>
  </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT ON
INSERT INTO indirect.usr(lname) VALUES('Doe');
-- implicit commit
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
-- implicit commit
```

`jdbc:transaction-id`

The Subscriber channel ignores this attribute unless the element's `jdbc:transaction-type` attribute value defaults to or is explicitly set to `manual`. The following XML shows an example of a manual transaction. The attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr" jdbc:transaction-id="0">
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement jdbc:transaction-type="manual"
jdbc:transaction-id="0">
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
      idu = {$idu}</jdbc:sql>
  </jdbc:statement>
</input>
```

This XML resolves to:

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

11.6 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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html>).

- ♦ none
- ♦ read uncommitted
- ♦ read committed
- ♦ repeatable read
- ♦ serializable

The driver's default transaction isolation level is `read committed` unless overridden by a descriptor file. In manual transactions, place the `jdbc:isolation-level` attribute on the first element in the transaction. This attribute is ignored on subsequent elements. In the following example, the attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr" jdbc:transaction-id="0"
        jdbc:isolation-level="serializable">
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement jdbc:transaction-type="manual"
        jdbc:transaction-id="0">
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
      WHERE idu = {${idu}}</jdbc:sql>
  </jdbc:statement>
</input>
```

This XML resolves to:

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

11.7 Statement Type

The Subscriber channel executes embedded SQL statements, but it doesn't understand them. The JDBC 1 interface defines several methods for executing different types of SQL statements. The following table contains these methods:

Table 11-1 *Methods for Executing SQL Statements*

Statement Type	Method Executed
SELECT	<code>java.sql.Statement.executeQuery(String query):java.sql.ResultSet</code>
INSERT	<code>java.sql.Statement.executeUpdate(String update):int</code>
UPDATE	<code>java.sql.Statement.executeUpdate(String update):int</code>
DELETE	<code>java.sql.Statement.executeUpdate(String update):int</code>
CALL or EXECUTE SELECT INSERT UPDATE DELETE	<code>java.sql.Statement.execute(String sql):boolean</code>

The simplest solution is to map all SQL statements to the `java.sql.Statement.execute(String sql):boolean` method. By default, the Subscriber channel uses this method.

Some third-party drivers, particularly Oracle's JDBC drivers, incorrectly implement the methods used to determine the number of result sets that this method generates. Consequently, the driver can get caught in an infinite loop leading to high CPU utilization. To circumvent this problem, you can use the `jdbc:type` attribute on any `<jdbc:statement>` element to map the SQL statements contained in it to the following methods instead of the default method:

- ♦ `java.sql.Statement.executeQuery(String query):java.sql.ResultSet`
- ♦ `java.sql.Statement.executeUpdate(String update):int`

The `jdbc:type` attribute has two values: `update` and `query`. For `INSERT`, `UPDATE`, or `DELETE` statements, set the value to `update`. For `SELECT` statements, set the value to `query`. In the absence of this attribute, the driver maps all SQL statements to the default method. If placed on any element other than `<jdbc:statement>`, this attribute is ignored.

Recommendations:

- ♦ Place the `jdbc:type="query"` attribute value on all `SELECT` statements.
- ♦ Place the `jdbc:type="update"` attribute value on all `INSERT`, `UPDATE`, and `DELETE` statements.
- ♦ Place no attribute value on stored procedure/function calls.

The following XML shows an example of the `jdbc:type` attribute. The attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr">
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement jdbc:type="update">
    <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
      WHERE idu = {$idu}</jdbc:sql>
  </jdbc:statement>
</input>
```

11.8 SQL Queries

To fully support the query capabilities of a database and avoid the difficulty of translating native SQL queries into an XDS format, the driver supports native SQL query processing. You can embed select statements in XDS documents in exactly the same way as any other SQL statement.

For example, assume that the table `usr` has the following contents:

Table 11-2 *Example Contents*

idu	fname	lname
1	John	Doe

The XML document below results in an output document containing a single result set.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <jdbc:statement jdbc:type="query">
    <jdbc:sql>SELECT * FROM indirect.usr</jdbc:sql>
  </jdbc:statement>
</input>

<output xmlns:jdbc="urn:dirxml:jdbc">
  <jdbc:result-set jdbc:number-of-rows="1">
    <jdbc:row jdbc:number="1">
      <jdbc:column jdbc:name="idu"
        jdbc:position="1"
        jdbc:type="java.sql.Types.BIGINT"
        <jdbc:value>1</jdbc:value>
      </jdbc:column>
      <jdbc:column jdbc:name="fname"
        jdbc:position="2"
        jdbc:type="java.sql.Types.VARCHAR"
        <jdbc:value>John</jdbc:value>
      </jdbc:column>
      <jdbc:column jdbc:name="lname"
        jdbc:position="3"
        jdbc:type="java.sql.Types.VARCHAR"
        <jdbc:value>Doe</jdbc:value>
      </jdbc:column>
    </jdbc:row>
  </jdbc:result-set>
  <status level="success"/>
</output>
```

SQL queries always produce a single `<jdbc:result-set>` element whether or not the result set contains any rows. If the result set is empty, the `jdbc:number-of-rows` attribute is set to zero.

You can embed more than one query in a document. SQL queries don't require that the referenced tables/views in the synchronization schema be visible to the driver. However, XDS queries do.

If you are building an event to be sent via the Command Processor instead of part of the regular event flow, you need to build the XML in a nodeset variable and use the Parse XML token before issuing the command. For more information, see ["XML Parse"](#) in the *Policies in Designer 4.0*.

Example logic XML:


```

<policy xmlns:cmd="http://www.novell.com/nxsl/java/
com.novell.nds.dirxml.driver.XdsCommandProcessor">
  <rule>
    <description>generate SQL Select Statement</description>
    <conditions>
      <and>
        <if-class-name op="equal">User</if-class-name>
      </and>
    </conditions>
    <actions>
      <do-set-local-variable name="sqlstatement">
        <arg-node-set>
          <token-xml-parse>
            <token-text xml:space="preserve">&lt;input
xmlns:jdbc="urn:dirxml:jdbc">&lt;jdbc:statement
jdbc:type="query">&lt;jdbc:sql></token-text>
            <token-text xml:space="preserve">SELECT * FROM lab.users;</token-
text>
            <token-text xml:space="preserve">&lt;/jdbc:sql>&lt;/
jdbc:statement>&lt;/input></token-text>
          </token-xml-parse>
        </arg-node-set>
      </do-set-local-variable>
      <do-trace-message color="yellow" level="1">
        <arg-string>
          <token-xpath expression="cmd:execute($destCommandProcessor,
$sqlstatement)"/>
        </arg-string>
      </do-trace-message>
      <do-veto/>
    </actions>
  </rule>
</policy>

```

11.9 Data Definition Language (DDL) Statements

Generally, it is not possible to run a Data Definition Language (DDL) statement in a database trigger because most databases do not allow mixed DML and DDL transactions. Although virtual triggers do not overcome this transactional limitation, they do allow DDL statements to be executed as a side effect of an XDS event.

For example:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr">
    <add-attr name="fname">
      <value>John</value>
    </add-attr>
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
  </add>
  <jdbc:statement>
    <jdbc:sql>CREATE USER jdoe IDENTIFIED BY novell</jdbc:sql>
  </jdbc:statement>
</input>

```

This XML resolves to:

```

SET AUTOCOMMIT OFF
INSERT INTO indirect.usr(fname, lname) VALUES('John', 'Doe');
COMMIT; -- explicit commit
SET AUTOCOMMIT ON
CREATE USER jdoe IDENTIFIED BY novell;
-- implicit commit

```

Using the `jdbc:transaction-id` and `jdbc:transaction-type` attributes to group DML and DDL statements into a single transaction causes the transaction to be rolled back on most databases. Because DDL statements are generally executed as separate transactions, it is possible that the `insert` statement in the above example might succeed and the `create user` statement might roll back.

It is not possible, however, that the `insert` statement fails and the `create user` statement succeeds. The driver stops executing chained transactions at the point where the first transaction is rolled back.

11.10 Logical Operations

Because it is not generally possible to mix DML and DDL statements in a single transaction, a single event can consist of one or more transactions. You can use the `jdbc:op-id` and `jdbc:op-type` to group multiple transactions together into a single logical operation. When grouped in this way, all members of the operation are handled as a single unit with regard to status. If one member has an error, all members return the same status level. Similarly, all members share the same status type.

```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr" jdbc:op-id="0"
        jdbc:op-type="password-set-operation">
    <add-attr name="fname">
      <value>John</value>
    </add-attr>
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
    <password>Doe{$idu}</password>
  </add>
  <jdbc:statement jdbc:op-id="0">
    <jdbc:sql>CREATE USER jdoe IDENTIFIED BY {$$password}
  </jdbc:sql>
  </jdbc:statement>
</input>

```

The `jdbc:op-type` attribute is ignored on all elements except the first element in a logical operation.

11.11 Implementing Password Set with Embedded SQL

Initially setting a password is usually accomplished by creating a database user account. Assuming that an `<add>` event is generated on the Subscriber channel, the following is an example of the output generated by XSLT style sheets that implement password set as a side effect of an XDS `<add>` event:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <add class-name="usr" jdbc:op-id="0"
      jdbc:op-type="password-set-operation">
    <add-attr name="fname">
      <value>John</value>
    </add-attr>
    <add-attr name="lname">
      <value>Doe</value>
    </add-attr>
    <password>Doe{$idu}</password>
  </add>
  <jdbc:statement jdbc:op-id="0">
    <jdbc:sql>CREATE USER jdoe IDENTIFIED BY {$$password}
  </jdbc:sql>
  </jdbc:statement>
</input>

```

The `<add>` event is logically bound to the `CREATE USER` DDL statement by the `jdbc:op-id` and `jdbc:op-type` attributes.

The User DDL Command Transformation style sheet in the `example.xml` configuration file contains sample XSLT templates that bind user account creation DDL statements to `<add>` events for all databases that support them.

11.12 Implementing Modify Password with Embedded SQL

Modifying a password is usually accomplished by altering an existing database user account. Assuming that a `<modify-password>` event is generated on the Subscriber channel, the following is an example of the output generated by XSLT style sheets that implement `modify-password`:

```

<input xmlns:jdbc="urn:dirxml:jdbc">
  <modify-password jdbc:op-id="0"
      jdbc:op-type="password-set-operation">
    <password>new password</password>
  </modify-password>
  <jdbc:statement jdbc:op-id="0">
    <jdbc:sql>ALTER USER jdoe IDENTIFIED BY {$$password}
  </jdbc:sql>
  </jdbc:statement>
</input>

```

NOTE: Some databases, such as Sybase Adaptive Server Enterprise and Microsoft SQL Server, differentiate between user account names and login account names. Therefore, you might need to supply the login name instead of the username.

The `<modify-password>` event is logically bound to the `ALTER USER` DDL statement by the `jdbc:op-id` and `jdbc:op-type` attributes.

The User DDL Command Transformation style sheet in the `example.xml` configuration contains sample XSLT templates that bind password maintenance DDL statements to `<modify-password>` events for all databases that support them.

11.13 Implementing Check Object Password

Unlike password set, check object password does not require embedded SQL statements or attributes. Only a user account name is required. This could be obtained from an association value (assuming that associations are being maintained manually), a directory attribute, or a database field. If stored in the directory or database, a query must be issued to retrieve the value.

The example .xml configuration file stores database user account names in database fields.

NOTE: Some databases, such as Sybase Adaptive Server Enterprise and Microsoft SQL Server, differentiate between user account names and login account names. Therefore, you might need to store two names, not just one.

To implement check object password, append a `dest-dn` attribute value to the `<check-object-password>` event. In the following example, the `dest-dn` attribute is bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
  <check-object-password dest-dn="jdoe">
    <password>whatever</password>
  </check-object-password>
</input>
```

11.14 Calling Stored Procedures and Functions

The JDBC driver enables you to use stored procedures. The ability to use the `<jdbc:call-procedure>` and `<jdbc:call-function>` elements to call stored procedures from a policy has been tested only against Oracle and is supported only on that platform.

- ♦ [Section 11.14.1, “Using Embedded SQL to Call Stored Procedures or Functions,” on page 132](#)
- ♦ [Section 11.14.2, “Using the jdbc:call-procedure Element,” on page 133](#)
- ♦ [Section 11.14.3, “Using the jdbc:call-function Element,” on page 137](#)

11.14.1 Using Embedded SQL to Call Stored Procedures or Functions

You can call stored procedures or functions in one of two ways:

- ♦ Call the procedure or function by using a Statement object.
- ♦ Call the procedure by using a Callable Statement object.

Example 1: Calling a Stored Procedure by Using a Statement

```
<!-- call syntax is Oracle -->
<jdbc:statement>
  <jdbc:sql>CALL schema.procedure-name</jdbc:sql/>
</jdbc:statement>
```

Example 2: Calling a Stored Procedure as a CallableStatement

```
<!-- call syntax is vendor agnostic -->
<jdbc:statement>
  <jdbc:call-procedure jdbc:name="schema.procedure-name"/>
</jdbc:statement>
```

Example 3: Calling a Function by Using a Statement

```
<!-- call syntax is Informix -->
<jdbc:statement>
  <jdbc:sql>EXECUTE FUNCTION schema.function-name</jdbc:sql/>
</jdbc:statement>
```

Example 4: Calling a Function as a CallableStatement

```
<!-- call syntax is vendor agnostic -->
<jdbc:statement>
  <jdbc:call-function jdbc:name="schema.function-name"/>
</jdbc:statement>
```

The principle advantage of using the CallableStatement interface is that you do not need to know the proprietary call syntaxes of each database vendor or JDBC implementation. Other advantages include the following:

- ♦ It's much easier to build procedure or function calls in the Policy Builder.
- ♦ You can differentiate between Null and empty string parameter values.
- ♦ You can call functions on all database platforms.
Oracle, for instance, doesn't support calling functions by using a statement.
- ♦ You can retrieve Out parameter values from stored procedure calls.

11.14.2 Using the jdbc:call-procedure Element

Stored procedures do not necessarily require parameters. Only a name is required. If a database supports schemas, we recommend that you schema-qualify the name. If a schema qualifier isn't provided, how names are resolved depends upon your third-party JDBC implementation and might change, depending upon driver configuration settings.

The jdbc:call-procedure element must be wrapped in a jdbc:statement element.

- ♦ [“Specifying a Procedure Name” on page 133](#)
- ♦ [“Passing In or In Out Parameter Values” on page 134](#)
- ♦ [“Handling Out or In Out Parameters” on page 135](#)
- ♦ [“Example Complex Stored Procedure Calls” on page 136](#)

Specifying a Procedure Name

```
<jdbc:call-procedure jdbc:name="schema.procedure-name"/>
```

Passing In or In Out Parameter Values

The number of jdbc:param elements specified must match the number of param elements declared in the procedure. Only jdbc:param elements corresponding to In or In Out procedure parameters can have values. Out parameters (those that can't be passed values) must be represented by an empty jdbc:param element.

- ♦ [“Calling a Procedure with No Parameters” on page 134](#)
- ♦ [“Calling a Procedure with a Null Parameter” on page 134](#)
- ♦ [“Calling a Procedure with an Empty String Parameter” on page 134](#)
- ♦ [“Calling a Procedure with a Literal Value” on page 134](#)
- ♦ [“Calling a Procedure with an Out Parameter” on page 134](#)

Calling a Procedure with No Parameters

```
<jdbc:statement>
  <jdbc:call-procedure jdbc:name="schema.procedure-name"/>
</jdbc:statement>
```

Calling a Procedure with a Null Parameter

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
  <!-- no value element = pass null -->
  <jdbc:param/>
</jdbc:call-procedure>
```

Calling a Procedure with an Empty String Parameter

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
  <!-- empty value element = pass empty string -->
  <jdbc:param>
    <jdbc:value/>
  </jdbc:param>
</jdbc:call-procedure>
```

Literals can be passed only to procedure parameters declared as In or In Out. Passed literals must be type-compatible with declared procedure parameters.

Calling a Procedure with a Literal Value

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
  <!-- non-empty value element = pass literal -->
  <jdbc:param>
    <jdbc:value>literal</jdbc:value>
  </jdbc:param>
</jdbc:call-procedure>
```

Calling a Procedure with an Out Parameter

Assuming that a procedure has two parameters, the first Out and the second In, you invoke the procedure as follows:

```

<jdbc:call-procedure jdbc:name="schema.procedure-name">
  <!-- the OUT param place -->
  <jdbc:param/>
  <!-- the IN param -->
  <jdbc:param>
    <jdbc:value>literal</jdbc:value>
  </jdbc:param>
</jdbc:param>

```

Handling Out or In Out Parameters

Stored procedures with Out or In Out parameters can return values. These values are returned by the driver and are accessible to policies. Out or In Out parameters values are returned at the same position as their corresponding declared parameter.

Also, to facilitate correlation of procedure calls and output parameter values, Out parameters contain the same event-ID value as the procedure call that generated them. This is particularly useful when multiple calls are made in the same document.

- ♦ [“Null or No Return Value” on page 135](#)
- ♦ [“Empty String Return Value” on page 135](#)
- ♦ [“Literal Return Value” on page 135](#)

Null or No Return Value

Assuming that a procedure has a single Out or In parameter, the following output is generated:

```

<output>
  <!-- no value element = OUT param returned null or IN param -->
  <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
    <jdbc:param/>
  </jdbc:out-parameters>
  <status event-id="0" level="success"/>
</output>

```

Empty String Return Value

Assuming that a procedure has a single Out or In Out parameter, the following output is generated:

```

<output>
  <!-- empty value element = returned empty string -->
  <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
    <jdbc:param>
      <jdbc:value/>
    </jdbc:param>
  </jdbc:out-parameters>
  <status event-id="0" level="success"/>
</output>

```

Literal Return Value

Assuming that a procedure has a single Out or In Out parameter, the following output is generated:

```

<output>
  <!-- no-empty value element = returned literal value -->
  <jdbc:out-parameters event-id="0" jdbc:number-of-params="2">
    <jdbc:param>
      <jdbc:value>literal</jdbc:value>
    </jdbc:param>
  </jdbc:out-parameters>
  <status event-id="0" level="success"/>
</output>

```

Example Complex Stored Procedure Calls

- ♦ [“Procedure Declaration” on page 136](#)
- ♦ [“Procedure Call from Policy” on page 136](#)
- ♦ [“Procedure Output to Policy” on page 137](#)

Procedure Declaration

This procedure uses Oracle PSQL syntax.

```

CREATE PROCEDURE indirect.p1(i1 IN VARCHAR2, io2 IN OUT VARCHAR2, o3 OUT
INTEGER, i4 IN VARCHAR2)
AS
BEGIN
  SELECT 'literal' INTO io2 FROM DUAL;
  SELECT 1 INTO o3 FROM DUAL;
END p1;

```

Procedure Call from Policy

```

<input>
  <jdbc:statement event-id="0">
    <jdbc:call-procedure jdbc:name="indirect.p1">
      <!-- i1 IN VARCHAR2 -->
      <jdbc:param>
        <!-- pass empty string -->
        <jdbc:value/>
      </jdbc:param>
      <!-- io2 IN OUT VARCHAR2 -->
      <jdbc:param>
        <!-- pass literal -->
        <jdbc:value>literal</jdbc:value>
      </jdbc:param>
      <!-- o3 OUT INTEGER -->
      <!-- param placeholder -->
      <jdbc:param/>
      <!-- o4 IN VARCHAR2 -->
      <!-- pass null -->
      <jdbc:param/>
    </jdbc:call-procedure>
  </jdbc:statement>
</input>

```


Procedure Output to Policy

```
<output>
  <jdbc:out-parameters event-id="0" jdbc:number-of-params="2">
    <jdbc:param/>
    <jdbc:param jdbc:name="IO2"
      jdbc:param-type="INOUT"
      jdbc:position="2"
      jdbc:sql-type="java.sql.Types.VARCHAR">
      <jdbc:value>literal</jdbc:value>
    </jdbc:param>
    <jdbc:param jdbc:name="O3"
      jdbc:param-type="OUT"
      jdbc:position="3"
      jdbc:sql-type="java.sql.Types.DECIMAL">
      <jdbc:value>1</jdbc:value>
    </jdbc:param>
  </jdbc:out-parameters>
  <status event-id="0" level="success"/>
</output>
```

11.14.3 Using the jdbc:call-function Element

Functions do not necessarily require parameters. Only a name is required. If a database supports schemas, we recommend that you schema-qualify the name. If a schema qualifier isn't provided, how names are resolved depends upon your third-party JDBC implementation and might change depending upon driver configuration settings.

The jdbc:call-function element must be wrapped in a jdbc:statement element.

- ♦ [“Specifying a Function Name” on page 137](#)
- ♦ [“Passing In Parameter Values” on page 137](#)
- ♦ [“Calling a Function with No Parameter” on page 137](#)
- ♦ [“Calling a Function with a Null Parameter” on page 138](#)
- ♦ [“Calling a Function with an Empty String Parameter” on page 138](#)
- ♦ [“Calling a Function with a Literal Value” on page 138](#)
- ♦ [“Handling Function Results” on page 138](#)
- ♦ [“Example Complex Function Calls” on page 140](#)

Specifying a Function Name

```
<jdbc:call-function jdbc:name="schema.function-name"/>
```

Passing In Parameter Values

The number of jdbc:param elements specified must match the number of params declared in the function.

Calling a Function with No Parameter

```
<jdbc:call-function jdbc:name="schema.function-name"/>
```

Calling a Function with a Null Parameter

```
<jdbc:call-function jdbc:name="schema.function-name">
  <!-- no value element = null -->
  <jdbc:param/>
</jdbc:call-procedure>
```

Calling a Function with an Empty String Parameter

```
<jdbc:call-function jdbc:name="schema.function-name">
  <!-- empty value element = pass empty string -->
  <jdbc:param>
    <jdbc:value/>
  </jdbc:param>
</jdbc:call-procedure>
```

Literals can be passed to function parameters declared as In. Passed literals must be type-compatible with declared function parameters.

Calling a Function with a Literal Value

```
<jdbc:call-function jdbc:name="schema.function-name">
  <!-- non-empty value element = pass literal -->
  <jdbc:param>
    <jdbc:value>literal</jdbc:value>
  </jdbc:param>
</jdbc:call-procedure>
```

Handling Function Results

Unlike stored procedures, functions do not support Out or In Out parameters. They can, however, return a single, scalar value (such as an integer or string) or return a result set. Also, to facilitate correlation of function calls and results, results contain the same event-id value as the function call that generated them. This is particularly useful when multiple calls are made in the same document.

- ♦ [“Scalar Return Value” on page 138](#)
- ♦ [“Empty Set” on page 139](#)
- ♦ [“Non-Empty Results Set” on page 139](#)
- ♦ [“Multiple Result Sets” on page 139](#)
- ♦ [“Oracle Results Set” on page 139](#)
- ♦ [“Returning Result Sets as Out Parameters” on page 139](#)
- ♦ [“Special Attributes” on page 140](#)

Scalar Return Value

Scalar return values are returned by using the same syntax as stored procedure Out parameters. The scalar return value is always returned in the first parameter position.

```
<output>
  <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
    <jdbc:param jdbc:name="return value">
```

```

        jdbc:param-type="OUT"
        jdbc:position="1"
        jdbc:sql-type="java.sql.Types.VARCHAR">
        <jdbc:value>1</jdbc:value>
    </jdbc:param>
</jdbc:out-parameters>
<status event-id="0" level="success"/>
</output>

```

Empty Set

Assuming that a function returns no results set or an empty result set, the following output is generated:

```

<output>
  <jdbc:result-set event-id="0" jdbc:number-of-rows="0"/>
  <status event-id="0" level="success"/>
</output>

```

Non-Empty Results Set

Assuming a function returns a non-empty result set, output similar to the following is generated:

```

<output>
  <jdbc:result-set event-id="0" jdbc:number-of-rows="1">
    <jdbc:row jdbc:number="1">
      <jdbc:column jdbc:name="SYSDATE"
        jdbc:position="1"
        jdbc:type="java.sql.Types.TIMESTAMP">
        <jdbc:value>2007-01-03 14:52:20.0</jdbc:value>
      </jdbc:column>
    </jdbc:row>
  </jdbc:result-set>
  <status event-id="0" level="success"/>
</output>

```

Multiple Result Sets

Multiple result sets are returned in the order returned by the function. They all share a common event-id value.

```

<output>
  <jdbc:result-set event-id="0" jdbc:number-of-rows="0"/>
  <jdbc:result-set event-id="0" jdbc:number-of-rows="0"/>
  <status event-id="0" level="success"/>
</output>

```

Oracle Results Set

Oracle's JDBC implementation uses a proprietary mechanism to return a single result set from a function. To return a result set from an Oracle function, you need to explicitly set the [jdbc:return-type](#) value to `OracleTypes.CURSOR` on the `jdbc:call-function` element.

Returning Result Sets as Out Parameters

See the special attribute [“jdbc:return-format”](#) on page 140.

Special Attributes

jdbc:return-format

This attribute can be placed on the `jdbc:call-function` element to format the first row of a returned results set as stored procedure Out parameters of the result.

This works only when the `jdbc:return-type` attribute isn't used.

```
<input>
  <jdbc:statement>
    <jdbc:call-function jdbc:name="schema.function-name"
      jdbc:return-format="return value">
    </jdbc:call-function>
  </jdbc:statement>
</input>
```

jdbc:return-type

This attribute can be placed on the `jdbc:call-function` element to allow Oracle functions to return a result set.

```
<input>
  <jdbc:statement>
    <jdbc:call-function jdbc:name="schema.function"
      jdbc:return-type="OracleTypes.CURSOR">
    </jdbc:call-function>
  </jdbc:statement>
</input>
```

Example Complex Function Calls

- ♦ [“Function Declaration” on page 140](#)
- ♦ [“Function Call from a Policy” on page 140](#)
- ♦ [“Function Results to a Policy” on page 141](#)

Function Declaration

This declaration is for Oracle PSQL syntax.

```
CREATE OR REPLACE FUNCTION indirect.f1(i1 IN VARCHAR2, i2 IN INTEGER)
  RETURN VARCHAR2
AS
  o_idu VARCHAR2(32);
BEGIN
  SELECT 'literal' INTO o_idu FROM DUAL;
  RETURN o_idu;
END f1;
```

Function Call from a Policy

```
<input>
  <jdbc:statement>
```

```

        <jdbc:call-function jdbc:name="indirect.fl">
            <jdbc:param>
                <jdbc:value>literal</jdbc:value>
            </jdbc:param>
            <jdbc:param>
                <jdbc:value>1</jdbc:value>
            </jdbc:param>
        </jdbc:call-function>
    </jdbc:statement>
</input>

```

Function Results to a Policy

```

<output>
    <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
        <jdbc:param jdbc:name="return value"
            jdbc:param-type="OUT"
            jdbc:position="1"
            jdbc:sql-type="java.sql.Types.VARCHAR">
            <jdbc:value>literal</jdbc:value>
        </jdbc:param>
    </jdbc:out-parameters>
    <status event-id="0" level="success"/>
</output>

```

11.15 Best Practices

For performance reasons, it is better to call a single stored procedure/function that contains multiple SQL statements than to embed multiple statements in an XDS document.

In the following examples, the single stored procedure or function is preferred.

Single Stored Procedure

```

<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>CALL PROCEDURE set_name('John', 'Doe')</jdbc:sql>
    </jdbc:statement>
</input>

```

Multiple Embedded Statements

```

<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>

```

```

        <jdbc:sql>UPDATE indirect.usr SET fname = 'John'                WHERE
idu = {$idu}</jdbc:sql>
    </jdbc:statement>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET lname = 'Doe'                WHERE
idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>

```

The syntax used to call stored procedures or functions varies by database. For additional information, see [“Syntaxes for Calling Stored Procedures and Functions” on page 146](#).

Supported Databases

- ♦ [Section 12.1, “Database Interoperability,” on page 143](#)
- ♦ [Section 12.2, “Supported Databases,” on page 143](#)
- ♦ [Section 12.3, “Database Characteristics,” on page 144](#)

12.1 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.

12.2 Supported Databases

The following databases or database versions have been tested and are recommended for use with this product:

Table 12-1 *Supported Databases*

Database	Minor Version
IBM DB2 Universal Database (UDB) 7	7.2 or later
IBM DB2 Universal Database (UDB) 8	8.1 or later
IBM DB2 Universal Database (UDB) 9	9.0 or later
Informix Dynamic Server (IDS) 9	9.40 or later
Informix Dynamic Server (IDS) 10	10.0 or later
Microsoft SQL Server 7	7.5, Service Pack 4 or later
Microsoft SQL Server 2000 (8)	Service Pack 3a or later
Microsoft SQL Server 2005 (9)	Service Pack 1 or later
Microsoft SQL Server 2008 (10)	
MySQL 3	3.23.58 or later
MySQL 4	4.1 or later
MySQL 5	5.0.20 or later
Oracle 8i	Release 3 (8.1.7) or later
Oracle 9i	Release 2 (9.2.0.1) or later
Oracle 10g	Release 1 (10.0.2.1) or later
Oracle 11g	Release 1 (11.1) or later

Database	Minor Version
PostgreSQL 7	7.4.6 or later
PostgreSQL 8	8.0 or later
Sybase Adaptive Server Enterprise (ASE) 12	12.5 or later
Sybase Adaptive Server Enterprise (ASE) 15	15.0 or later

IMPORTANT: The following is a list of databases that are supported with Windows Server 2008.

- ♦ Oracle 10g and 11g
- ♦ MySQL 5.0 and later
- ♦ IBM DB2 V9
- ♦ MS-SQL Server 2005 and 2008
- ♦ Informix

You can use the JDBC driver with other databases or database versions. However, Novell does not support them. To interoperate with the JDBC driver, a database must meet the following requirements:

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

12.3 Database Characteristics

- ♦ [Section 12.3.1, “Database Features,” on page 145](#)
- ♦ [Section 12.3.2, “Current Time Stamp Statements,” on page 145](#)
- ♦ [Section 12.3.3, “Syntaxes for Calling Stored Procedures and Functions,” on page 146](#)
- ♦ [Section 12.3.4, “Left Outer Join Operators,” on page 146](#)
- ♦ [Section 12.3.5, “Undelimited Identifier Case Sensitivity,” on page 147](#)
- ♦ [Section 12.3.6, “Supported Transaction Isolation Levels,” on page 147](#)
- ♦ [Section 12.3.7, “Commit Keywords,” on page 148](#)
- ♦ [Section 12.3.8, “IBM DB2 Universal Database \(UDB\),” on page 148](#)
- ♦ [Section 12.3.9, “Informix Dynamic Server \(IDS\),” on page 149](#)
- ♦ [Section 12.3.10, “Microsoft SQL Server,” on page 150](#)
- ♦ [Section 12.3.11, “MySQL,” on page 151](#)
- ♦ [Section 12.3.12, “Oracle,” on page 152](#)
- ♦ [Section 12.3.13, “PostgreSQL,” on page 153](#)
- ♦ [Section 12.3.14, “Sybase Adaptive Server Enterprise \(ASE\),” on page 153](#)

12.3.1 Database Features

Table 12-2 *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
Informix IDS 10	X	X	X ²	0	X ³	X	X	X
MS SQL 7	X	X	X	0	X	0	X	0
MS SQL 2000, 2005, and 2008	X	X	X	0	X	X	X	X
MySQL 4	0	0	X ⁴	0	0	0	0	0
MySQL 5	X	X	X ⁴	0	X	X	X	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

¹ DB2 natively supports stored procedures or functions written in Java. To write procedures by using the native SQL procedural language, install a C compiler on the database server.

² The Informix identity column keyword is `SERIAL8`.

³ Informix stored procedures cannot return values through OUT parameters.

⁴ The MySQL identity column keyword is `AUTO_INCREMENT`.

⁵ 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 or instead-of-triggers written in a variety of procedural programming languages.

12.3.2 Current Time Stamp Statements

The following table lists SQL statements used to retrieve the current date and time by database:

Table 12-3 Time Stamp Statements

Database	Current Time Stamp Statement	ANSI-Compliant
IBM DB2 UDB	SELECT (CURRENT_TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY	No
Informix IDS	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES	No
MSSQL	SELECT (CURRENT_TIMESTAMP)	Yes
MySQL	SELECT (CURRENT_TIMESTAMP)	Yes
Oracle	SELECT (SYSDATE) FROM SYS.DUAL	No
PostgreSQL	SELECT (CURRENT_TIMESTAMP)	Yes
Sybase ASE	SELECT GETDATE()	No

12.3.3 Syntaxes for Calling Stored Procedures and Functions

The following table lists the syntaxes for calling a stored procedure or function by database vendor. There's also a vendor-neutral JDBC escape syntax (see [JDBC Escape Syntax \(http://edocs.bea.com/wls/docs81/jdbc_drivers/sqlescape.html\)](http://edocs.bea.com/wls/docs81/jdbc_drivers/sqlescape.html)). Whenever possible, it is more secure to call a stored procedure or function by using the `jdbc:call-function` or `jdbc:call-procedure` syntax. See [Section 11.14, "Calling Stored Procedures and Functions," on page 132.](#) Other syntaxes should be used only when specifying procedure or function calls in driver parameters (for example, ["Post Polling Statements" on page 84](#) and ["Connection Initialization Statements" on page 58](#)).

Table 12-4 Calling a Stored Procedure or Function

Database	Stored Procedure/Function JDBC Call Syntax
IBM DB2 UDB	{call <i>schema-name.procedure-name</i> (<i>parameter-list</i>)}
Informix IDS	EXECUTE [PROCEDURE FUNCTION] <i>schema-name.routine-name</i> (<i>parameter-list</i>)
MSSQL	EXECUTE <i>schema-name.procedure-name</i> (<i>parameter-list</i>)
MySQL	[TODO]
Oracle ¹	CALL <i>schema-name.procedure-name</i> (<i>parameter-list</i>)
PostgreSQL	SELECT <i>schema-name.procedure-name</i> (<i>parameter-list</i>)
Sybase ASE	EXECUTE <i>schema-name.procedure-name</i> (<i>parameter-list</i>)

¹ Oracle's JDBC implementation does not support calling functions as a string.

12.3.4 Left Outer Join Operators

The following table lists outer join operators by database.

Table 12-5 *Outer Join Operators*

Database	Left Outer Join Operator	ANSI-Compliant
IBM DB2 UDB	LEFT OUTER JOIN	Yes
Informix IDS	LEFT OUTER JOIN	Yes
MSSQL 7.5, 2000	*=	No
MSSQL 2005	LEFT OUTER JOIN	Yes
MySQL	LEFT OUTER JOIN	Yes
Oracle	(+) [TODO]	No
PostgreSQL	LEFT OUTER JOIN	Yes
Sybase ASE	*=	No

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

12.3.5 Undelimited Identifier Case Sensitivity

Table 12-6 *Case Sensitivity for Undelimited Identifiers*

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

12.3.6 Supported Transaction Isolation Levels

Table 12-7 *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/tjvdiso.htm)

Database	None	Read Uncommitted	Read Committed	Repeatable Read	Serializable	URL
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.

12.3.7 Commit Keywords

The following table identifies the commit keywords for supported databases:

Table 12-8 *Commit Keywords*

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.

12.3.8 IBM DB2 Universal Database (UDB)

- ♦ [“Database Properties” on page 149](#)
- ♦ [“Dynamic Defaults” on page 149](#)
- ♦ [“Known Issues” on page 149](#)

Database Properties

Table 12-9 *Properties for IBM DB2 UDB*

Property	Value
Current Timestamp Statement	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Dynamic Defaults

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

Table 12-10 *Dynamically Configured IBM DB2 Universal Database Settings*

Display Name	Tag Name	Value
Current Timestamp Statement:	current-timestamp-stmt	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
Timestamp Translator class:	time-translator-class	com.novell.nds.dirxml.driver.jdbc.db.DB2Timestamp

Known Issues

The timestamp format is proprietary. See [“Known Issues” on page 154](#).

12.3.9 Informix Dynamic Server (IDS)

- ♦ [“Database Properties” on page 149](#)
- ♦ [“Dynamic Defaults” on page 150](#)
- ♦ [“Known Issues” on page 150](#)

Database Properties

Table 12-11 *Settings for Informix Dynamic Server*

Property	Value
Current Timestamp Statement	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES
Case-Sensitive?	No

Property	Value
Commit Keyword	COMMIT WORK ¹
Left Outer Join Operator	LEFT OUTER JOIN

¹ For logging and ANSI-compliant databases. Nonlogging databases do not support transactions.

Dynamic Defaults

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

Table 12-12 *Dynamically Configured Informix Dynamic Server Settings*

Display Name	Tag Name	Value
Current Timestamp Statement:	current-timestamp-stmt	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES

Known Issues

- ♦ NUMERIC or DECIMAL columns cannot be used as primary keys unless the scale (the number of digits to the right of the decimal point) is explicitly set to 0 when the table is created. By default, the scale is set to 255.
- ♦ DBAs cannot grant privileges to objects they don't own.

12.3.10 Microsoft SQL Server

- ♦ [“Database Properties” on page 150](#)
- ♦ [“Dynamic Defaults” on page 151](#)

Database Properties

Table 12-13 *Settings for Microsoft SQL Server*

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Case-Sensitive?	No
Commit Keyword	GO
Left Outer Join Operator (7, 2000)	*=
Left Outer Join Operator (2005)	LEFT OUTER JOIN

Dynamic Defaults

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

Table 12-14 *Dynamically Configured Microsoft SQL Server Settings*

Display Name	Tag Name	Value
Add default values on insert?	add-default-values-on-view-insert	true
Left outer-join operator (7, 2000):	left-outer-join-operator	*=
Left outer-join operator (2005):	left-outer-join-operator	LEFT OUTER JOIN

12.3.11 MySQL

- ♦ [“Database Properties” on page 151](#)
- ♦ [“Dynamic Defaults” on page 151](#)
- ♦ [“Known Issues” on page 152](#)

Database Properties

Table 12-15 *Settings for MySQL*

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Case-Sensitive?	Yes
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Dynamic Defaults

The following table lists database compatibility parameters that are dynamically configured at runtime for this database.

Table 12-16 *Dynamically Configured MySQL Settings*

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

Known Issues

- ♦ `TIMESTAMP` columns, when they are updated after being initially set to 0 or `NULL`, are always set to the current date and time. To compensate for this behavior, we recommend that you map Identity Vault Time and Timestamp syntaxes to `DATETIME` columns.

12.3.12 Oracle

- ♦ [“Database Properties” on page 152](#)
- ♦ [“Dynamic Defaults” on page 152](#)
- ♦ [“Limitations” on page 152](#)

Database Properties

Table 12-17 *Settings for Oracle*

Property	Value
Current Timestamp Statement	SELECT (SYSDATE) FROM SYS.DUAL
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	(+)

Dynamic Defaults

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

Table 12-18 *Dynamically Configured Oracle Settings*

Display Name	Tag Name	Value
Left outer-join operator	left-outer-join-operator	(+)
Exclude filter expression	exclude-table-filter	BIN\\${22}==\\${0}
Lock statement generator class	lock-generator-class	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator

The default exclusion filter omits dropped tables (that are visible in Oracle 10g) from the synchronization schema.

Limitations

`LONG`, `LONG RAW`, and `BLOB` columns cannot be referenced in a trigger. You can't reference columns of these types by using the `:NEW` qualifier in a trigger, including instead-of-triggers.

12.3.13 PostgreSQL

- ♦ [“Database Properties” on page 153](#)
- ♦ [“Known Issues” on page 153](#)

Database Properties

Table 12-19 *Settings for PostgreSQL*

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Known Issues

PostgreSQL does not support `<check-object-password>` events. You control authentication by manually inserting entries into the `pg_hba.conf` file.

12.3.14 Sybase Adaptive Server Enterprise (ASE)

- ♦ [“Database Properties” on page 153](#)
- ♦ [“Dynamic Defaults” on page 153](#)
- ♦ [“Known Issues” on page 154](#)

Database Properties

Table 12-20 *Settings for Sybase ASE*

Property	Value
Current Timestamp Statement	SELECT GETDATE()
Case-Sensitive?	Yes
Commit Keyword	GO
Left Outer Join Operator	*=

Dynamic Defaults

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

Table 12-21 *Dynamically Configured Sybase ASE Settings*

Display Name	Tag Name	Value
Current timestamp statement	current-timestamp-stmt	SELECT GETDATE()
Left outer-join operator	left-outer-join-operator	*=
Timestamp Translator class	time-translator-class	com.novell.nds.dirxml.driver.jdbc.db.SybaseTimestamp

Known Issues

- ♦ Padding and truncation of binary values.

To ensure ANSI-compliant padding and truncation behavior for binary values, make sure that binary column types (other than `IMAGE`) meet the following criteria:

- ♦ They are exactly the size of the eDirectory™ attribute that maps to them.
- ♦ They are constrained `NOT NULL`.
- ♦ They are added to the Publisher and Subscriber Creation policies.

If they are constrained `NULL`, trailing zeros, which are significant to eDirectory, are truncated. If binary columns exceed the size of their respective eDirectory attributes, extra 0s are appended to the value.

The recommended solution is to use only the `IMAGE` data type when synchronizing binary values.

- ♦ `DATETIME` fractions of a second are rounded. Sybase Timestamps are at best accurate to 1/300th of a second (approximately .003 seconds). The database server rounds to the nearest 1/300th of a second as opposed to the nearest 1/1000th of a second (.001 seconds or 1 millisecond).
- ♦ Timestamp formats are proprietary.

- ♦ [Section 13.1, “Third-Party JDBC Driver Interoperability,” on page 155](#)
- ♦ [Section 13.2, “Third-Party JDBC Driver Types,” on page 155](#)
- ♦ [Section 13.3, “Third-Party Jar File Placement,” on page 156](#)
- ♦ [Section 13.4, “Supported Third-Party JDBC Drivers \(Recommended\),” on page 157](#)
- ♦ [Section 13.5, “Supported Third-Party JDBC Drivers \(Not Recommended\),” on page 169](#)
- ♦ [Section 13.6, “Deprecated Third-Party JDBC Drivers,” on page 175](#)
- ♦ [Section 13.7, “Other Third-Party JDBC Drivers,” on page 175](#)
- ♦ [Section 13.8, “Security Issues,” on page 176](#)

13.1 Third-Party JDBC Driver Interoperability

The Identity Manager JDBC driver is designed to interoperate with a specific set of third-party JDBC drivers, instead of a specific set of databases. In fact, the third-party JDBC driver, not the database, is the primary determinant of whether the JDBC driver works against any given database. As a general rule, if the JDBC driver interoperates well with a given third-party JDBC driver, it interoperates well with databases and database versions that the third-party driver supports.

We strongly recommend that you use the third-party JDBC drivers supplied by major enterprise database vendors whenever possible, such as those listed in [Section 13.4, “Supported Third-Party JDBC Drivers \(Recommended\),” on page 157](#). They are usually free, mature, and known to interoperate well with the JDBC driver and the databases they target. You can use other third-party drivers, but Novell does not support them.

In general, most third-party drivers are backward compatible. However, even if they are backward compatible, they are generally not forward compatible. Anytime a database server is upgraded, the third-party driver used with this product should probably be updated as well.

Also, as a general rule, we recommend that you use the latest version of a third-party driver, unless otherwise noted.

13.2 Third-Party JDBC Driver Types

The following sections describe four third-party JDBC driver types and explain the recommended type for use with the Identity Manager JDBC driver:

- ♦ [Section 13.2.1, “Driver Types,” on page 155](#)
- ♦ [Section 13.2.2, “Which Type To Use?,” on page 156](#)

13.2.1 Driver Types

There are four types of third-party drivers:

- ♦ **Type 1:** A third-party JDBC driver that is partially Java and communicates indirectly with a database server through a native ODBC driver.

Type 1 drivers serve as a JDBC-ODBC bridge. Sun provides a JDBC-ODBC bridge driver for experimental use and for situations when no other type of third-party JDBC driver is available.

- ♦ **Type 2:** A third-party JDBC driver that is part Java and communicates indirectly with a database server through its native client APIs.
- ♦ **Type 3:** A third-party JDBC driver that is pure Java and communicates indirectly with a database server through a middleware server.
- ♦ **Type 4:** A third-party JDBC driver that is pure Java and communicates directly with a database server.

13.2.2 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.

If you choose to use a type 1 or type 2 driver (one containing native code) with the JDBC driver, use the Remote Loader to ensure the integrity of the directory process.

13.3 Third-Party Jar File Placement

The following tables identify the paths where third-party JDBC driver jar files should be placed on an Identity Manager or Remote Loader server, assuming default installation paths.

Table 13-1 *Locations for jar Files: Identity Manager Server*

Platform	Directory Path
Solaris, Linux, or AIX	/opt/novell/eDirectory/lib/dirxml/classes (eDirectory 8.8.5)
Windows	novell\NDS\lib

Table 13-2 *Locations for jar Files: Remote Loader Server*

Platform	Directory Path
Solaris, Linux, or AIX	/opt/novell/eDirectory/lib/dirxml/classes (eDirectory 8.8.5)
Windows	novell\RemoteLoader\lib

13.4 Supported Third-Party JDBC Drivers (Recommended)

This section discusses supported third-party drivers. Using one of these supported drivers is recommended.

- ♦ [Section 13.4.1, “Third-Party JDBC Driver Features,” on page 157](#)
- ♦ [Section 13.4.2, “JDBC URL Syntaxes,” on page 157](#)
- ♦ [Section 13.4.3, “JDBC Driver Class Names,” on page 158](#)
- ♦ [Section 13.4.4, “IBM DB2 Universal Database Type 4 JDBC Driver,” on page 158](#)
- ♦ [Section 13.4.5, “Informix JDBC Driver,” on page 160](#)
- ♦ [Section 13.4.6, “jTDS JDBC Driver,” on page 161](#)
- ♦ [Section 13.4.7, “MySQL Connector/J JDBC Driver,” on page 163](#)
- ♦ [Section 13.4.8, “Oracle Thin Client JDBC Driver,” on page 164](#)
- ♦ [Section 13.4.9, “Oracle OCI JDBC Driver,” on page 166](#)
- ♦ [Section 13.4.10, “PostgreSQL JDBC Driver,” on page 167](#)
- ♦ [Section 13.4.11, “Sybase Adaptive Server Enterprise JConnect JDBC Driver,” on page 168](#)

Additional drivers are supported but not recommended. For a list of these drivers, see [Section 13.5, “Supported Third-Party JDBC Drivers \(Not Recommended\),” on page 169](#).

13.4.1 Third-Party JDBC Driver Features

The following table summarizes third-party JDBC driver features:

Table 13-3 *Third-Party JDBC Driver Features*

Driver	Supports Encrypted Transport?	Supports Retrieval of Auto-Generated Keys?
IBM DB2 UDB Type 4	No	No
Informix	No	No
MySQL Connector/J	Yes	Yes
jTDS	Yes	Yes
Oracle Thin Client	Yes	No
Oracle OCI	Yes	No
PostgreSQL	Yes, for JDBC 3 (Java 1.4) versions and later	No
Sybase jConnect*	Yes	No

13.4.2 JDBC URL Syntaxes

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

Table 13-4 *URL Syntaxes*

Third-Party JDBC Driver	JDBC URL Syntax
IBM DB2 UDB Type 4, Universal	<code>jdbc:db2://ip-address:50000/database-name</code>
Informix	<code>jdbc:informix-sqli://ip-address:1526/database-name:informixserver=server-id</code>
jTDS	<code>jdbc:jtds:sqlserver://ip-address/database-name</code>
MySQL Connector/J	<code>jdbc:mysql://ip-address:3306/database-name</code>
Oracle OCI	<code>jdbc:oracle:oci8:@tns-name</code>
Oracle Thin Client	<code>jdbc:oracle:thin:@ip-address:1521:sid</code>
PostgreSQL	<code>jdbc:postgresql://ip-address:5432/database-name</code>
Sybase jConnect	<code>jdbc:sybase:Tds:ip-address:2048/database-name</code>

This information is used in conjunction with the Authentication Context parameter. For information on this parameter, see [“Authentication Context” on page 46](#).

13.4.3 JDBC Driver Class Names

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

Table 13-5 *Class Names of Third-Party JDBC Drivers*

Third-Party JDBC Driver	Class Name
IBM DB2 UDB Type 4, Universal	<code>com.ibm.db2.jcc.DB2Driver</code>
Informix	<code>com.informix.jdbc.IfxDriver</code>
jTDS	<code>net.sourceforge.jtds.jdbc.Driver</code>
MySQL Connector/J	<code>org.gjt.mm.mysql.Driver</code>
Oracle OCI	<code>oracle.jdbc.driver.OracleDriver</code>
Oracle Thin Client	<code>oracle.jdbc.driver.OracleDriver</code>
PostgreSQL	<code>org.postgresql.Driver</code>
Sybase jConnect 5.5	<code>com.sybase.jdbc2.jdbc.SybDriver</code>
Sybase jConnect 6.05	<code>com.sybase.jdbc3.jdbc.SybDriver</code>

This information is used in conjunction with the JDBC Driver Class Name parameter. For information on this parameter, see [“Third-Party JDBC Driver Class Name” on page 49](#).

13.4.4 IBM DB2 Universal Database Type 4 JDBC Driver

- ♦ [“Driver Information” on page 159](#)
- ♦ [“Compatibility” on page 159](#)

- ♦ “Security” on page 159
- ♦ “Known Issues” on page 159

Driver Information

Table 13-6 IBM DB2 Driver: Type 4

Supported Database Versions	IBM DB2 7.2 or later, 8.1 or later, 9.0 or later
Class Name	com.ibm.db2.jcc.DB2Driver
Type	4
URL Syntax	<code>jdbc:db2://ip-address:50000/database-name</code>
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. <code>file:///database-installation-directory/java</code>
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)

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

Compatibility

The IBM DB2 driver is backward compatible. However, 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 be time consuming and frustrating. A log file (`db2diag.log` on the database server) can often provide additional debugging information. In addition, all error codes are documented and available online.

13.4.5 Informix JDBC Driver

- ♦ “Driver Information” on page 160
- ♦ “Compatibility” on page 160
- ♦ “Security” on page 160
- ♦ “Required Parameter Settings for ANSI-Compliant Databases” on page 160
- ♦ “Dynamic Parameter Defaults” on page 161
- ♦ “Known Issues” on page 161

Driver Information

Table 13-7 *Informix JDBC Driver*

Supported Database Versions	Dynamic Server 9.40 or later, 10.0 or later
Class Name	<code>com.informix.jdbc.IfxDriver</code>
Type	4
URL Syntax	<code>jdbc:informix-sqli://ip-address:1526/database-name:informixserver=server-id</code>
Download Instructions	Download URL (http://www-306.ibm.com/software/data/informix/tools/jdbc)
Filenames (7, 9)	<code>ifxjdbc.jar</code> , <code>ifxjdbcx.jar</code> (optional)
Filenames (10)	<code>jdbc.jar</code> , <code>jdbcx.jar</code> (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 JDBC driver to interoperate with the Informix driver against ANSI-compliant databases.

Table 13-8 *Driver Settings for ANSI-Compliant Databases*

Display Name	Tag Name	Value
Supports schemas in metadata retrieval?	supports-schemas-in-metadata-retrieval	false
	See “Supports Schemas in Metadata Retrieval?” on page 67.	
Force username case:	force-username-case	upper
	See “Force Username Case” on page 65.	

Dynamic Parameter Defaults

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

Table 13-9 *Informix JDBC Settings Not to Override*

Display Name	Tag Name	Value
Function return method:	function-return-method	result set
	See “Function Return Method” on page 67.	

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 67 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 65 to `upper`.

13.4.6 jTDS JDBC Driver

- ♦ [“Driver Information”](#) on page 162
- ♦ [“Limitations”](#) on page 162
- ♦ [“Compatibility”](#) on page 162
- ♦ [“Security”](#) on page 162
- ♦ [“URL Properties”](#) on page 162

Driver Information

Table 13-10 *jTDS Driver Settings*

Supported Database Versions:	Microsoft SQL Server 7.5, Service Pack 4 or later, 2000 (8) Service Pack 3a or later, 2005 (9) Service Pack 1 or later, 2008 (10), Sybase Adaptive Server Enterprise (ASE) 12.5 or later, Sybase Adaptive Server Enterprise (ASE) 15.0 or later
Class Name	net.sourceforge.jtds.jdbc.Driver
Type	4 (2 if NTLM or SSO authentication is enabled)
URL Syntax	<i>jdbc:jtds:sqlserver://ip-address/database-name</i> <i>jdbc:jtds:sybase://ip-address/database-name</i>
Download Instructions	The jTDS Project (http://jtds.sourceforge.net/)
Filenames	jtds-<version>.jar

Limitations

The jTDS JDBC driver does not support views or stored procedures. Novell recommends that you use the Microsoft 2000 JDBC driver when Subscribing to views.

Compatibility

The jTDS driver works with all versions of Microsoft SQL Server. It also supports all versions of Sybase ASE, but it hasn't been tested by Novell against that database server yet. Driver updates are infrequent.

Security

The jTDS driver supports encrypted transport.

URL Properties

Delimit URL properties by using a semicolon (;).

The following table lists values for the domain URL property for the jTDS driver.

Table 13-11 *Values for the Domain URL Property*

Legal Value	Description
<any-domain-name>	When a domain name is specified, either NTLM or SSO authentication can be used. NTLM authentication is selected when a username and password are supplied. SSO authentication is selected when a username and password are not supplied.
<no-value>	JDBC authentication is used.

The following table lists values for the SSL URL property for the jTDS driver.

Table 13-12 *Values for the SSL URL Property*

Legal Value	Description
off	SSL is not used. This is the default.
request	SSL is used if the server supports it.
require	SSL is required. An exception is thrown if the server doesn't support it.
authenticate	SSL is required. An exception is thrown if the server doesn't support it. In addition, the server's certificate must be signed by a trusted certificate authority.

13.4.7 MySQL Connector/J JDBC Driver

- ♦ “Driver Information” on page 163
- ♦ “Compatibility” on page 163
- ♦ “Security” on page 164
- ♦ “Required Parameter Settings for MyISAM Tables” on page 164

Driver Information

Table 13-13 *MySQL Connector/J JDBC Driver Settings*

Supported Database Versions	3.23.58 or later, 4.1 or later, 5.0.20 or later
Class Name	org.gjt.mm.mysql.Driver
Type	4
URL Syntax	<code>jdbc:mysql://ip-address:3306/database-name</code>
Download Instructions	Download and extract. The <code>jar</code> file is located in the <code>extract-dir/mysql-connector-java-version</code> directory. MySQL Connector/J (http://www.mysql.com/products/connector/j/)
Filename	<code>mysql-connector-java-version-bin.jar</code>
Documentation URLs	MySQL Connector/J Documentation (http://dev.mysql.com/doc/refman/5.0/en/java-connector.html) Connecting Over SSL (http://dev.mysql.com/doc/refman/5.0/en/cj-using-ssl.html)

Also see “Generation/Retrieval Method (Table-Global)” on page 72.

Compatibility

The Connector/J driver is backward compatible. Database server updates are frequent. Driver updates are infrequent.

Security

The Connector/J driver supports JSSE (Java Secure Sockets Extension) SSL-encrypted transport.

Required Parameter Settings for MyISAM Tables

The following table lists driver parameters that you must set so that the JDBC driver can interoperate with the Connector/J driver against MyISAM tables.

Table 13-14 *Settings for MyISAM Tables*

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

13.4.8 Oracle Thin Client JDBC Driver

- ♦ “Driver Information” on page 164
- ♦ “Compatibility” on page 165
- ♦ “Security” on page 165
- ♦ “Dynamic Parameter Defaults” on page 165
- ♦ “Connection Properties” on page 165
- ♦ “Known Issues” on page 165

Driver Information

Table 13-15 *Oracle Thin Client Settings*

Supported Database Versions	8i (Release 3 (8.1.7) or later), 9i (Release 2 (9.2.0.1) or later), 10g (Release 1 (10.0.2.1) or later), 11g (Release 1 (11.1) or later)
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)
Filenames	ojdbc14.jar, orail8n.jar (optional) Filenames for different JVM versions (http://www.oracle.com/technology/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm#02_07)
Documentation URLs	Oracle Advanced Security (http://www.stanford.edu/dept/itss/docs/oracle/10g/java.101/b10979/clientsec.htm) JDBC FAQ (http://www.oracle.com/technology/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm)

Compatibility

The Thin Client driver is backward compatible. Database server updates and driver updates are infrequent.

Oracle releases thin client drivers for various JVMs. Even though all of them work with this product, we recommend that you use the 1.4 version.

Security

The Thin Client driver supports Oracle Advanced Security encrypted transport.

Dynamic Parameter Defaults

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

Table 13-16 *Oracle Thin Client Settings Not to Override*

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

Connection Properties

The following table lists important connection properties for this driver.

Table 13-17 *Oracle Thin Client: Connection Properties*

Property	Significance
includeSynonyms	If the value of this property is <code>true</code> , synonym column metadata is available.
ORACLE.NET.ENCRIPTION_CLIENT	Defines the level of security that the client wants to negotiate with the server.
ORACLE.NET.ENCRIPTION_TYPES_CLIENT	Defines the encryption algorithm to be used.
ORACLE.NET.CRYPTO_CHECKSUM_CLIENT	Defines the level of security that it wants to negotiate with the server for data integrity.
ORACLE.NET.CRYPTO_CHEKSUM_TYPES_CLIENT	Defines the data integrity algorithm to be used.

Known Issues

- ♦ High CPU utilization triggered by execution of embedded SQL statements:

The most common problem experienced with this driver is high CPU utilization. As a result, this driver always indicates that more results are available from calls to method `java.sql.Statement.execute(String stmt)`, which can lead to an infinite loop condition. This condition occurs only if all the following happen:

- ♦ A value other than `single`, `no`, or `one` in the driver compatibility parameter “[Number of Returned Result Sets](#)” on page 63 is being executed.

- ♦ An embedded SQL statement is being executed.
- ♦ The type of statement is not explicitly specified.

To avoid the conditions that produce high CPU utilization:

- ♦ Do not explicitly set this parameter. Defer to the dynamic default value.
- ♦ Always place a `jdbc:type` attribute on embedded `<jdbc:statement>` elements.

NOTE: The `jdbc` namespace prefix must map to `urn:dirxml:jdbc`.

- ♦ Can't retrieve synonym column metadata.
The connection property `includeSynonyms` must be set to `true`.
- ♦ Can't see synonym table primary key constraint.
The only known solution to this problem is to use a view.

13.4.9 Oracle OCI JDBC Driver

Table 13-18 *Oracle OCI JDBC Driver Settings*

Supported Database Versions	8i (Release 3 (8.1.7) or later), 9i (Release 2 (9.2.0.1) or later), 10g (Release 1 (10.0.2.1) or later), 11g (Release 1 (11.1) or later)
Class Name	<code>oracle.jdbc.driver.OracleDriver</code>
Type	2
URL Syntax	<code>jdbc:oracle:oci8:@<i>tns-name</i></code>
Download Instructions	<p>The SQLNet infrastructure is the main requirement for OCI. SQLNet can run on any platform that Oracle supports, not just Linux.</p> <p>For Linux, register for free and download the following:</p> <ul style="list-style-type: none"> ♦ The Oracle Instant Client (<code>instantclient-basic-linux32-10.2.0.1-20050713.zip</code>) from Instant Client Downloads (http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html). ♦ The Oracle SQLPlus binary (<code>instantclient-sqlplus-linux32-10.2.0.1-20050713.zip</code>) from Instant Client Downloads (http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html).
Filenames	<p><code>ojdbc14.jar</code>, <code>orail8n.jar</code> (optional)</p> <p>Filenames for different JVM versions (http://www.oracle.com/technology/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm#02_07)</p>

Documentation URLs	Oracle Call Interface (http://www.oracle.com/technology/tech/oci/index.html) OCI FAQ (http://www.oracle.com/technology/tech/oci/htdocs/oci_faq.html) Oracle Advanced Security (http://www.stanford.edu/dept/itss/docs/oracle/10g/java.101/b10979/clientsec.htm) Instant Client (http://www.oracle.com/technology/tech/oci/instantclient/index.html) Instant Client (http://download-west.oracle.com/docs/cd/B12037_01/java.101/b10979/instclient.htm#CHDGDIGG)
--------------------	--

You can install SQLNet by doing either of the following:

- ♦ Use the Instant Client, which bypasses unneeded components of the full version.
- ♦ Download the full package from Oracle.

If the database is running on the same server as Identity Manager, you don't need to install SQLNet because SQLNet comes as standard on the database server.

The Oracle OCI driver is essentially the same as the Thin Client driver. See [Section 13.4.8, “Oracle Thin Client JDBC Driver,” on page 164](#). The OCI client differs in the following ways:

- ♦ The OCI Client supports clustering, failover, and high availability.
- ♦ The OCI Client has additional security options.

For information on setting up the Oracle OCI Client, see [Appendix M, “Setting Up an OCI Client on Linux,” on page 215](#).

13.4.10 PostgreSQL JDBC Driver

- ♦ [“Driver Information” on page 167](#)
- ♦ [“Compatibility” on page 168](#)
- ♦ [“Security” on page 168](#)

Driver Information

Table 13-19 *PostgreSQL JDBC Driver Settings*

Supported Database Versions	7.4.6 or later, 8.0 or later
Class Name	org.postgresql.Driver
Type	4
URL Syntax	<code>jdbc:postgresql://ip-address:5432/database-name</code>
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)
--------------------	--

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.

13.4.11 Sybase Adaptive Server Enterprise JConnect JDBC Driver

- ♦ “Driver Information” on page 168
- ♦ “Compatibility” on page 168
- ♦ “Security” on page 169
- ♦ “Connection Properties” on page 169

Driver Information

Table 13-20 *Sybase Adaptive Server Enterprise Driver Settings*

Supported Database Versions	Adaptive Server Enterprise 12..5 or later, 15.0 or later
Class Name	com.sybase.jdbc2.jdbc.SybDriver (for jconn2.jar) com.sybase.jdbc3.jdbc.SybDriver (for jconn3.jar)
Type	4
URL Syntax	<code>jdbc:sybase:Tds:ip-address:2048/database-name</code>
Download Instructions	Sybase Downloads (http://www.sybase.com/downloads)
Filenames	jconn2.jar or jconn3.jar
Documentation URLs	JConnect Documentation (http://sybooks.sybase.com/onlinebooks/group-jc/jcg0600e/prjdbc)

For JDBC 3 (Java 1.4) versions and later.

Compatibility

The Adaptive Server driver is backward compatible. Database server updates and driver updates are infrequent.

Security

The Adaptive Server driver supports SSL-encrypted transport. To enable SSL encryption, you must specify a custom socket implementation via the SYB SOCKET_FACTORY connection property. For additional information on how to set connection properties, see [“Connection Properties” on page 58](#).

Connection Properties

The SYB SOCKET_FACTORY property can be used to specify the class name of a custom socket implementation that supports encrypted transport.

13.5 Supported Third-Party JDBC Drivers (Not Recommended)

This section identifies third-party JDBC drivers that are supported but whose use with the Identity Manager JDBC driver is not recommended:

- ♦ [Section 13.5.1, “Third-Party JDBC Driver Features,” on page 169](#)
- ♦ [Section 13.5.2, “JDBC URL Syntaxes,” on page 169](#)
- ♦ [Section 13.5.3, “JDBC Driver Class Names,” on page 170](#)
- ♦ [Section 13.5.4, “IBM DB2 Universal Database JDBC Driver,” on page 170](#)
- ♦ [Section 13.5.5, “Microsoft SQL Server 2000 Driver for JDBC,” on page 172](#)
- ♦ [Section 13.5.6, “Microsoft SQL Server 2005 JDBC Driver,” on page 174](#)

For information about supported third-party drivers that are recommended, see [Section 13.5, “Supported Third-Party JDBC Drivers \(Not Recommended\),” on page 169](#).

13.5.1 Third-Party JDBC Driver Features

The following table summarizes third-party JDBC driver features:

Table 13-21 *Third-Party JDBC Driver Features*

Driver	Supports Encrypted Transport?	Supports Retrieval of Auto-Generated Keys?
IBM DB2 UDB Type 3	No	No
Microsoft 2000	No	No
Microsoft 2005	Yes	Yes

13.5.2 JDBC URL Syntaxes

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

Table 13-22 *URL Syntaxes*

Third-Party JDBC Driver	JDBC URL Syntax
IBM DB2 UDB Type 3	<code>jdbc:db2://ip-address:6789/database-name</code>
Microsoft SQL Server 7, 2000	<code>jdbc:microsoft:sqlserver://ip-address-or-dns-name:1433;DatabaseName=database-name</code>
Microsoft SQL Server 2005	<code>jdbc:sqlserver://ip-address-or-dns-name:1433;databaseName=database-name</code>

This information is used in conjunction with the Authentication Context parameter. For information on this parameter, see [“Authentication Context” on page 46](#).

13.5.3 JDBC Driver Class Names

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

Table 13-23 *Class Names of Third-Party JDBC Drivers*

Third-Party JDBC Driver	Class Name
IBM DB2 UDB Type 3	<code>COM.ibm.db2.jdbc.net.DB2Driver</code>
Microsoft 2000	<code>com.microsoft.jdbc.sqlserver.SQLServerDriver</code>
Microsoft 2005	<code>com.microsoft.sqlserver.jdbc.SQLServerDriver</code>

This information is used in conjunction with the JDBC Driver Class Name parameter. For information on this parameter, see [“Third-Party JDBC Driver Class Name” on page 49](#).

13.5.4 IBM DB2 Universal Database JDBC Driver

- ♦ [“Driver Information” on page 170](#)
- ♦ [“Compatibility” on page 171](#)
- ♦ [“Security” on page 171](#)
- ♦ [“Known Issues” on page 171](#)

Driver Information

Table 13-24 *IBM DB2 Driver Settings*

	Type 3 Driver	Type 4 Driver
Supported Database Versions	8.x and 9.x	8.x and 9.x
Class Name	<code>COM.ibm.db2.jdbc.net.DB2Driver</code>	<code>com.ibm.db2.jcc.DB2Driver</code>

	Type 3 Driver	Type 4 Driver
URL Syntax	jdbc:db2://ip-address:6789/ <i>database-name</i>	jdbc:db2://ip-address:50000/ <i>database-name</i>
Download Instructions	Copy the file from the database server. file:///database-installation-directory/java	Copy the file from the database server. file:///database-installation-directory/java
File Name	db2java.zip	db2jcc.jar
Documentation URLs	DB2 Information Center (http://publib.boulder.ibm.com/infocenter/db2v7luw/) JDBC Programming (http://publib.boulder.ibm.com/infocenter/db2v7luw/index.jsp?topic=/com.ibm.db2v7.doc/db2a0/db2a0159.htm)	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)

The type 3 driver was deprecated as of DB2 UDB 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.

The IBM DB2 driver must be updated on the Identity Manager or Remote Loader server every time the target database is updated, even if only at the FixPack level.

Security

The IBM DB2 driver does not support encrypted transport.

Known Issues

- ♦ A version mismatch usually results in connectivity-related failures.

The most common problem experienced with the IBM DB2 driver is because of a driver/database version mismatch. The symptom of a version mismatch is connectivity-related failures such as CLI0601E Invalid statement handle or statement is closed. To remedy the problem, overwrite the db2java.zip file on the Identity Manager or Remote Loader server with the version installed on the database server.

- ♦ It's very difficult to diagnose and remedy Java-related errors on the database server.

Numerous error conditions and error-codes can arise when you attempt to install and execute user-defined stored procedures and functions written in Java. Diagnosing them can be time consuming and frustrating. A log file (db2diag.log on the database server) can often provide additional debugging information. In addition, all error codes are documented and available online.

13.5.5 Microsoft SQL Server 2000 Driver for JDBC

- ♦ “Driver Information” on page 172
- ♦ “Compatibility” on page 172
- ♦ “Security” on page 172
- ♦ “URL Properties” on page 172
- ♦ “Dynamic Parameter Defaults” on page 173
- ♦ “Known Issues” on page 173

Driver Information

Table 13-25 *Microsoft SQL Server 2000 Driver Settings*

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

The filename, URL syntax and classname differ (often subtly) from those of the 2005 driver.

Compatibility

The SQL Server 2000 driver only works with SQL Server 2000. However, 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 semicolon (;).

Table 13-26 *Values for the SelectMethod URL Property*

Legal Value	Description
<code>direct</code>	The default value. Doesn’t allow for multiple active statements on a single connection
<code>cursor</code>	Allows for multiple active statements on a single connection

Dynamic Parameter Defaults

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

Table 13-27 *SQL Server 2000 Settings Not to Override*

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

Known Issues

- ♦ Can't start a 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) (<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 the driver compatibility parameter "Reuse Statements?" on page 62 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 article:

- ♦ [Article 313181](http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B313181) (<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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html>) value for native `UNIQUEIDENTIFIER` columns. To compensate, the JDBC driver 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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html>) because it best mirrored the native database type, which is 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) (<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, remove all invalidated associations and reassociate all previously associated objects.

13.5.6 Microsoft SQL Server 2005 JDBC Driver

- ♦ “Driver Information” on page 174
- ♦ “Compatibility” on page 174
- ♦ “Security” on page 174
- ♦ “URL Properties” on page 174

Driver Information

Table 13-28 *Microsoft SQL Server 2005 Driver Settings*

Supported Database Versions:	2005 (9)
Class Name	com.microsoft.sqlserver.jdbc.SQLServerDriver
Type	4 (2 if integrated security is enabled)
URL Syntax	<code>jdbc:sqlserver://ip-address-or-dns-name:1433;databaseName=database-name</code>
Download Instructions	Microsoft SQL Server 2005 JDBC Driver (http://msdn2.microsoft.com/en-us/data/aa937724.aspx)
Filenames	<code>sqljdbc.jar</code>

The filename, URL syntax, and classname differ (often subtly) from those of the 2000 driver.

Compatibility

The SQL Server 2005 driver works only with SQL Server 2005. Database server and driver updates are infrequent.

Security

The SQL Server 2005 driver does not support encrypted transport.

URL Properties

Delimit URL properties by using a semicolon (;).

The following table lists values for the `integratedSecurity` URL property for the SQL Server 2005 driver.

Table 13-29 *Values for the integratedSecurity URL Property*

Legal Value	Description
<code>false</code>	The default value. JDBC authentication is used.
<code>true</code>	Windows process-level authentication is used.

13.6 Deprecated Third-Party JDBC Drivers

The BEA WebLogic jDriver for Microsoft SQL Server is no longer supported.

13.7 Other Third-Party JDBC Drivers

This section lists an unsupported driver of interest (IBM Toolbox for Java/JTOpen) and discusses how to assess the feasibility of using unsupported third-party JDBC drivers with this product.

- ♦ [Section 13.7.1, “IBM Toolbox for Java/JTOpen,” on page 175](#)
- ♦ [Section 13.7.2, “Minimum Third-Party JDBC Driver Requirements,” on page 175](#)
- ♦ [Section 13.7.3, “Considerations When Using Other Third-Party JDBC Drivers,” on page 176](#)

13.7.1 IBM Toolbox for Java/JTOpen

Table 13-30 *Settings for IBM Toolbox for Java/JTOpen*

Database	IBM Toolbox for Java/JTOpen <ul style="list-style-type: none">♦ iSeries Toolbox for Java (alias)♦ AS/400 Toolbox for Java (alias)
Class Name	com.ibm.as400.access.AS400JDBCDriver
Type	4
URL Syntax	jdbc:as400://ip-address/database-name
Download Instructions	Download URLs for JTOpen <ul style="list-style-type: none">♦ JTOpen (http://jt400.sourceforge.net)♦ Toolbox for Java/JTOpen (http://www-03.ibm.com/servers/eserver/iseries/toolbox/downloads.html)
Filenames	jt400.jar
Documentation URLs	Toolbox for Java/JTOpen (http://www-03.ibm.com/servers/eserver/iseries/toolbox/)

If you use the IBM Toolbox for Java/JTOpen driver, you must manually enter values for the JDBC Driver Class Name and Authentication Context parameters. The settings are not automatically populated. See [“Third-Party JDBC Driver Class Name” on page 49](#) and [“Authentication Context” on page 46](#).

13.7.2 Minimum Third-Party JDBC Driver Requirements

The JDBC driver might not interoperate with all third-party JDBC drivers. If you use an unsupported third-party JDBC driver, it must meet the following requirements:

- ♦ Support required metadata methods

For a current list of the required and optional `java.sql.DatabaseMetaData` method calls that the JDBC driver makes, see [Appendix F, “java.sql.DatabaseMetaData Methods,” on page 197](#).

- ♦ Support other required JDBC methods

For a list of required JDBC methods that the JDBC driver uses, refer to [Appendix G, “JDBC Interface Methods,” on page 199](#). You can use this list in collaboration with third-party driver documentation to identify potential incompatibilities.

13.7.3 Considerations When Using Other Third-Party JDBC Drivers

- ♦ Because the JDBC driver is directly dependent upon third-party JDBC driver implementations, bugs in those implementations might cause this product to malfunction.

To assist you in debugging third-party JDBC drivers, the JDBC driver supports the following:

- ♦ Tracing at the JDBC API level (level 6)
- ♦ Third-party JDBC driver (level 7) tracing
- ♦ Stored procedure or function support is a likely point of failure.
- ♦ You probably need to write a custom driver descriptor file.

Specifically, you need to categorize error codes and SQL states for the third-party driver that you are using.

13.8 Security Issues

To ensure that a secure connection exists between the Identity Manager JDBC driver and a third-party driver, we recommend the following:

- ♦ Run the JDBC driver remotely on the database server.
- ♦ Use SSL to encrypt communications between the Identity Manager server and the database server.

If you cannot run the JDBC driver remotely, you might want to use a type 2 or type 3 JDBC driver. These driver types often facilitate a greater degree of security through middleware servers or client APIs unavailable to other JDBC driver types. Some type 4 drivers support encrypted transport, but encryption is the exception rather than the rule.

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

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

- ♦ [Section 14.1, “Independent Operations,” on page 177](#)
- ♦ [Section 14.2, “Before You Begin,” on page 178](#)
- ♦ [Section 14.3, “Using the Association Utility,” on page 178](#)
- ♦ [Section 14.4, “Parameters for Searching and Replacing,” on page 179](#)

14.1 Independent Operations

The Association utility supports seven independent operations:

Table 14-1 *Independent Operations*

Operation	Description	Read-Write Functionality
1	Lists objects associated with a driver (default).	Read-only
2	Lists objects that have multiple associations to a driver.	Read-only
3	Lists objects that have invalid associations to a driver. An association is invalid if: <ul style="list-style-type: none"> ♦ It is malformed. For example, the association is missing the schema RDN, missing the table RDN, or the schema keyword is misspelled. ♦ It contains database identifiers that do not map to identifiers in the target database. For example, an association includes a mapping to a table that does not exist. ♦ It maps to no row or multiple rows. An association is broken if it doesn't map to a row. Also, associations aren't unique if they map to more than one row. 	Read-only
4	Lists objects that need to be normalized. A normalized association is valid, correctly ordered, and uses the correct case. Normal case is uppercase for case-insensitive databases and mixed case for case-sensitive databases.	Read-only
5	Normalizes object associations listed during operation 4.	Write

Operation	Description	Read-Write Functionality
6	<p>Lists object associations to be modified.</p> <p>Allows for global replacement of schema, table, and column names based on search criteria.</p> <p>This operation requires two parameters (oldRDN and newRDN). See “Parameters for Searching and Replacing” on page 179.</p>	Read-only
7	<p>Modifies object associations listed during operation 6.</p> <p>This operation requires two parameters (oldRDN and newRDN). See “Parameters for Searching and Replacing” on page 179.</p>	Write

14.2 Before You Begin

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

Review the following cautions before using the utility:

- ♦ The Association utility, like the driver, assumes that database identifiers are undelimited (unquoted and contain no special characters).
- ♦ Update all object associations related to a driver at the same time.
Updating associations at the same time is extremely important.
To see all of the objects associated with a particular driver, run the Association utility on the Identity Manager server associated with a particular driver instance.
The LDAP search base must contain all of the objects associated with a particular driver.
To ensure complete containment, we recommend that you use your tree’s root container as the search base.
- ♦ Make sure that the JDBC URL of the target database supplied to this utility is the same as the URL that the driver uses. Pointing this utility at a case-insensitive database when the database is actually case-sensitive might result in associations being normalized to the wrong case.
- ♦ Because the Association utility runs locally, it uses an unsecured connection. Therefore, the Identity Vault LDAP server must be temporarily configured to accept clear text passwords. Depending upon the third-party JDBC driver you are using, the database connection established by this utility might not be secure.
We recommend changing the driver’s authentication password on the database after you run this utility.

14.3 Using the Association Utility

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

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

Table 14-2 *Properties Files*

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

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

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

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

As a general rule, if the problem is isolated, manually edit each invalid association. If the problem is repetitive and affects a large number of associations, consider using operations 6 and 7. This utility can replace bad identifiers on a global basis, but cannot insert or remove them where they do not already exist. See [Section 14.4, “Parameters for Searching and Replacing,” on page 179](#) for information about search parameters.
- 4 Run the Association utility to normalize associations (operations 4 and 5).

14.4 Parameters for Searching and Replacing

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

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

Three types of search and replace operations are possible:

Table 14-3 *Search and Replace Operations*

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

- ♦ [Section 15.1, “Recognizing Publication Events,” on page 181](#)
- ♦ [Section 15.2, “Executing Test Scripts,” on page 181](#)
- ♦ [Section 15.3, “Troubleshooting Driver Processes,” on page 181](#)

15.1 Recognizing Publication Events

Publication events might not be recognized by the Publisher channel unless you explicitly commit changes. For the commit keywords of supported databases, see [Section 12.3.7, “Commit Keywords,” on page 148](#).

15.2 Executing Test Scripts

The test scripts should be executed by a user other than the driver’s `idm` database user account. If you execute them as the `idm` user, events are ignored by the driver’s Publisher channel, unless publication loopback is allowed. For additional information on allowing or disallowing publication loopback, refer to [“Allow Loopback?” on page 82](#).

15.3 Troubleshooting Driver Processes

Viewing driver processes is necessary to analyze unexpected behavior. To view the driver processing events, use DSTrace. You should only use it during testing and troubleshooting the driver. Running DSTrace while the drivers are in production increases the utilization on the Identity Manager server and can cause events to process very slowly. For more information, see [“Viewing Identity Manager Processes”](#) in the *Identity Manager 4.0 Common Driver Administration Guide*.

Uninstalling the Driver

A

- ♦ [Section A.1, “Deleting Identity Manager Driver Objects,” on page 183](#)
- ♦ [Section A.2, “Running the Product Uninstaller,” on page 183](#)
- ♦ [Section A.3, “Executing Database Uninstallation Scripts,” on page 183](#)

Novell recommends 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.

A.1 Deleting Identity Manager Driver Objects

When you are deleting Novell Identity Vault objects, you must delete all child objects before you can delete a parent object. For example, you must delete all rules and style sheets on the Publisher channel before you can delete the Publisher object. Similarly, you must delete both the Publisher and Subscriber objects before you can delete the Driver object.

To remove a driver object from an Identity Vault:

- 1 In Novell iManager, click *Identity Manager > Identity Manager Overview*.
- 2 Select a driver set.
- 3 On the Identity Manager Overview page, click *Delete Driver*.
- 4 Select the driver that you want to delete, then click *OK*.

A.2 Running the Product Uninstaller

Uninstallation procedures vary by platform.

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

A.3 Executing Database Uninstallation Scripts

This section provides helps you execute database uninstallation SQL scripts.

- ♦ [Section A.3.1, “IBM DB2 Universal Database \(UDB\) Uninstallation,” on page 184](#)
- ♦ [Section A.3.2, “Informix Dynamic Server \(IDS\) Uninstallation,” on page 184](#)
- ♦ [Section A.3.3, “Microsoft SQL Server Uninstallation,” on page 184](#)
- ♦ [Section A.3.4, “MySQL Uninstallation,” on page 185](#)
- ♦ [Section A.3.5, “Oracle Uninstallation,” on page 185](#)
- ♦ [Section A.3.6, “PostgreSQL Uninstallation,” on page 185](#)
- ♦ [Section A.3.7, “Sybase Adaptive Server Enterprise \(ASE\) Uninstallation,” on page 186](#)

A.3.1 IBM DB2 Universal Database (UDB) Uninstallation

The directory context for DB2 is *install-dir\DirXMLUtilities\jdbc\sql\db2_udbl\install*.

- 1 Drop the *idm*, *indirect*, and *direct* operating system user accounts.
- 2 If you haven't already done so, change the name of the administrator account name and password in the installation scripts.
- 3 Using the Command Line Processor (CLP), execute the *uninstall.sql* script.

For example: *db2 -f uninstall.sql*

This script won't execute in the Command Center interface beyond version 7 because the script uses the \ line continuation character. Later versions of the Command Center don't recognize this character.

- 4 Delete the *idm_db2.jar* file.

A.3.2 Informix Dynamic Server (IDS) Uninstallation

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

- 1 Drop the *idm* operating system user account.
- 2 Start a client such as SQL Editor.
- 3 Log on to your server as user *informix* or another user with DBA (database administrator) privileges.

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

If you execute scripts as a user other than *informix*, change all references to *informix* in the install scripts prior to execution.

- 4 If you aren't using the *informix* account with the default password, change the name of the DBA account name and password in the installation scripts.
- 5 Open and execute *uninstall.sql* from the *ansi* (transactional, ANSI-compliant), *log* (transactional, non-ANSI-compliant), or *no_log* (non-transactional, non-ANSI-compliant) subdirectory, depending upon which type of database you installed.

A.3.3 Microsoft SQL Server Uninstallation

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

- 1 Start a client such as Query Analyzer.
 - 2 Log on to your database server as user *sa*.
- By default, the *sa* user has no password.
- 3 Open and execute the first installation script, *uninstall.sql*.

The execute hotkey in Query Analyzer is F5.

A.3.4 MySQL Uninstallation

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

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

For example, from the command line execute `mysql -u root -p`

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

- 2 Execute the `uninstall.sql` uninstallation script.

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

Don't use a semicolon to terminate this statement.

A.3.5 Oracle Uninstallation

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

- 1 From an Oracle client, such as SQL Plus, log on as user `SYSTEM`.

By default, the password for `SYSTEM` is `MANAGER`.

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

- 2 Execute the uninstallation script `uninstall.sql`.

For example: `SQL> @c:\uninstall.sql`

A.3.6 PostgreSQL Uninstallation

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

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

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

By default, the Postgres user has no password.

- 2 From inside `psql`, execute the script `uninstall.sql`.

For example: `idm=# \i uninstall.sql`

- 3 Drop the database `idm`.

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

- 4 Remove or comment out entries for the `idm` user in the `pg_hba.conf` file.

For example:

```
#host      idm          idm          255.255.255.255  255.255.255.0
```

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

A.3.7 Sybase Adaptive Server Enterprise (ASE) Uninstallation

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

- 1 From a Sybase client, such as *isql*, log on as user *sa*.

- 2 Execute the installation script *uninstall.sql*.

For example, from the command line, execute *isql -U sa -P -i uninstall.sql*

By default, the *sa* account has no password.

Known Issues and Limitations

B

- ♦ [Section B.1, “Known Issues,” on page 187](#)
- ♦ [Section B.2, “Limitations,” on page 187](#)

B.1 Known Issues

- ♦ Identity Vault Time and Timestamp syntaxes are inadequate for expressing the range and granularity of their database counterparts. This is a publication problem because database time-related types typically have a wider range and greater degree of granularity (typically nanoseconds). The converse is not true. For more information, see [“Time Syntax” on page 49](#).
- ♦ The JDBC driver is unable to parse proprietary database time stamp formats. Some databases, such as Sybase and DB2, have proprietary time stamp formats that the `java.sql.Timestamp` (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html>) class can’t parse. When synchronizing time stamp columns from these databases, the JDBC driver, by default, assumes that time stamp values placed in the event log table are in ODBC canonical format (that is, `yyyy-mm-dd hh:mm:ss.ffffffffff`). The recommended method for enabling the JDBC driver to handle proprietary database time stamp formats is to implement a custom `DBTimestampTranslator` class. This interface is documented in the JavaDoc Tool that ships with the JDBC driver. Using this approach avoids the problem of reformatting time stamps in the database before they are inserted into the event log table or reformatted in style sheets. The JDBC driver ships with default implementations for the native DB2 time stamp format and the Sybase style 109 time stamp format.
- ♦ Statements executed against the database server might block indefinitely.

Typically, blocking is caused by a database resource being exclusively locked. Because the locking mechanisms and locking SQL vary by database, the general solution to this problem is to implement a custom `DBLockStatementGenerator` class. For additional information, see [“Lock Statement Generator Class” on page 64](#). The JDBC driver ships with a default implementation for Oracle.

Many factors can cause blocking. To mitigate the likelihood of blocking, we recommend that you do not set the [Transaction Isolation Level](#) parameter to a level greater than `read committed`.

The JDBC interface defines a method `java.sql.Statement.setQueryTimeout(int):void` (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html>) that allows a statement to time out after a specified number of seconds. Unfortunately, implementations of this method between third-party JDBC drivers range from not being implemented to having bugs. For this reason, this method was deemed unsuitable as a general-purpose solution.

B.2 Limitations

- ♦ The JDBC driver does not support the use of delimited (quoted) database identifiers (for example, “names with spaces”).
- ♦ JDBC 2 data types are not supported, with the exception of Large Object data types (LOBs) such as `CLOB` and `BLOB`.

- ♦ JDBC 3 data types are not supported.
- ♦ PostgreSQL does not support `<check-object-password>` events. Authentication is controlled by manually inserting entries into the `pg_hba.conf` file.

Best Practices

C

The following section lists important best practices for using the JDBC driver. You can find additional information in [Chapter 6, “Configuring the JDBC Driver,” on page 43](#).

Security/Performance:

- ♦ For performance and security reasons, run the driver remotely on the database server whenever possible. Be sure to enable SSL encryption between the Identity Vault and the Remote Loader service.
- ♦ You should enable SSL encryption for third-party drivers whenever the JDBC driver is not running remotely on the database server. For information on the security capabilities of supported third-party drivers, see [“Third-Party JDBC Drivers” on page 155](#).
- ♦ In a production environment, turn off tracing.

Other:

- ♦ For direct synchronization, prefix one or more view column names with “pk_” (case-insensitive).
- ♦ For both direct and indirect synchronization, use different primary key column names between logical database classes.
- ♦ Delimit (double-quote) primary key values placed in the event log `table_key` field if they contain the following characters: `, ; ' + = \ " < >` This caution is usually an issue only if the primary key column is a binary type.
- ♦ When an Identity Vault is the authoritative source of primary key values, GUID rather than CN is recommended for use as a primary key. Unlike CN, GUID is single-valued and does not change.
- ♦ From publication triggers, omit foreign key columns that link child and parent tables.
- ♦ If primary key columns are static (they do not change), do not include them in publication triggers.
- ♦ Place the `jdbc:type="query"` attribute value on all embedded `SELECT` statements. Place the `jdbc:type="update"` attribute value on all embedded `INSERT`, `UPDATE` and `DELETE` statements.
- ♦ To avoid issues that arise when you run a sql query that has a reserved word as a column name, specify a fully qualified name for the column.

For example, when you use *group* as a column name under *usr* table, *group* being is a sql keyword, your query might not be properly executed. To avoid this, specify a fully qualified name *such as* `usr.group (<Tablename>.<Columnname>)` for the column.

- ♦ [Section D.1, “Can’t See Tables or Views,” on page 191](#)
- ♦ [Section D.2, “Synchronizing with Tables,” on page 191](#)
- ♦ [Section D.3, “Processing Rows in the Event Log Table,” on page 191](#)
- ♦ [Section D.4, “Managing Database User Accounts,” on page 192](#)
- ♦ [Section D.5, “Synchronizing Large Data Types,” on page 192](#)
- ♦ [Section D.6, “Slow Publication,” on page 192](#)
- ♦ [Section D.7, “Synchronizing Multiple Classes,” on page 192](#)
- ♦ [Section D.8, “Encrypted Transport,” on page 193](#)
- ♦ [Section D.9, “Mapping Multivalued Attributes,” on page 193](#)
- ♦ [Section D.10, “Synchronizing Garbage Strings,” on page 193](#)
- ♦ [Section D.11, “Running Multiple JDBC Driver Instances,” on page 193](#)

D.1 Can’t See Tables or Views

Question: Why can’t the driver see my tables or views?

Answer: The driver is capable of synchronizing only tables that have explicit primary key constraints and views that contain one or more columns prefixed with “pk_” (case-insensitive). The driver uses these constraints to determine which fields to use when constructing associations. As such, the driver ignores any unconstrained tables. If you are trying to synchronize with tables or views that lack the necessary constraints, either add them or synchronize to intermediate tables with the required constraints.

Another possibility is that the driver lacks the necessary database privileges to see the tables. Usually, visibility is determined by the presence or absence of the `SELECT` privilege.

D.2 Synchronizing with Tables

Question: How do I synchronize with tables located in multiple schemas?

Answer: Do one of the following:

- ♦ Alias the tables into the synchronization schema.
 - ♦ Synchronize to intermediate tables in the synchronization schema and move the data across schema boundaries.
 - ♦ Use a view.
 - ♦ Create a virtual schema by using the Table/View Names parameter.
- See [“Table/View Names” on page 56](#).

D.3 Processing Rows in the Event Log Table

Question: Why isn’t the driver processing rows in the Event Log Table?

Answer: Do the following:

- 1 Check the `perpetrator` field of the rows in question and make sure that the value is set to something other than the driver's database username.
The Publisher channel checks the `perpetrator` field to detect loopback events if the Publisher channel Allow Loopback parameter is set to Boolean False (the default). See [“Allow Loopback?” on page 82](#).
When the Allow Loopback parameter is set to Boolean False, the Publisher channel ignores all records where the `perpetrator` field value is equal to the driver's database username. The driver's database username is specified by using the Authentication ID parameter. See [“Authentication ID” on page 46](#).
- 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.

D.4 Managing Database User Accounts

Question: Can the driver manage database user accounts?

Answer: Yes. You can manage database accounts by using embedded SQL. For more information, see [Chapter 11, “Embedded SQL Statements in XDS Events,” on page 119](#).

D.5 Synchronizing Large Data Types

Question: Can the driver synchronize large binary and string data types?

Answer: Yes. Large binary and string data types can be subscribed and published. Publish large binary and string data types by using query-back event types. For additional information, see [Section 10.2, “Event Types,” on page 110](#).

D.6 Slow Publication

Question: Why is publication slow?

Answer: If the event log table contains a large number of rows, index the table. Example indexes are provided in all database installation scripts. By using trace level 3, you can view the statements that the driver uses to maintain the event log.

You can further refine indexes in the installation scripts to enhance publication performance. Placing indexes in a different tablespace or physical disk than the event log table also enhances publication performance.

Furthermore, in a production environment, set the Delete Processed Rows parameter to Boolean False, unless processed rows are being periodically moved to another table. See [“Delete Processed Rows?” on page 81](#).

D.7 Synchronizing Multiple Classes

Question: Can the driver synchronize multiple classes?

Answer: Yes. However, primary key column names must be unique between logical database classes. For example, if *class1* is mapped to *table1* with primary key column name *key1*, and *class2* is mapped to *table2* with primary key column name *key2*, then the name of *key1* cannot equal *key2*.

This requirement can always be satisfied, no matter which synchronization model is employed.

D.8 Encrypted Transport

Question: Does the driver support encrypted transport?

Answer: No. How the driver communicates with a given database depends upon the third-party driver being used. Some third-party drivers support encrypted transport, but others do not. Even if encrypted transport is supported, no standardized way exists to enable encryption between third-party JDBC drivers.

The general solution for this problem is to remotely run the JDBC driver and your third-party driver. This method allows both the JDBC driver and the third-party driver to run locally on the database server. Then all data traveling across the network between the Metadirectory engine and the JDBC driver are SSL encrypted.

Another possibility is to use a type 3 or type 2 third-party JDBC driver. Database middleware and client APIs usually provide encrypted transport mechanisms.

D.9 Mapping Multivalue Attributes

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

Answer: See [Section 8.7, “Mapping Multivalue Attributes to Single-Value Database Fields,”](#) on page 103.

D.10 Synchronizing Garbage Strings

Question: Why is the driver synchronizing garbage strings?

Answer: The database and the third-party driver are probably using incompatible character encoding. Adjust the character encoding that your third-party driver uses.

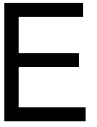
For more information, refer to [Character Encoding Values \(http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html\)](http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html), defined by Sun.

D.11 Running Multiple JDBC Driver Instances

Question: How do I run multiple JDBC driver instances in the same driver set? The instances require different versions of the same third-party JDBC driver (for example, the Oracle JDBC driver or the IBM DB2 Type 3 JDBC driver).

Answer: Use the Remote Loader to load each JDBC driver instance in a separate Java Virtual Machine (JVM). When run locally in the same JVM, different versions of the same third-party classes collide.

Supported Data Types



The JDBC driver can synchronize all JDBC 1 data types and a small subset of JDBC 2 data types. How JDBC data types map to a database's native data types depends on the third-party driver.

The following list includes the supported JDBC 1 [java.sql.Types](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) (<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) (<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`

java.sql.DatabaseMetaData Methods

F

This section lists the required and optional [java.sql.DatabaseMetaData](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html>) methods.

The following JDBC 1 methods are required only if the [Synchronization Filter](#) parameter is set to something other than *Exclude all tables/views*:

- ♦ `getColumns(java.lang.String catalog, java.lang.String schemaPattern, java.lang.String tableNamePattern, java.lang.String columnNamePattern):java.sql.ResultSet`
- ♦ `getPrimaryKeys(java.lang.String catalog, java.lang.String schema, java.lang.String table):java.sql.ResultSet`
- ♦ `getTables(java.lang.String catalog, java.lang.String schemaPattern, java.lang.String tableNamePattern, java.lang.String[] types):java.sql.ResultSet`
- ♦ `storesLowerCaseIdentifiers():boolean`
- ♦ `storesMixedCaseIdentifiers():boolean`
- ♦ `storesUpperCaseIdentifiers():boolean`

Optional JDBC 1 methods:

- ♦ `dataDefinitionCausesTransactionCommit():boolean`
- ♦ `dataDefinitionIgnoredInTransactions():boolean`
- ♦ `getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern):java.sql.ResultSet`
- ♦ `getDatabaseProductName():java.lang.String`
- ♦ `getDatabaseProductVersion():java.lang.String`
- ♦ `getDriverMajorVersion():int`
- ♦ `getDriverMinorVersion():int`
- ♦ `getDriverName():java.lang.String`
- ♦ `getDriverVersion():java.lang.String`
- ♦ `getExportedKeys(java.lang.String catalog, java.lang.String schema, java.lang.String table):java.sql.ResultSet`
- ♦ `getMaxStatements():int`
- ♦ `getMaxConnections():int`
- ♦ `getMaxColumnsInSelect():int`
- ♦ `getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern):java.sql.ResultSet`
- ♦ `getSchemas():java.sql.ResultSet`
- ♦ `getTableTypes():java.sql.ResultSet`
- ♦ `getUserName():java.lang.String`

- ♦ `supportsColumnAliasing():boolean`
- ♦ `supportsDataDefinitionAndDataManipulationTransactions():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`

JDBC Interface Methods

G

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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html>) methods) that the JDBC driver uses. Methods are organized by class.

Often, third-party JDBC driver vendors list defects or known issues by method. You can use the following methods in collaboration with third-party JDBC driver documentation to troubleshoot or anticipate potential interoperability problems.

- ♦ [java.sql.DriverManager](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html>)
- ♦ [java.sql.CallableStatement](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html>)
- ♦ [java.sql.Connection](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html>)
- ♦ [java.sql.PreparedStatement](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html>)
- ♦ [java.sql.ResultSet](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html>)
- ♦ [java.sql.ResultSetMetaData](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html>)
- ♦ [java.sql.Statement](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html>)
- ♦ [java.sql.Timestamp](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html) (<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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html>) methods that the JDBC driver uses:

Table G-1 *java.sql.DriverManager Methods*

Method Signature	JDBC Version	Required?
<code>getConnection(String url, java.util.Properties info):java.sql.Connection</code>	1	yes ¹
<code>getConnection(String url, java.util.Properties info):java.sql.Connection</code>	1	yes ¹
<code>setLogStream(java.io.PrintStream out):void</code>	1	no

¹Use 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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html>) methods that the JDBC driver uses:

Table G-2 *java.sql.CallableStatement Methods*

Method Signature	JDBC Version	Required?
<code>getBigDecimal(int parameterIndex, int scale):java.math.BigDecimal</code>	1	yes

Method Signature	JDBC Version	Required?
<code>getBoolean(int parameterIndex):boolean</code>	1	yes
<code>getBoolean(String parameterName):boolean</code>	3	no
<code>getBytes(int parameterIndex):byte</code>	1	yes
<code>getBytes(String parameterName):byte</code>	3	no
<code>getBytes(int parameterIndex):byte[]</code>	1	yes
<code>getBytes(String parameterName):byte[]</code>	3	no
<code>getDate(int parameterIndex):java.sql.Date</code>	1	yes
<code>getDate(String parameterName):java.sql.Date</code>	3	no
<code>getDouble(int parameterIndex):double</code>	1	yes
<code>getDouble(String parameterName):double</code>	3	no
<code>getFloat(int parameterIndex):float</code>	1	yes
<code>getFloat(String parameterName):float</code>	3	no
<code>getInt(int parameterIndex):int</code>	1	yes
<code>int getInt(String parameterName)</code>	3	no
<code>getLong(int parameterIndex):long</code>	1	yes
<code>getLong(String parameterName):long</code>	3	no
<code>getShort(int parameterIndex):short</code>	1	yes
<code>getShort(String parameterName):short</code>	3	no
<code>getString(int parameterIndex):String</code>	1	yes
<code>getString(String parameterName):String</code>	3	no
<code>getTime(int parameterIndex):java.sql.Time</code>	1	yes
<code>getTime(String parameterName):java.sql.Time</code>	3	no
<code>getTimestamp(int parameterIndex):java.sql.Timestamp</code>	1	yes
<code>getTimestamp(String parameterName):java.sql.Timestamp</code>	3	no
<code>registerOutParameter(int parameterIndex, int sqlType):void</code>	1	yes
<code>wasNull():boolean</code>	1	yes

The following table lists [java.sql.Connection](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html>) methods that the JDBC driver uses:

Table G-3 *java.sql.Connection Methods*

Method Signature	JDBC Version	Required?
close():void	1	yes
commit():void	1	no
createStatement():java.sql.Statement	1	yes
getAutoCommit():boolean	1	no
getMetaData():java.sql.DatabaseMetaData	1	yes
getTransactionIsolation():int	1	no
getWarnings():java.sql.SQLWarning	1	no
isClosed():boolean	1	no
prepareCall(String sql):java.sql.CallableStatement	1	no
prepareStatement(String sql):java.sql.PreparedStatement	1	yes
rollback():void	1	no
setAutoCommit(boolean autoCommit):void	1	no
setTransactionIsolation(int level):void	1	no

The following table lists [java.sql.PreparedStatement](http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html>) methods that the JDBC driver uses:

Table G-4 *java.sql.PreparedStatement Methods*

Method Signature	JDBC Version	Required?
clearParameters() :void	1	no
execute():boolean	1	yes
executeQuery():java.sql.ResultSet	1	yes
executeUpdate():int	1	yes
setBigDecimal(int parameterIndex, java.math.BigDecimal x):void	1	yes
setBoolean(int parameterIndex, boolean x):void	1	yes
setByte(int parameterIndex, byte x):void	1	yes
setBytes(int parameterIndex, byte x[]):void	1	yes
setDate(int parameterIndex, java.sql.Date x):void	1	yes
setDouble(int parameterIndex, double x):void	1	yes
setFloat(int parameterIndex, float x):void	1	yes
setInt(int parameterIndex, int x):void	1	yes
setLong(int parameterIndex, long x):void	1	yes

Method Signature	JDBC Version	Required?
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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html>) methods that the JDBC driver uses:

Table G-5 *java.sql.ResultSet Methods*

Method Signature	JDBC Version	Required?
close():void	1	yes
getBigDecimal(int columnIndex, int scale):java.math.BigDecimal	1	yes
getBigDecimal(String columnName, int scale):java.math.BigDecimal	1	yes
getBinaryStream(int columnIndex):java.io.InputStream	1	yes
getBinaryStream(String columnName):java.io.InputStream	1	yes
getBoolean(int columnIndex):boolean	1	yes
getBoolean(String columnName):boolean	1	yes
getByte(int columnIndex):byte	1	yes
getByte(String columnName):byte	1	yes
getBytes(int columnIndex):byte[]	1	yes
getBytes(String columnName):byte[]	1	yes
getDate(int columnIndex):java.sql.Date	1	yes
getDate(String columnName):java.sql.Date	1	yes
getFloat(int columnIndex):float	1	yes
getFloat(String columnName):float	1	yes
getInt(int columnIndex):int	1	yes
getInt(String columnName):int	1	yes
getLong(int columnIndex):long	1	yes
getLong(String columnName):long	1	yes
getMetaData():java.sql.ResultSetMetaData	1	no
getShort(int columnIndex):short	1	yes
getShort(String columnName):short	1	yes

Method Signature	JDBC Version	Required?
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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html>) methods that the JDBC driver uses:

Table G-6 *java.sql.ResultSetMetaData Methods*

Method Signature	JDBC Version	Required?
getColumnCount():int	1	yes
columnName(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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html>) methods that the JDBC driver uses:

Table G-7 *java.sql.Statement Methods*

Method Signature	JDBC Version	Required?
addBatch(java.lang.String sql):void	2	no
clearBatch():void	2	no
clearWarnings():void	1	no
close():void	1	yes
execute(java.lang.String sql):boolean	1	yes
executeBatch():int[]	2	no
executeUpdate(String sql):int	1	yes
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) (<http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html>) methods that the JDBC driver uses:

Table G-8 *java.sql.Timestamp Methods*

Method Signature	JDBC Version	Required?
getNanos():int	1	yes
getTime():long	1	yes
setNanos(int n):void	1	yes
setTime(long time):void	1	yes
toString ():String	1	yes

Third-Party JDBC Driver Descriptor DTD



This section contains the DTD for third-party JDBC descriptor files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--ELEMENT actions (exec-sql | check-for-closed-connection | fetch
metadata | rollback)*-->
<!--ELEMENT add-default-values-on-view-insert (#PCDATA)-->
<!--ELEMENT authentication (regular-expression | sql-state | error-code
| sql-state-class | error-code-range | actions)*-->
<!--ELEMENT check-for-closed-connection EMPTY-->
<!--ELEMENT column-position-comparator (#PCDATA)-->
<!--ELEMENT connection-properties (property)*-->
<!--ELEMENT connectivity (regular-expression | sql-state | error-code |
sql-state-class | error-code-range | actions)*-->
<!--ELEMENT current-timestamp-stmt (#PCDATA)-->
<!--ELEMENT error-code (value)-->
<!--ATTLIST error-code
    description CDATA #IMPLIED
-->
<!--ELEMENT error-code-range (from, to)-->
<!--ATTLIST error-code-range
    description CDATA #IMPLIED
-->
<!--ELEMENT errors (connectivity | authentication | retry | fatal)*-->
<!--ELEMENT exclude-table-filter (#PCDATA)-->
<!--ELEMENT exec-sql (#PCDATA)-->
<!--ELEMENT fatal (regular-expression | sql-state | error-code | sql
state-class | error-code-range | actions)*-->
<!--ELEMENT fetch-metadata EMPTY-->
<!--ELEMENT from (#PCDATA)-->
<!--ELEMENT function-return-method (#PCDATA)-->
<!--ELEMENT handle-stmt-results (#PCDATA)-->
<!--ELEMENT identity (name?, target-database?, jdbc-type?, jdbc-class?)-->
<!--ELEMENT import (#PCDATA)-->
<!--ELEMENT imports (import)*-->
<!--ELEMENT include-table-filter (#PCDATA)-->
<!--ELEMENT jdbc-class (#PCDATA)-->
<!--ELEMENT jdbc-driver (imports?, identity, (metadata-override |
connection-properties | sql-type-map | options | errors)*)-->
<!--ELEMENT jdbc-type (#PCDATA)-->
<!--ELEMENT key (#PCDATA)-->
<!--ELEMENT left-outer-join-operator (#PCDATA)-->
<!--ELEMENT lock-generator-class (#PCDATA)-->
<!--ELEMENT metadata-override (supports-schemas-in-procedure-calls?)-->
<!--ELEMENT minimal-metadata (#PCDATA)-->
<!--ELEMENT name (#PCDATA)-->
<!--ELEMENT options (lock-generator-class | supports-schemas-in
metadata-retrieval | time-translator-class | column-position
comparator | use-manual-transactions | minimal-metadata | transaction
isolation-level | use-single-connection | exclude-table-filter |
include-table-filter | left-outer-join-operator | current-timestamp
stmt | add-default-values-on-view-insert | reuse-statements |
```

```

function-return-method | handle-stmt-results)*>
<!ELEMENT property (key, value)>
<!ELEMENT regular-expression (value)>
<!ELEMENT retry (regular-expression | sql-state | error-code | sql
state-class | error-code-range | actions)*>
<!ELEMENT reuse-statements (#PCDATA)>
<!ELEMENT rollback EMPTY>
<!ELEMENT sql-state (value)>
<!ATTLIST sql-state
  description CDATA #IMPLIED
>
<!ELEMENT sql-state-class (value)>
<!ATTLIST sql-state-class
  description CDATA #IMPLIED
>
<!ELEMENT sql-type-map (type*)>
<!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT supports-schemas-in-procedure-calls (#PCDATA)>
<!ELEMENT target-database (#PCDATA)>
<!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT type (from, to)>
<!ELEMENT use-manual-transactions (#PCDATA)>
<!ELEMENT use-single-connection (#PCDATA)>
<!ELEMENT value (#PCDATA)>

```

Third-Party JDBC Driver Descriptor Import DTD

This section contains the DTD for third-party JDBC descriptor import files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT actions (exec-sql | check-for-closed-connection | fetch-metadata |
rollback)*>
<!ELEMENT add-default-values-on-view-insert (#PCDATA)>
<!ELEMENT authentication (regular-expression | sql-state | error-code | sql-
state-class | error-code-range | actions)*>
<!ELEMENT check-for-closed-connection EMPTY>
<!ELEMENT column-position-comparator (#PCDATA)>
<!ELEMENT connection-properties (property)*>
<!ELEMENT connectivity (regular-expression | sql-state | error-code | sql-
state-class | error-code-range | actions)*>
<!ELEMENT current-timestamp-stmt (#PCDATA)>
<!ELEMENT error-code (value)>
<!ATTLIST error-code
    description CDATA #IMPLIED
>
<!ELEMENT error-code-range (from, to)>
<!ATTLIST error-code-range
    description CDATA #IMPLIED
>
<!ELEMENT errors (connectivity | authentication | retry | fatal)*>
<!ELEMENT exclude-table-filter (#PCDATA)>
<!ELEMENT exec-sql (#PCDATA)>
<!ELEMENT fatal (regular-expression | sql-state | error-code | sql-state-class
| error-code-range | actions)*>
<!ELEMENT fetch-metadata EMPTY>
<!ELEMENT from (#PCDATA)>
<!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)>
<!ELEMENT include-table-filter (#PCDATA)>
<!ELEMENT jdbc-driver (metadata-override | connection-properties | sql-type-
map | options | errors)*>
<!ELEMENT key (#PCDATA)>
<!ELEMENT left-outer-join-operator (#PCDATA)>
<!ELEMENT lock-generator-class (#PCDATA)>
<!ELEMENT metadata-override (supports-schemas-in-procedure-calls?)>
<!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT options (lock-generator-class | supports-schemas-in-metadata-
retrieval | time-translator-class | column-position-comparator | use-manual-
transactions | minimal-metadata | transaction-isolation-level | use-single-
connection | exclude-table-filter | include-table-filter | left-outer-join-
operator | current-timestamp-stmt | add-default-values-on-view-insert | reuse-
statements | function-return-method | handle-stmt-results)*>
<!ELEMENT property (key, value)>
<!ELEMENT regular-expression (value)>
<!ELEMENT retry (regular-expression | sql-state | error-code | sql-state-class
| error-code-range | actions)*>
<!ELEMENT reuse-statements (#PCDATA)>
<!ELEMENT rollback EMPTY>
```

```

<!ELEMENT sql-state (value)>
<!ATTLIST sql-state
    description CDATA #IMPLIED
>
<!ELEMENT sql-state-class (value)>
<!ATTLIST sql-state-class
    description CDATA #IMPLIED
>
<!ELEMENT sql-type-map (type*)>
<!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT supports-schemas-in-procedure-calls (#PCDATA)>
<!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT type (from, to)>
<!ELEMENT use-manual-transactions (#PCDATA)>
<!ELEMENT use-single-connection (#PCDATA)>
<!ELEMENT value (#PCDATA)>

```


Database Descriptor DTD

J

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)>
```


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)>
```


Policy Example: Triggerless Future Event Processing



The following example assumes that a “commence” attribute exists and does the following:

- Holds the time stamp value indicating when an event should be processed
- Contains an integer or Java string time stamp value. See [“Time Syntax” on page 49](#).

```
<policy xmlns:Timestamp="http://www.novell.com/nxsl/java/
java.sql.Timestamp"
  xmlns:TimestampUtil="http://www.novell.com/nxsl/java/
com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil"
  xmlns:jdbc="urn:dirxml:jdbc">
<rule>
<description>Get commencement date from datasource.</description>
  <conditions>
    <and>
      <if-xpath op="true">.</if-xpath>
    </and>
  </conditions>
  <actions>
    <do-set-local-variable name="commence">
      <arg-string>
        <token-src-attr class-name="User" name="commence"/>
      </arg-string>
    </do-set-local-variable>
  </actions>
</rule>

<rule>
  <description>Break if commencement date unavailable.</description>
  <conditions>
    <and>
      <if-local-variable name="commence" op="equal"/>
    </and>
  </conditions>
  <actions>
    <do-break/>
  </actions>
</rule>

<rule>
<description>Parse times.</description>
  <conditions>
    <and>
      <if-xpath op="true">.</if-xpath>
    </and>
  </conditions>
  <actions>
    <do-set-local-variable name="dbTime">
      <arg-object>
        <token-xpath expression="Timestamp:valueOf(@jdbc:database-local-
time)"/>
      </arg-object>
    </do-set-local-variable>
  </actions>
</rule>
```

```

        </do-set-local-variable>
        <do-set-local-variable name="eventTime">
            <arg-object>
                <token-xpath expression="Timestamp:valueOf($commence)"/>
            </arg-object>
        </do-set-local-variable>
    </actions>
</rule>
<rule>
    <description>Is commencement date after database time?</description>
    <conditions>
        <and>
            <if-xpath op="true">.</if-xpath>
        </and>
    </conditions>
    <actions>
        <do-set-local-variable name="after">
            <arg-string>
                <token-xpath expression="TimestampUtil:after($eventTime, $dbTime)"/>
            </arg-string>
        </do-set-local-variable>
    </actions>
</rule>

<rule>
<description>Retry if future event.</description>
    <conditions>
        <and>
            <if-local-variable name="after" op="equal">true</if-local-variable>
        </and>
    </conditions>
    <actions>
        <do-status level="retry">
            <arg-string>
                <token-text xml:space="preserve">Future event detected.</token-text>
            </arg-string>
        </do-status>
    </actions>
</rule>
</policy>

```

Setting Up an OCI Client on Linux

M

- ♦ Section M.1, “Downloading the Instant Client,” on page 215
- ♦ Section M.2, “Setting Up the OCI Client,” on page 215
- ♦ Section M.3, “Configuring the OCI Driver,” on page 216

M.1 Downloading the Instant Client

- 1 Download the Oracle Instant Client (`instantclient-basic-linux32-11.1.0.7.zip`).

The file is available from [Instant Client Downloads \(http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html\)](http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html).

- 2 Download the Oracle SQL Plus binary (`instantclient-sqlplus-linux32-11.1.0.7.zip`).

The file is available from [Instant Client Downloads \(http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html\)](http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html).

M.2 Setting Up the OCI Client

Set up the Oracle Instant Client on the machine where the JDBC driver is running (not on the machine where Oracle is running).

- 1 Log into Linux as root, and create the following structure:

```
/oracle /oracle/client /oracle/client/bin /oracle/client/lib /oracle/client/
network/admin
```

- 2 Unzip all files from `instantclient-basic-linux32-11.1.0.7.zip` to `/oracle/client/lib`.

- 3 Unzip all files from `instantclient-sqlplus-linux32-11.1.0.7.zip` to `/oracle/client/bin`.

- 4 Copy `libsqlplus.so` from `/oracle/client/bin` to `/oracle/client/lib`.

- 5 Copy `libsqlplusic.so` from `/oracle/client/bin` to `/oracle/client/lib`.

- 6 Using `chmod`, ensure that the file `sqlplus` in `/oracle/client/bin` is executable.

- 7 Copy a valid `tnsnames.ora` into `/oracle/client/network/admin`.

If you don't have a `tnsnames.ora` file, use the Oracle configuration tool to create one.

Make sure that the `tnsnames.ora` filename is in lowercase.

- 8 Modify the `profile.local` file by adding the following lines:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/oracle/client/lib
export TNS_ADMIN=/oracle/client/network/admin
export PATH=$PATH:/oracle/client/lib
```

The `profile.local` file is in the `/etc` folder. If the file doesn't exist, create one. The file can consist of only the three export lines.

The `profile.local` file extends the `LD_LIBRARY_PATH`, sets `TNS_ADMIN`, and extends the `PATH`. This file is read when the server boots.

- 9 Ensure that the exports in the `profile.local` file are always valid.
- 10 Copy the `classes12.jar` and `ojdbc14.jar` and `ojdbc5.jar` or `ojdbc6.jar` to the Identity Manager classes directory.

These `.jar` files are supplied with the Instant Client.

The Identity Manager classes directory is the directory where your driver is located.
- 11 Start SQL Plus with the following example command (assuming that the directory is `/oracle/client/bin`):

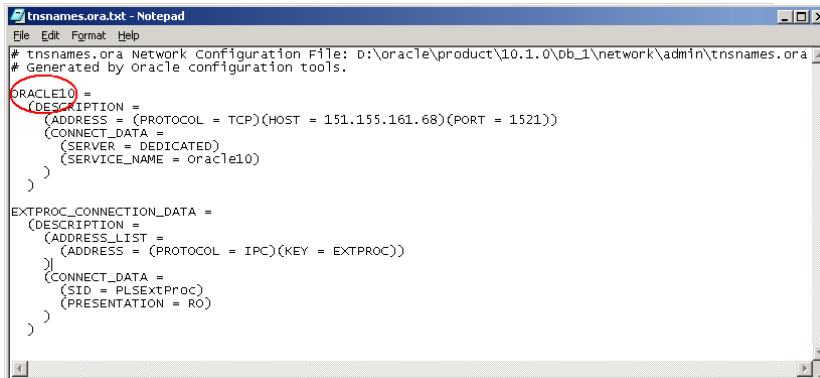
`./sqlplus username/password@sid`

M.3 Configuring the OCI Driver

To configure the driver, customize the driver's URL syntax. See [Table 13-18 on page 166](#).

An example URL syntax is `jdbc:oracle:oci8:@ORACLE10`. In this example, *ORACLE10* is the connection string in the `tnsnames.ora` file.

Figure M-1 Example `tnsnames.ora` File



Sybase Chain Modes and the Identity Manager JDBC driver

N

Sybase can execute stored procedures in two distinct modes: chained and unchained. Depending upon the configuration of the Identity Manager JDBC driver and stored procedures in a database, various problems can arise. This section can help you understand and resolve those problems.

- ♦ [Section N.1, “Error Codes,” on page 217](#)
- ♦ [Section N.2, “Procedures and Modes,” on page 218](#)

N.1 Error Codes

- ♦ [“Error 226: SET CHAINED command not allowed within multi-statement transaction” on page 217](#)
- ♦ [“Error 7112: Stored procedure 'x' may be run only in chained transaction mode” on page 217](#)
- ♦ [“Error 7113: Stored procedure 'x' may be run only in unchained transaction mode” on page 217](#)

Error 226: SET CHAINED command not allowed within multi-statement transaction

Effect: Throws the exception of `com.sybase.jdbc2.jdbc.SybSQLException` with error code 226 and an SQL state of `ZZZZZ`.

Cause: This exception is usually caused by a defect in older versions of `jConnect`.

Solution: Download and upgrade to the latest version. Downloads are available at the [jConnect for JDBC Web page \(http://www.sybase.com/products/informationmanagement/softwaredeveloperkit/jconnect\)](http://www.sybase.com/products/informationmanagement/softwaredeveloperkit/jconnect).

Error 7112: Stored procedure 'x' may be run only in chained transaction mode

Effect: Throws the exception of `com.sybase.jdbc2.jdbc.SybSQLException` with error code 7112 and an SQL state of `ZZZZZ`.

Cause: The stored procedure was created in chained mode, or later altered to run in chained mode, but the driver is currently running in unchained mode. The probable cause is that the [Use Manual Transactions?](#) parameter is set to *False*. Another possibility is that the transaction type has been overridden to *auto* in a policy.

Solution: Do one of the following:

- ♦ Use stored procedure `sp_procxmode` to change the stored procedure's mode to *unchained* or *anymode* (preferred).
- ♦ Change the driver's [Use Manual Transactions?](#) parameter to *True*, or change the policy transaction type to *manual*.

Error 7113: Stored procedure 'x' may be run only in unchained transaction mode

Effect: Throws the exception `com.sybase.jdbc2.jdbc.SybSQLException` with error code 7113 and an SQL state of `ZZZZZ`.

Cause: The stored procedure was created in unchained mode, or later altered to run in unchained mode, but the driver is currently running in chained mode. The probable cause is that the [Use Manual Transactions?](#) parameter is set to *True*. Another possibility is that the transaction type has been overridden to *manual* in policy.

Solution: Do one of the following:

- ♦ Use stored procedure `sp_procxmode` to change the stored procedure's mode to *chained* or *anymode* (preferred).
- ♦ Change the driver's [Use Manual Transactions?](#) parameter to *False*, or change the policy transaction type to *auto*.

If you set `use-manual-transactions` to *False*, all transactions consist of a maximum of one statement.

N.2 Procedures and Modes

- ♦ [Section N.2.1, “Using Stored Procedure `sp_procxmode`,” on page 218](#)
- ♦ [Section N.2.2, “Chained and Unchained Modes,” on page 218](#)
- ♦ [Section N.2.3, “Managing Transactions in a Policy,” on page 219](#)
- ♦ [Section N.2.4, “Useful Links,” on page 219](#)

N.2.1 Using Stored Procedure `sp_procxmode`

The preferred way to avoid errors 7112 and 7113 is to alter all stored procedures invoked directly or indirectly by the driver (via triggers, for example) to run in both chained and unchained mode. To alter a procedure, invoke the `sp_procxmode` procedure with two arguments:

- ♦ The procedure name
- ♦ The mode

The following example illustrates how to invoke the `sp_procxmode` procedure from the `isql` command line:

```
client:sp_procxmode my_procedure, anymode go
```

Of course, not all customers are willing to alter stored procedure modes. Altering a procedure's mode might alter its runtime behavior, which could alter the behavior of other applications that invoke the procedure.

N.2.2 Chained and Unchained Modes

Unchained mode is Sybase's native way of executing SQL. A second mode, chained mode, was later added to make the database compatible with SQL standards.

Table N-1 *Modes and Compatibility*

Mode	Compatibility
Chained	SQL-compatible mode

Mode	Compatibility
Unchained	Sybase native mode

Sybase provides a third-party JDBC driver called jConnect. The default mode of jConnect is unchained. Whenever the method `Connection.setAutoCommit(boolean autoCommit):void` is invoked, jConnect switches modes. See [java.sql Interface Connection \(http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html\)](http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html).

Table N-2 *Methods and Switches*

Method	Effect
<code>Connection.setAutoCommit(true)</code>	Switches to unchained mode
<code>Connection.setAutoCommit(false)</code>	Switches to chained mode

If the [Use Manual Transactions?](#) parameter is set to *False*, the driver invokes `Connection.setAutoCommit(true)`. That is, the driver enters unchained mode. This is the normal processing mode for SELECT statements and SQL embedded in a policy where the transaction type is set to `auto`. See [Section 11.5, “Manual vs. Automatic Transactions,” on page 125](#). When the driver is in this state, any chained stored procedures invoked directly or indirectly by the driver yield the 7112 error.

If the [Use Manual Transactions?](#) parameter is set to *True*, the driver invokes `Connection.setAutoCommit(false)`. That is, the driver enters chained mode. This is the normal processing mode for all statements except SELECT statements and SQL embedded in a policy where the transaction type is set to `manual`. See [Section 11.5, “Manual vs. Automatic Transactions,” on page 125](#). When the driver is in this state, any unchained stored procedures invoked directly or indirectly by the driver yield the 7113 error.

N.2.3 Managing Transactions in a Policy

For information on managing transactions in a policy, see [Section 11.5, “Manual vs. Automatic Transactions,” on page 125](#)

N.2.4 Useful Links

- ♦ [Transaction modes and stored procedures \(http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/55096;hf=0;pt=55096#X\)](http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/55096;hf=0;pt=55096#X) in the *Transact-SQL User's Guide*
- ♦ [Selecting the transaction mode and isolation level \(http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/53713;pt=53001\)](http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/53713;pt=53001) in the *Transact-SQL User's Guide*

