

# **IBM Informix JDBC Driver**

## **Programmer's Guide**

UNIX and Windows

Informix Dynamic Server, Version 7.x

Informix Dynamic Server, Workgroup and Developer Editions, Version 7.x

Informix Dynamic Server with Advanced Decision Support and Extended Parallel Options, Version 8.x

Informix Dynamic Server with Universal Data Option, Version 9.x

INFORMIX-OnLine Dynamic Server, Version 5.x

INFORMIX-SE, Version 5.x

INFORMIX-SE, Version 7.2x

Version 1.4

August 2001

Part No. 000-5343A

© Copyright International Business Machines Corporation 2001. All rights reserved.

#### Trademarks

AIX; DB2; DB2 Universal Database; Distributed Relational Database Architecture; NUMA-Q; OS/2, OS/390, and OS/400; IBM Informix<sup>®</sup>; C-ISAM<sup>®</sup>; Foundation.2000<sup>™</sup>; IBM Informix<sup>®</sup> 4GL; IBM Informix<sup>®</sup> DataBlade<sup>®</sup> Module; Client SDK<sup>™</sup>; Cloudscape<sup>™</sup>; Cloudsync<sup>™</sup>; IBM Informix<sup>®</sup> Connect; IBM Informix<sup>®</sup> Driver for JDBC; Dynamic Connect<sup>™</sup>; IBM Informix<sup>®</sup> Dynamic Scalable Architecture<sup>™</sup> (DSA); IBM Informix<sup>®</sup> Dynamic Server<sup>™</sup>; IBM Informix<sup>®</sup> Enterprise Gateway Manager (Enterprise Gateway Manager); IBM Informix<sup>®</sup> Extended Parallel Server<sup>™</sup>; i. Financial Services<sup>™</sup>; J/Foundation<sup>™</sup>; MaxConnect<sup>™</sup>; Object Translator<sup>™</sup>; Red Brick Decision Server<sup>™</sup>; IBM Informix<sup>®</sup> SE; IBM Informix<sup>®</sup> SQL; InformiXML<sup>™</sup>; RedBack<sup>®</sup>; SystemBuilder<sup>™</sup>; U2<sup>™</sup>; UniData<sup>®</sup>; UniVerse<sup>®</sup>; wintegrate<sup>®</sup> are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

Documentation Team: June Smith, Juliet Shackell, Oakland Editing & Production

---

# Table of Contents

## Introduction

In This Introduction . . . . .	3
About This Manual . . . . .	3
Organization of This Manual . . . . .	3
Material Not Covered . . . . .	4
Types of Users . . . . .	4
Software Dependencies . . . . .	4
Assumptions About Your Locale. . . . .	5
Documentation Conventions . . . . .	5
Typographical Conventions . . . . .	6
Icon Conventions . . . . .	7
Additional Documentation . . . . .	8
Printed Documentation . . . . .	8
On-Line Documentation . . . . .	9
Vendor-Specific Documentation . . . . .	9
Compliance with Industry Standards . . . . .	10
Informix Welcomes Your Comments . . . . .	10

## Chapter 1

### Getting Started

In This Chapter . . . . .	1-3
What Is JDBC? . . . . .	1-3
What Is a JDBC Driver? . . . . .	1-5
Overview of Informix JDBC Driver . . . . .	1-6
Installing the Driver . . . . .	1-7
Interactive Installation . . . . .	1-7
Silent Installation . . . . .	1-10
Uninstalling the Driver . . . . .	1-12
Using the Driver in an Application . . . . .	1-13
Using the Driver in an Applet . . . . .	1-14

## Chapter 2

## Programming with Informix JDBC Driver

In This Chapter . . . . .	2-3
Establishing a Connection . . . . .	2-3
Loading Informix JDBC Driver . . . . .	2-4
Creating a Connection . . . . .	2-4
Accessing Database Metadata . . . . .	2-14
Querying the Database . . . . .	2-15
Manipulating Informix Large Object Data Types . . . . .	2-15
Manipulating Informix INTERVAL Data Types . . . . .	2-21
Manipulating Other Data Types. . . . .	2-23
Informix-Specific Information About Querying a Database . . . . .	2-23
Example of Sending a Query to an Informix Database . . . . .	2-25
Escape Syntax . . . . .	2-26
Mapping Data Types . . . . .	2-26
Mapping Between Informix and JDBC Data Types . . . . .	2-27
Supported ResultSet.getXXX() Methods . . . . .	2-29
Handling Errors . . . . .	2-31
Using the SQLException Class . . . . .	2-31
Retrieving Informix Error Message Text . . . . .	2-32
Internationalization . . . . .	2-33
JDK 1.1 and 1.2 Internationalization Support . . . . .	2-33
Support for Informix GLS Variables . . . . .	2-33
Support for End-User Formats . . . . .	2-34
Precedence Rules Regarding DATE Value End-User Formats . . . . .	2-41
Support for Code Set Conversion . . . . .	2-42
Handling Transactions . . . . .	2-48
Other Informix Extensions to the JDBC API . . . . .	2-49
Using the Informix SERIAL and SERIAL8 Data Types . . . . .	2-49
Obtaining Driver Version Information . . . . .	2-50
Using an HTTP Proxy Server . . . . .	2-51
Restrictions and Limitations . . . . .	2-53

<b>Chapter 3</b>	<b>Troubleshooting</b>	
	In This Chapter . . . . .	3-3
	Debugging Your JDBC API Program . . . . .	3-3
	Using the Debug Version of the Driver . . . . .	3-3
	Turning on Tracing . . . . .	3-4
	Performance Issues . . . . .	3-5
	Using the FET_BUF_SIZE Environment Variable . . . . .	3-6
	Memory Management of Large Objects . . . . .	3-6
	Reducing Network Traffic . . . . .	3-8
<b>Appendix A</b>	<b>Sample Code Files</b>	
	<b>Glossary</b>	
	<b>Error Messages</b>	
	<b>Index</b>	



---

# Introduction

In This Introduction . . . . .	3
About This Manual. . . . .	3
Organization of This Manual . . . . .	3
Material Not Covered . . . . .	4
Types of Users . . . . .	4
Software Dependencies . . . . .	4
Assumptions About Your Locale. . . . .	5
Documentation Conventions . . . . .	5
Typographical Conventions . . . . .	6
Icon Conventions . . . . .	7
Comment Icons . . . . .	7
Platform Icons . . . . .	7
Additional Documentation . . . . .	8
Printed Documentation . . . . .	8
On-Line Documentation. . . . .	9
Vendor-Specific Documentation . . . . .	9
Compliance with Industry Standards . . . . .	10
Informix Welcomes Your Comments . . . . .	10





## In This Introduction

This Introduction provides an overview of the information in this manual and describes the conventions it uses.

---

## About This Manual

This guide describes how to install, load, and use Informix JDBC Driver to connect to an Informix database from within a Java application or applet.

This section discusses the organization of the manual, the intended audience, and the associated software products you must have to use Informix JDBC Driver.

## Organization of This Manual

This manual includes the following chapters:

- Chapter 1, “Getting Started,” describes Informix JDBC Driver and the JDBC application programming interface (API) in general. It provides essential information for programmers to immediately start using the product, such as instructions on how to install and load the driver.
- Chapter 2, “Programming with Informix JDBC Driver,” explains in more detail the Informix-specific information needed to use Informix JDBC Driver to connect to an Informix database. This information includes how to create a connection to an Informix database, query tables, and handle errors.

This chapter also explains the Informix-specific data types supported in Informix JDBC Driver. This information includes how to map data types.

- Chapter 3, “Troubleshooting,” provides troubleshooting tips to solve programming errors and problems with the driver. It also describes browser security issues when you use Informix JDBC Driver in a Java applet.
- Appendix A, “Sample Code Files,” lists examples referred to in the guide.
- A glossary of relevant terms and a list of error messages follow the chapters, and an index directs you to areas of particular interest.

## Material Not Covered

This guide does not describe all the interfaces, classes, and methods of the JDBC API and does not provide detailed descriptions of how to use the JDBC API to write Java applications that connect to Informix databases. The examples in the guide provide enough information to show how to use Informix JDBC Driver but do not provide an extensive description of the JDBC API.

For more information about the JDBC API, visit the JavaSoft Web site at:

<http://www.javasoft.com/products/jdk/1.1/docs/guide/jdbc/index.html>

## Types of Users

This guide is for Java programmers who use the JDBC API to connect to Informix databases using Informix JDBC Driver. To use this guide, you should know how to program in Java and, in particular, understand the classes and methods of the JDBC API.

## Software Dependencies

To use Informix JDBC Driver to connect to an Informix database, you must use one of the following Informix database servers:

- Informix Dynamic Server, Version 7.x
- Informix Dynamic Server, Workgroup and Developer Editions, Version 7.x

- Informix Dynamic Server with Advanced Decision Support and Extended Parallel Options, Version 8.x
- Informix Dynamic Server with Universal Data Option, Version 9.x
- INFORMIX-OnLine Dynamic Server, Version 5.x
- INFORMIX-SE, Version 5.x
- INFORMIX-SE, Version 7.2x

You must also use Java Development Kit (JDK), Version 1.1.5 or later.

## Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a GLS (Global Language Support) locale.

The examples in this manual are written with the assumption that you are using the default locale, **en\_us.8859-1**. This locale supports U.S. English format conventions for dates, times, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

---

## Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set:

- Typographical conventions
- Icon conventions

## Typographical Conventions

This manual uses the following conventions to introduce new terms, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i> <i>italics</i> <i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
<b>boldface</b> <b>boldface</b>	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
monospace <i>monospace</i>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of product- or platform-specific information.
→	This symbol indicates a menu item. For example, “Choose <b>Tools→Options</b> ” means choose the <b>Options</b> item from the <b>Tools</b> menu.



**Tip:** When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.






**Tip:** The text and many of the examples in this manual show routine and data type names in mixed lettercasing (uppercase and lowercase). Because Informix Dynamic Server is case insensitive, you do not need to enter routine names exactly as shown: you can use uppercase letters, lowercase letters, or any combination of the two.

## Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.


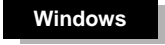
### *Comment Icons*

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
	<i>Warning:</i>	Identifies paragraphs that contain vital instructions, cautions, or critical information
	<i>Important:</i>	Identifies paragraphs that contain significant information about the feature or operation that is being described
	<i>Tip:</i>	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

### *Platform Icons*

Platform icons identify paragraphs that contain platform-specific information.

Icon	Description
	Identifies information that is specific to the UNIX environment.
	Identifies information that is specific to the Windows environment.

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the platform-specific information.

---

## Additional Documentation

This section describes the following parts of the documentation set:

- Printed documentation
- On-line documentation
- Vendor-specific documentation

### Printed Documentation

The following related Informix documents complement the information in this manual:

- If you have never used Structured Query Language (SQL), read the *Informix Guide to SQL: Tutorial*. It provides a tutorial on SQL as it is implemented by Informix products. It also describes the fundamental ideas and terminology for planning and implementing a relational database.
- A companion volume to the *Tutorial*, the *Informix Guide to SQL: Reference*, includes details of the Informix system catalog tables, describes Informix and common environment variables that you should set, and describes the column data types that Informix database servers support.
- The *Informix Guide to SQL: Syntax* provides information about SQL syntax as it is implemented by Informix products.
- *Informix Error Messages* is useful if you do not want to look up your error messages on-line.

## On-Line Documentation

The Informix Answers OnLine CD allows you to print chapters or entire books and perform full-text searches for information in specific books or throughout the documentation set. You can install the documentation or access it directly from the CD. For information about how to install, read, and print on-line manuals, see the installation insert that accompanies Answers OnLine. You can also obtain the same information on the Web at <http://www.informix.com/answers>.

In addition to the Informix set of manuals, the following on-line files supplement the information in this manual.

On-Line File	Purpose
JDBCREL.TXT	The release notes describe any special actions required to configure and use Informix JDBC Driver on your computer. Additionally, this file contains information about any known problems and their workarounds.
JBCDOC.TXT	The documentation notes describe features not covered in the manuals or modified since publication.

### UNIX

On-line files are located in *\$JDBCLOCATION/doc/release*, where *\$JDBCLOCATION* refers to the directory where you installed Informix JDBC Driver. ♦

### Windows

On-line files are located in *%JDBCLOCATION%\doc\release*, where *%JDBCLOCATION%* refers to the directory where you installed Informix JDBC Driver. ♦

Please examine these files because they contain vital information about application and performance issues.

## Vendor-Specific Documentation

For more information about the JDBC API, visit the JavaSoft Web site at:

<http://www.javasoft.com/products/jdk/1.1/docs/guide/jdbc/index.html>

---

## Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

---

## Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual you are using
- Any comments you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.  
Technical Publications  
300 Lakeside Dr., Suite 2700  
Oakland, CA 94612

If you prefer to send electronic mail, our address is:

`doc@informix.com`

We appreciate your suggestions.



---

# Getting Started

In This Chapter . . . . .	1-3
What Is JDBC? . . . . .	1-3
What Is a JDBC Driver? . . . . .	1-5
Overview of Informix JDBC Driver . . . . .	1-6
Installing the Driver . . . . .	1-7
Interactive Installation . . . . .	1-7
Silent Installation . . . . .	1-10
Uninstalling the Driver . . . . .	1-12
Using the Driver in an Application . . . . .	1-13
Using the Driver in an Applet . . . . .	1-14



## In This Chapter

This chapter provides an overview of Informix JDBC Driver and the JDBC API. It includes the following sections:

- “What Is JDBC?”
- “What Is a JDBC Driver?”
- “Overview of Informix JDBC Driver”
- “Installing the Driver”
- “Uninstalling the Driver”
- “Using the Driver in an Application”
- “Using the Driver in an Applet”

---

## What Is JDBC?

Java database connectivity (JDBC) is the JavaSoft specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language.

Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results.

The JDBC API is consistent with the style of the core Java interfaces and classes, such as **java.lang** and **java.awt**. The following table describes the interfaces, classes, and exceptions that make up the JDBC API.

Interface, Class, or Exception	Description
java.sql.CallableStatement	Interface used to execute stored procedures.
java.sql.Connection	Interface used to establish a connection to a database. SQL statements run within the context of a connection.
java.sql.DatabaseMetaData	Interface used to return information about the database.
java.sql.Driver	Interface used to locate the driver for a particular database management system.
java.sql.PreparedStatement	Interface used to send precompiled SQL statements to the database server and obtain results.
java.sql.ResultSet	Interface used to process the results returned from executing an SQL statement.
java.sql.ResultSetMetaData	Interface used to return information about the columns in a <b>ResultSet</b> object.
java.sql.Statement	Interface used to send static SQL statements to the database server and obtain results.
java.sql.Date	Subclass of <b>java.util.Date</b> used for the SQL DATE data type.
java.sql.DriverManager	Class used to manage a set of JDBC drivers.
java.sql.DriverPropertyInfo	Class used to discover and supply properties to a connection.
java.sql.Time	Subclass of <b>java.util.Date</b> used for the SQL TIME data type.
java.sql.TimeStamp	Subclass of <b>java.util.Date</b> used for the SQL TIMESTAMP data type.
java.sql.Types	Class used to define constants that are used to identify standard SQL data types, such as VARCHAR, INTEGER, and DECIMAL.

(1 of 2)

Interface, Class, or Exception	Description
java.sql.String	Class used to identify long text data types such as VARCHAR.
java.sql.DataTruncation	Exception thrown or warning reported when data has been truncated.
java.sql.SQLException	Exception that provides information about a database error.
java.sql.SQLWarning	Warning that provides information about a database warning.

(2 of 2)

Since JDBC is a standard specification, one Java program that uses the JDBC API can connect to any database management system (DBMS), as long as a driver exists for that particular DBMS.

For more information about the JDBC API, visit the JavaSoft Web site at:

<http://www.javasoft.com/products/jdk/1.1/docs/guide/jdbc/index.html>

## What Is a JDBC Driver?

The JDBC API defines the Java interfaces and classes that programmers use to connect to databases and send queries. A JDBC driver implements these interfaces and classes for a particular DBMS vendor.

A Java program that uses the JDBC API loads the specified driver for a particular DBMS before it actually connects to a database. The JDBC **Driver-Manager** class then sends all JDBC API calls to the loaded driver.

There are four types of JDBC drivers:

- JDBC-ODBC bridge plus ODBC driver, also called Type 1.  
Translates JDBC API calls into Microsoft ODBC calls that are then passed to the ODBC driver. The ODBC binary code must be loaded on every client computer that uses this type of driver.  
ODBC is an acronym for Open Database Connectivity.

- Native-API, partly Java driver, also called Type 2.  
Converts JDBC API calls into DBMS-specific client API calls. Like the bridge driver, this type of driver requires that some binary code be loaded on each client computer.
- JDBC-Net, pure-Java driver, also called Type 3.  
Sends JDBC API calls to a middle-tier net server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.
- Native-protocol, pure-Java driver, also called Type 4.  
Converts JDBC API calls directly into the DBMS-specific network protocol without a middle tier. This allows the client applications to connect directly to the database server.

---

## Overview of Informix JDBC Driver

Informix JDBC Driver is a native-protocol, pure-Java driver (Type 4). This means that when you use Informix JDBC Driver in a Java program that uses the JDBC API to connect to an Informix database, your session connects directly to the database or database server, without a middle tier.

Informix JDBC Driver is based on Version 1.22 of the JDBC API.

Informix JDBC Driver is released as a Java class file called **setup.class**. For instructions on how to install the driver, refer to “Installing the Driver” on page 1-7.

The product (after installation) consists of the following files, some of which are Java archive (JAR) files:

- **lib/ifxjdbc.jar**  
JAR file that contains the optimized implementations of the JDBC API interfaces, classes, and methods.  
The file is compiled with the **-O** option of the **javac** command.
- **lib/ifxjdbc-g.jar**  
Debug version of **ifxjdbc.jar**.  
The file is compiled with the **-g** option of the **javac** command.

- **demo/basic/\***
- **demo/rmi/\***
- **demo/stores7/\***
- **demo/clob-blob/\***
- **demo/udt-distinct/**

Directories that contain the sample Java programs that use the JDBC API. For descriptions of these sample files, see Appendix A, “Sample Code Files.”

- **proxy/IfxJDBCProxy.class**  
HTTP tunneling proxy class file.
- **proxy/SessionMgr.class**  
Session manager class file supporting the HTTP tunneling proxy.
- **doc/release/\***  
Directory that contains the on-line release and documentation notes, as well as the HTML and PDF versions of this programmer’s guide.

---

## Installing the Driver

Informix JDBC Driver is released as a Java class file called **setup.class**.

There are two ways to install the driver: using a **Setup** program, or using the command line. The following sections describe the two ways for both UNIX and Windows.

### Interactive Installation

This section describes how to interactively install Informix JDBC Driver with the **Setup** program.

#### To interactively install Informix JDBC Driver on UNIX

1. If you are installing Informix JDBC Driver from a CD-ROM, load the disc into the CD-ROM drive.

On Hewlett-Packard platforms, you must use the **-o cdcase** option of the **mount** command to read the CD in case-sensitive mode.



2. Copy the **ifxjdbc\_version.tar** file from the Web or the CD into a temporary directory (*not* the directory into which you are installing Informix JDBC Driver). The **version** is the product version: for example, 1.40.JC2.

**Warning:** *If you copy the tar file to the same directory into which you attempt to install the driver, the installation fails.*

3. Execute the following command:

```
tar xvf ifxjdbc_version.tar
```

The **setup.class** and **install.txt** files appear in the temporary directory.

4. Be sure your **CLASSPATH** environment variable points to Version 1.1.5 or later of the Java Development Kit (JDK).
5. At the UNIX shell prompt, create a directory to hold the contents of the driver.

For example, to create the directory **/work/jdbcdriver\_home**, execute the following command:

```
mkdir /work/jdbcdriver_home
```

6. Change directory to the temporary directory that contains the **setup.class** file.
7. Launch the **Setup** program with the **java** command at the UNIX shell prompt:

```
java setup
```

8. The **Setup** program guides you through the installation of Informix JDBC Driver.

The following warning message might appear:

```
Font specified in font.properties not found [-b&h-lucida sans  
typewriter-bold-r-normal-sans-*-%d-*-*m-*-*iso8859-1]
```

This condition does not affect the installation.

After the **Welcome** window, the program asks you for your serial number and key. It then asks you to accept a licensing agreement. The program then asks you for the name of the directory that will hold the contents of the driver. In this example, this directory is called **/work/jdbcdriver\_home** and was created in Step 5 of these instructions.

The installation is complete when you get to the **Installation Complete** window. ♦





### To interactively install Informix JDBC Driver on Windows

1. If you are installing Informix JDBC Driver from a CD-ROM, load the disc into the CD-ROM drive.
2. Copy the **ifxjdbc\_<i>version</i>.tar** file from the Web or the CD into a temporary directory (*not* the directory into which you are installing Informix JDBC Driver). The **<i>version</i>** is the product version: for example, 1.40.JC2.

**Warning:** *If you copy the tar file to the same directory into which you attempt to install the driver, the installation fails.*

3. Use WinZip or a similar utility to unpack the tar file. The **setup.class** and **install.txt** files appear in the temporary directory.
4. Be sure your **CLASSPATH** environment variable points to Version 1.1.5 or later of the Java Development Kit (JDK).
5. Using Windows Explorer, create a directory to hold the contents of the driver.

Assume, for this example, that the new directory is called **c:\work\jdbcdriver\_home**.

6. Change directory to the temporary directory that contains the **setup.class** file.
7. Launch the **Setup** program with the **java** command at the Windows command prompt:

```
java setup
```

8. The **Setup** program guides you through the installation of Informix JDBC Driver.

The following warning message might appear:

```
Font specified in font.properties not found [-b&h-lucida sans
typewriter-bold-r-normal-sans-*-%d-**-m*-iso8859-1]
```

This condition does not affect the installation.

After the **Welcome** window, the program asks you for your serial number and key. It then asks you to accept a licensing agreement. The program then asks you for the name of the directory that will hold the contents of the driver. In this example, this directory is called **c:\work\dbcdriver\_home** and was created in Step 5 of these instructions.

The installation is complete when you get to the **Installation Complete** window. ♦

UNIX



## Silent Installation

This section describes how to silently install Informix JDBC Driver from the UNIX shell prompt or Windows command line.

### To silently install Informix JDBC Driver on UNIX

1. If you are installing Informix JDBC Driver from a CD-ROM, load the disc into the CD-ROM drive.  
On Hewlett-Packard platforms, you must use the **-o cdcase** option of the **mount** command to read the CD in case-sensitive mode.
2. Copy the **ifxjdbc\_version.tar** file from the Web or the CD into a temporary directory (*not* the directory into which you are installing Informix JDBC Driver). The **version** is the product version: for example, 1.40.JC2.

**Warning:** If you copy the tar file to the same directory into which you attempt to install the driver, the installation fails.

3. Execute the following command:

```
tar xvf ifxjdbc_version.tar
```

The **setup.class** and **install.txt** files appear in the temporary directory.

4. Be sure your **CLASSPATH** environment variable points to Version 1.1.5 or later of the Java Development Kit (JDK).
5. At the UNIX shell prompt, create a directory to hold the contents of the driver.

For example, to create the directory **/work/jdbcdriver\_home**, execute the following command:

```
mkdir /work/jdbcdriver_home
```

6. Change directory to the temporary directory that contains the **setup.class** file.

- Execute the following command at the UNIX shell prompt:

```
java setup -o <directory> serialNo=<serial_no> key=<key>
```

In this command, *directory* refers to the directory that will hold the contents of the driver (created in Step 5 of these instructions), and *serial\_no* and *key* refer to the installation serial number and key.

The keywords **serialNo** and **key** are case sensitive. You can also use the keywords **SERIALNO**, **serialno**, and **KEY**.

For example, to install Informix JDBC Driver in the directory **/work/jdbcdriver\_home** using a serial number of **INF#J123456** and a key of **ABCDEF**, execute the following command:

```
java setup -o /work/jdbcdriver_home serialNo=INF#J123456 key=ABCDEF
```

If the specified directory already contains Informix JDBC Driver files, the command asks you if you want to overwrite them.

The installation is complete after the command has finished executing. ♦

## Windows

### To silently install Informix JDBC Driver on Windows

- If you are installing Informix JDBC Driver from a CD-ROM, load the disc into the CD-ROM drive.
- Copy the **ifxjdbc\_version.tar** file from the Web or the CD into a temporary directory (*not* the directory into which you are installing Informix JDBC Driver). The **version** is the product version: for example, 1.40.JC2.

**Warning:** *If you copy the tar file to the same directory into which you attempt to install the driver, the installation fails.*

- Use WinZip or a similar utility to unpack the tar file. The **setup.class** and **install.txt** files appear in the temporary directory.
- Be sure your **CLASSPATH** environment variable points to Version 1.1.5 or later of the Java Development Kit (JDK).
- Using Windows Explorer, create a directory to hold the contents of the driver.  
Assume, for this example, that the new directory is called **c:\work\jdbcdriver\_home**.
- Change directory to the temporary directory that contains the **setup.class** file.



7. Execute the following command at the Windows command prompt:

```
java setup -o <directory> serialNo=<serial_no> key=<key>
```

In this command, *directory* refers to the directory that will hold the contents of the driver (created in Step 5 of these instructions), and *serial\_no* and *key* refer to the installation serial number and key.

The keywords **serialNo** and **key** are case sensitive. You can also use the keywords **SERIALNO**, **serialno**, and **KEY**.

For example, to install Informix JDBC Driver in the directory **c:\work\jdbcdriver\_home** using a serial number of **INF#J123456** and a key of **ABCDEF**, execute the following command:

```
java setup -o c:\work\jdbcdriver_home serialNo=INF#J123456 key=ABCDEF
```

If the directory already contains Informix JDBC Driver files, the command asks you if you want to overwrite them.

The installation is complete once the command has finished executing. ♦

---

## Uninstalling the Driver

Uninstalling Informix JDBC Driver completely removes the driver and all of its components from your computer. The following sections describe how to uninstall Informix JDBC Driver on UNIX and Windows.

### UNIX

#### To uninstall Informix JDBC Driver on UNIX

1. Change to the directory in which you installed Informix JDBC Driver.

For example, if you installed the driver in the directory **/work/jdbcdriver\_home**, execute the following command at the UNIX shell prompt:

```
cd /work/jdbcdriver_home
```

2. Launch the **Uninstall** program with the **java** command:

```
java uninstall
```

3. The **Uninstall** program guides you through the uninstallation of Informix JDBC Driver. ♦

## Windows

## To uninstall Informix JDBC Driver on Windows

1. Change to the directory in which you installed Informix JDBC Driver. For example, if you installed the driver in the directory `c:\work\jdbcdriver_home`, execute the following command at the command prompt:

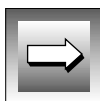
```
cd c:\work\jdbcdriver_home
```

2. Launch the **Uninstall** program with the **java** command:

```
java uninstall
```

3. The **Uninstall** program guides you through the uninstallation of Informix JDBC Driver. ♦

***Important:** When you uninstall Informix JDBC Driver, you always get a message that says the `ifxjdbc.jar` and `ifxjdbc-g.jar` files have changed, even if you have never used the driver. This is because the files are automatically written to during the installation of the driver.*




---

## Using the Driver in an Application

To use Informix JDBC Driver in an application, you must set your **CLASSPATH** environment variable to point to the driver files. The **CLASSPATH** environment variable tells the Java virtual machine (JVM) and other applications where to find the Java class libraries used in a Java program.

There are two ways of setting your **CLASSPATH** environment variable:

- Add the full pathname of the `ifxjdbc.jar` file to the **CLASSPATH** environment variable, as shown in the following example:

```
setenv CLASSPATH /work/jdbcdriver_home/lib/ifxjdbc.jar:$CLASSPATH
```

To use the version of the driver that supports debugging, specify the file `ifxjdbc-g.jar` instead of `ifxjdbc.jar`.

- Unpack the `ifxjdbc.jar` file and add its directory to the **CLASSPATH** environment variable, as shown in the following example:

```
cd /work/jdbcdriver_home/lib
jar xvf ifxjdbc.jar
setenv CLASSPATH /work/jdbcdriver_home/lib:$CLASSPATH
```

To use the version of the driver that supports debugging, specify the file `ifxjdbc-g.jar` instead of `ifxjdbc.jar`. ♦

## UNIX

There are two ways of setting your **CLASSPATH** environment variable:

- Add the full pathname of the **ifxjdbc.jar** file to the **CLASSPATH** environment variable, as shown in the following example:

```
set CLASSPATH=c:\work\jdbcdriver_home\lib\ifxjdbc.jar;%CLASSPATH%
```

To use the version of the driver that supports debugging, specify the file **ifxjdbc-g.jar** instead of **ifxjdbc.jar**.

- Unpack the **ifxjdbc.jar** file and add its directory to the **CLASSPATH** environment variable, as shown in the following example:

```
cd c:\work\jdbcdriver_home\lib
jar xvf ifxjdbc.jar
set CLASSPATH=c:\work\jdbcdriver_home\lib;%CLASSPATH%
```

To use the version of the driver that supports debugging, specify the file **ifxjdbc-g.jar** instead of **ifxjdbc.jar**. ♦

For more information on the **jar** utility, refer to the JavaSoft documentation at <http://www.javasoft.com>.

---

## Using the Driver in an Applet

You can use Informix JDBC Driver in an applet to connect to an Informix database from a browser such as Netscape Navigator or Microsoft Internet Explorer. The following steps show how to specify Informix JDBC Driver in the applet and how to ensure that the driver is correctly downloaded from the Web server.

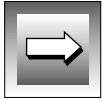
### To use Informix JDBC Driver in an applet

1. Install the **ifxjdbc.jar** file in the same directory as your applet class file.  
To use the version of the driver that supports debugging, install the file **ifxjdbc-g.jar** instead of **ifxjdbc.jar**.

2. Specify the **ifxjdbc.jar** file in the ARCHIVE attribute of the APPLET tag in your HTML file, as shown in the following example:

```
<APPLET ARCHIVE=ifxjdbc.jar CODE=my_applet.class  
CODEBASE=http://www.myhost.com WIDTH=460 HEIGHT=160>  
</APPLET>
```

To use the version of the driver that supports debugging, specify the file **ifxjdbc-g.jar** instead of **ifxjdbc.jar**.



**Important:** A few browsers do not support the ARCHIVE attribute of the APPLET tag. If you think this is true of the browsers that are going to download your applet, you must unpack and install the **ifxjdbc.jar** file in the root directory of your Web server.

If the browsers also do not support the JDBC API, you must install the class files included in the **java.sql** package in the root directory of the Web server as well.

See your Web server documentation for information on installing files in the root directory.





---

# Programming with Informix JDBC Driver

In This Chapter . . . . .	2-3
Establishing a Connection . . . . .	2-3
Loading Informix JDBC Driver . . . . .	2-4
Creating a Connection . . . . .	2-4
Format of Database URLs . . . . .	2-5
Connecting from Trusted Clients . . . . .	2-7
Database Versus Database Server Connections . . . . .	2-8
Specifying Environment Variables with the Properties Class . . . . .	2-9
Supported Informix Environment Variables . . . . .	2-11
Accessing Database Metadata . . . . .	2-14
Querying the Database . . . . .	2-15
Manipulating Informix Large Object Data Types . . . . .	2-15
Caching Large Objects . . . . .	2-16
Examples . . . . .	2-16
Manipulating Informix INTERVAL Data Types . . . . .	2-21
Manipulating Other Data Types . . . . .	2-23
Informix-Specific Information About Querying a Database . . . . .	2-23
Example of Sending a Query to an Informix Database . . . . .	2-25
Escape Syntax . . . . .	2-26
Mapping Data Types . . . . .	2-26
Mapping Between Informix and JDBC Data Types . . . . .	2-27
Supported ResultSet.getXXX() Methods . . . . .	2-29
Handling Errors . . . . .	2-31
Using the SQLException Class . . . . .	2-31
Retrieving Informix Error Message Text . . . . .	2-32

Internationalization. . . . .	2-33
JDK 1.1 and 1.2 Internationalization Support. . . . .	2-33
Support for Informix GLS Variables. . . . .	2-33
Support for End-User Formats. . . . .	2-34
GL_DATE Variable . . . . .	2-35
DBDATE Variable. . . . .	2-39
DBCENTURY Variable . . . . .	2-41
Precedence Rules Regarding DATE Value End-User Formats . . . . .	2-41
Support for Code Set Conversion. . . . .	2-42
Unicode to Database Code Set . . . . .	2-43
Unicode to Client Code Set . . . . .	2-46
Connecting to a Database with Non-ASCII Characters . . . . .	2-46
Code Set Conversion for TEXT Data Types . . . . .	2-47
Handling Transactions. . . . .	2-48
Other Informix Extensions to the JDBC API . . . . .	2-49
Using the Informix SERIAL and SERIAL8 Data Types . . . . .	2-49
Obtaining Driver Version Information . . . . .	2-50
Using an HTTP Proxy Server . . . . .	2-51
Restrictions and Limitations. . . . .	2-53

## In This Chapter

This chapter explains the Informix-specific information you need to use Informix JDBC Driver to connect to an Informix database. The chapter includes the following sections:

- “Establishing a Connection”
- “Accessing Database Metadata”
- “Querying the Database”
- “Mapping Data Types”
- “Handling Errors”
- “Internationalization”
- “Handling Transactions”
- “Other Informix Extensions to the JDBC API”
- “Using an HTTP Proxy Server”

---

## Establishing a Connection

You must first establish a connection to an Informix database server or database before you can start sending queries and receiving results in your Java program.

You establish a connection by completing two actions:

1. Load Informix JDBC Driver.
2. Create a connection to either a database server or a specific database.

## Loading Informix JDBC Driver

To load Informix JDBC Driver, use the **Class.forName()** method, passing it the value `com.informix.jdbc.IfxDriver`, as shown in the following code example from the **CreateDB.java** program:

```
try
{
    Class.forName("com.informix.jdbc.IfxDriver");
}
catch (Exception e)
{
    System.out.println("ERROR: failed to load Informix JDBC driver.");
    e.printStackTrace();
    return;
}
```

The **Class.forName()** method loads the Informix implementation of the **Driver** class, **IfxDriver**. The **IfxDriver** class then creates an instance of the driver and registers it with the **DriverManager** class.

Once you have loaded Informix JDBC Driver, you are ready to connect to an Informix database or database server.

### Windows

If you are writing an applet to be viewed with Microsoft Internet Explorer, you might need to explicitly register Informix JDBC Driver to avoid platform incompatibilities.

To explicitly register the driver, use the **DriverManager.registerDriver()** method, as shown in the following code example:

```
DriverManager.registerDriver((Driver)
    Class.forName("com.informix.jdbc.IfxDriver").newInstance());
```

This method might register Informix JDBC Driver twice, which does not cause a problem. ♦

## Creating a Connection

To create a connection to an Informix database or database server, use the **DriverManager.getConnection()** method. This method creates a **Connection** object, which is later used to create SQL statements, send them to an Informix database, and process the results.

The **DriverManager** class keeps track of the available drivers and handles connection requests between appropriate drivers and databases or database servers. The **url** parameter of the **getConnection()** method is a database URL that specifies the subprotocol (the database connectivity mechanism), the database or database server identifier, and a list of properties. A second parameter to the **getConnection()** method, **property**, is the property list. See “Specifying Environment Variables with the Properties Class” on page 2-9 for an example of how to specify a property list.

The following example shows a database URL that connects to a database called **testDB**:

```
jdbc:informix-sqli://123.45.67.89:1533/testDB:INFORMIXSERVER=myserver;
user=rdtest;password=test
```

The details of the database URL syntax are described in the next section.

The following code example from the **CreateDB.java** program shows how to connect to database **testDB** using Informix JDBC Driver. In the full example, the **url** variable, described in the preceding example, is passed in as a parameter when the program is run at the command line.

```
try
{
    conn = DriverManager.getConnection(url);
}
catch (SQLException e)
{
    System.out.println("ERROR: failed to connect!");
    System.out.println("ERROR: " + e.getMessage());
    e.printStackTrace();
    return;
}
```

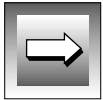


**Important:** *Not all methods of the **Connection** interface are supported by Informix JDBC Driver. For a list of unsupported methods, see “Restrictions and Limitations” on page 2-51.*

## Format of Database URLs

Informix JDBC Driver supports database URLs of the following format:

```
jdbc:informix-sqli://[ip-address|host-name]:port-number][dbname]:
INFORMIXSERVER=server-name;[user=user;password=password]
[;name=value;name=value...]
```



In the preceding syntax:

- curly brackets ( {} ) together with vertical lines ( | ) denote more than one choice of variable.
- *italics* denote a variable value.
- brackets ( [] ) denote an optional value.
- words or symbols not enclosed in brackets are required (INFORMIXSERVER=, for example).

**Important:** *Blank spaces are not allowed in the database URL.*

The following table describes the variable parts of the database URL.

Database URL Variable	Required?	Description
<i>ip-address</i> <i>host-name</i>	Yes	The IP address or the host name of the computer running the Informix database server.  An example of an IP address is 123.45.67.89. An example of a host name is myhost.com or myhost.informix.com.
<i>port-number</i>	Yes	The port number of the Informix database server.
<i>dbname</i>	No	The name of the Informix database to which you want to connect. If you do not specify the name of a database, a connection is made to the Informix database server.

(1 of 2)

Database URL Variable	Required?	Description
<i>server-name</i>	Yes	The name of the Informix server to which you want to connect. This is the value of the <b>INFORMIXSERVER</b> environment variable.  The <b>INFORMIXSERVER</b> environment variable is required in the database URL, unless it is included in the property list.
<i>user</i> <i>password</i>	Yes, unless client is trusted	The name of the user that wants to connect to the Informix database or database server, and the password of that user. You must specify both the user and the password if the client is not trusted. If the client is trusted, you can specify both or neither.
<i>name=value</i>	No	A name-value pair that specifies a <i>value</i> for the Informix environment variable contained in the <i>name</i> variable, recognized by either Informix JDBC Driver or Informix database servers. The <i>name</i> variable is case insensitive.  Informix JDBC Driver reads Informix environment variables from either the database URL or from a connection property list, described in “Specifying Environment Variables with the Properties Class” on page 2-9. The user’s environment is not consulted.  Refer to “Supported Informix Environment Variables” on page 2-11 for a list of Informix environment variables supported by Informix JDBC Driver.

(2 of 2)

### ***Connecting from Trusted Clients***

For a trusted client, the *user* and *password* properties are optional. If these are not specified, the database connection uses the current user and no password. You must specify both *user* and *password* or neither; specifying one without the other generates an error. A connection that does not use a password is unsuccessful unless the client is trusted by the server.

## Database Versus Database Server Connections

Using the **DriverManager.getConnection()** method, you can create a connection to either an Informix database or an Informix database server.

To create a connection to an Informix database, specify the name of the database in the *dbname* variable of the database URL. If you omit the name of a database, a connection is made to the database server specified by the **INFORMIXSERVER** environment variable of the database URL or the connection property list.

If you connect directly to an Informix database server, you can execute an SQL statement that connects to a database later in your Java program.

All connections to both databases and database servers must include the name of an Informix database server via the **INFORMIXSERVER** environment variable.



**Important:** *If you are connecting to a 5.x database server (either **INFORMIX-OnLine Dynamic Server** or **INFORMIX-SE**), you must specify the **USEV5SERVER** environment variable in the database URL or property list. Its value should be 1, such as “**USEV5SERVER=1**”.*

The example given in “Creating a Connection” on page 2-4 shows how to create a connection directly to the Informix database called **testDB** with the database URL.

The following example from the **DBConnection.java** program shows how to first create a connection to the Informix database server called **myserver** and then connect to the database **testDB** later in the Java program using the **Statement.executeUpdate()** method.

The following database URL is passed in as a parameter to the program when the program is run at the command line; note that the URL does not include the name of a database:

```
jdbc:informix-sqli://123.45.67.89:1533:INFORMIXSERVER=myserver;user=rdtest;  
password=test
```



Here is the example code:

```
String cmd = null;
int rc;

Connection conn = null;

try
{
    Class.forName("com.informix.jdbc.IfxDriver");
}
catch (Exception e)
{
    System.out.println("ERROR: failed to load Informix JDBC driver.");
}

try
{
    conn = DriverManager.getConnection(url);
}
catch (SQLException e)
{
    System.out.println("ERROR: failed to connect!");
}

try
{
    Statement stmt = conn.createStatement();
    cmd = "database testDB";
    rc = stmt.executeUpdate(cmd);
    stmt.close();
}
catch (SQLException e)
{
    System.out.println("ERROR: execution failed - statement: " + cmd);
    System.out.println("ERROR: " + e.getMessage());
}
```

### ***Specifying Environment Variables with the Properties Class***

Informix JDBC Driver reads Informix environment variables only from the name-value pairs in the connection database URL or from a connection property list. The driver does not consult the user's environment for any environment variables. Refer to "Supported Informix Environment Variables" on page 2-11 for a list of supported Informix environment variables.

To specify Informix environment variables in the name-value pairs of the connection database URL, refer to "Format of Database URLs" on page 2-5.

To specify Informix environment variables via a property list, use the **java.util.Properties** class to build the list of properties. The list of properties might include Informix environment variables, such as **INFORMIXSERVER**, as well as *user* and *password*. After you have built the property list, pass it to the **DriverManager.getConnection()** method as a second parameter. You still need to include a database URL as the first parameter, although in this case you do not need to include the list of properties in the URL.

The following code from the **optofc.java** example shows how to use the **java.util.Properties** class to set connection properties. It first uses the **Properties.put()** method to set the environment variable **OPTOFC** to **1** in the connection property list; then it connects to the database.

The **DriverManager.getConnection()** method in this example takes two parameters: the database URL and the property list. The example creates a connection similar to the example given in “Creating a Connection” on page 2-4.

```
try
{
    Class.forName("com.informix.jdbc.IfxDriver");
}
catch (Exception e)
{
    System.out.println("ERROR: failed to load Informix JDBC driver.");
}

try
{
    Properties pr = new Properties();
    pr.put("OPTOFC","1");
    conn = DriverManager.getConnection(newUrl, pr);
}
catch (SQLException e)
{
    System.out.println("ERROR: failed to connect!");
}
```

## Supported Informix Environment Variables

The following table lists the Informix environment variables supported by Informix JDBC Driver.

Supported Informix Environment Variables	Description
CLIENT_LOCALE	Specifies the locale of the client that is accessing the database. Together with the <b>DB_LOCALE</b> variable, the server uses this variable to establish the server processing locale. This variable is available on and optional for servers that support GLS.
DBANSIWARN	Checks for Informix extensions to ANSI standard syntax.
DBCENTURY	Enables you to specify the appropriate expansion for one- or two-digit year DATE and DATETIME values. See the <i>Informix Guide to SQL: Reference</i> for detailed information on this environment variable.
DBDATE	Specifies the end-user formats of values in DATE columns.
DB_LOCALE	Specifies the locale of the database. Together with the <b>CLIENT_LOCALE</b> variable, the server uses this variable to establish the server processing locale. This variable is available on and optional for servers that support GLS.
DBSPACETEMP	Specifies the dbspaces in which temporary tables are built.
DBUPSPACE	Specifies the amount of system disk space that the UPDATE STATISTICS statement can use when it simultaneously constructs multiple-column distributions.
DELIMIDENT	When set to Y, specifies that strings set off by double quotes are delimited identifiers.
FET_BUF_SIZE	Overrides the default setting for the size of the fetch buffer for all data except large objects. The default size is 4096 bytes.

(1 of 4)

Supported Informix Environment Variables	Description
GL_DATE	Specifies the end-user formats of values in DATE columns. This variable is supported in Informix database servers 7.2x and beyond.
IFX_AUTOFREE	<p>When set to 1, specifies that the <b>Statement.close()</b> method does not require a network round-trip to free the server cursor resources if the cursor has already been closed in the database server.</p> <p>The database server automatically frees the cursor resources after the cursor is closed, either explicitly by the <b>ResultSet.close()</b> method or implicitly by the <b>OPTOFC</b> environment variable.</p> <p>Once the cursor resources have been freed, the cursor can no longer be referenced.</p>
INFORMIXCONRETRY	Specifies the maximum number of additional connection attempts that can be made to each database server by the client during the time limit specified by the default value of the <b>INFORMIXCONTIME</b> environment variable (15 seconds).
INFORMIXCONTIME	Sets the timeout period for an attempt to connect to the server. If a connection attempt does not succeed in this time, the attempt is aborted and a connection error is reported. The default value is 15 seconds.
INFORMIXOPCACHE	Specifies the size of the memory cache for the staging-area blob space of the client application.
INFORMIXSERVER	Specifies the default database server to which an explicit or implicit connection is made by a client application.
INFORMIXSTACKSIZE	Specifies the stack size, in kilobytes, that the database server uses for a particular client session.

(2 of 4)

Supported Informix Environment Variables	Description
LOBCACHE	<p>Determines the buffer size for large object data that is fetched from the database server:</p> <ul style="list-style-type: none"> <li>■ If <b>LOBCACHE</b> &gt; 0, the maximum <b>LOBCACHE</b> number of bytes is allocated in memory to hold the data. If the data size exceeds the <b>LOBCACHE</b> value, the data is stored in a temporary file. If a security violation occurs during creation of this file, the data is stored in memory.</li> <li>■ If <b>LOBCACHE</b> = 0, the data is always stored in a file. In this case, if a security violation occurs, Informix JDBC Driver makes no attempt to store the data in memory.</li> <li>■ If <b>LOBCACHE</b> &lt; 0, the data is always stored in memory. If the required amount of memory is not available, an error occurs.</li> </ul> <p>If the <b>LOBCACHE</b> value is not specified, the default is 4096 bytes.</p>
NODEFDAC	<p>When set to <b>YES</b>, prevents default table and routine privileges from being granted to the <b>PUBLIC</b> user when a new table or routine is created in a database that is not ANSI compliant. Default is <b>NO</b>.</p>
OPTCOMPIND	<p>Specifies the join method that the query optimizer uses.</p>
OPTOFC	<p>When set to 1, the <b>ResultSet.close()</b> method does not require a network round-trip if all the qualifying rows have already been retrieved in the client's tuple buffer. The database server automatically closes the cursor after all the rows have been retrieved.</p> <p>Informix JDBC Driver might not have additional rows in the client's tuple buffer before the next <b>ResultSet.next()</b> method is called. Therefore, unless Informix JDBC Driver has received all the rows from the database server, the <b>ResultSet.close()</b> method might still require a network round-trip when <b>OPTOFC</b> is set to 1.</p>
PATH	<p>Specifies the directories that should be searched for executable programs.</p>

(3 of 4)

---

Supported Informix Environment Variables	Description
PDQPRIORITY	Determines the degree of parallelism used by the database server.
PLCONFIG	Specifies the name of the configuration file used by the high-performance loader.
PSORT_DBTEMP	Specifies one or more directories to which the database server writes the temporary files it uses when performing a sort.
PSORT_NPROCS	Enables the database server to improve the performance of the parallel-process sorting package by allocating more threads for sorting.
USEV5SERVER	When set to 1, specifies that the Java program is connecting to an INFORMIX-OnLine or INFORMIX-SE 5.x database server.  This environment variable is mandatory if you are connecting to an INFORMIX-OnLine or INFORMIX-SE 5.x database server.

---

(4 of 4)

For a detailed description of a particular environment variable, refer to *Informix Guide to SQL: Reference*. You can find the on-line version of this guide at <http://www.informix.com/answers>.

---

## Accessing Database Metadata

To access information about an Informix database, use the JDBC API **DatabaseMetaData** interface.

Informix JDBC Driver is completely compatible with the JDBC API specification for accessing database metadata. The driver supports all the methods of the **DatabaseMetaData** interface.

Informix JDBC Driver uses the **sysmaster** database to get database metadata. If you want to use the **DatabaseMetaData** interface in your Java program, the **sysmaster** database must exist in the Informix database server to which your Java program is connected.

Informix JDBC Driver interprets the JDBC API term *schemas* to mean the names of Informix users who own tables. The **DatabaseMetaData.getSchemas()** method returns all the users found in the **owner** column of the **systables** system catalog.

Similarly, Informix JDBC Driver interprets the JDBC API term *catalogs* to mean the names of Informix databases. The method **DatabaseMetaData.getCatalogs()** returns the names of all the databases that currently exist in the Informix database server to which your Java program is connected.

The example **DBMetaData.java** shows how to use the **DatabaseMetaData** and **ResultSetMetaData** interfaces to gather information about a new procedure. Refer to Appendix A, “Sample Code Files,” for the full text of this example.

---

## Querying the Database

Informix JDBC Driver complies with the JDBC API specification for sending queries to a database and retrieving the results. The driver supports almost all the methods of the **Statement**, **PreparedStatement**, **CallableStatement**, **ResultSet**, and **ResultSetMetaData** interfaces.

Refer to “Restrictions and Limitations” on page 2-51 for a list of methods that are not supported by Informix JDBC Driver and should not be used in your Java program. This reference also includes a list of methods that behave differently than described in the JDBC API specification.

## Manipulating Informix Large Object Data Types

This section describes the Informix **BYTE**, **TEXT**, **BLOB**, and **CLOB** data types and how to manipulate columns of these data types with the JDBC API.

The **BYTE** data type is a data type for a simple large object that stores any kind of data in an undifferentiated byte stream. Examples of binary data include spreadsheets, digitized voice patterns, and video clips. The **TEXT** data type is a data type for a simple large object that stores any kind of text data. It can contain both single and multibyte characters.

Columns of either data type have a theoretical limit of  $2^{31}$  bytes and a practical limit determined by your disk capacity.

Support for Informix smart large objects and BLOB or CLOB data types is only available with 9.x versions of the server. Columns of either data type have a theoretical limit of 4 terabytes and a practical limit determined by your disk capacity.

For more detailed information about the Informix BYTE, TEXT, BLOB, and CLOB data types, refer to *Informix Guide to SQL: Reference* and *Informix Guide to SQL: Syntax*. You can find the on-line version of both of these guides at <http://www.informix.com/answers>.

### ***Caching Large Objects***

Whenever a BLOB, CLOB, TEXT, or BYTE object is fetched from the server, the data is cached into memory. If the size of the large object is bigger than the value in the LOBCACHE environment variable, the large object data is stored in a temporary file. For more information about the LOBCACHE variable, see “Memory Management of Large Objects” on page 3-6.

### ***Examples***

The examples in this section illustrate the following features:

- Inserting into or updating BYTE and TEXT columns
- Selecting from BYTE and TEXT columns

These examples apply to BLOB and CLOB columns, respectively, as well. For additional examples, see Appendix A, “Sample Code Files.”

#### *Inserting into or Updating BYTE and TEXT Columns*

To insert into or update BYTE and TEXT columns, read a stream of data from a source, such as an operating system file, and transmit it to the database as a **java.io.InputStream** object. The **PreparedStatement** interface provides methods for setting an input parameter to this Java input stream. When the statement is executed, Informix JDBC Driver makes repeated calls to the input stream, reading its contents and transmitting those contents as the actual parameter data to the database.



For BYTE data types, use the **PreparedStatement.setBinaryStream()** method to set the input parameter to the **InputStream** object. For TEXT data types, use the **PreparedStatement.setAsciiStream()** method.

The following example from the **ByteType.java** program shows how to insert the contents of the operating system file **data.dat** into a column of data type **BYTE**:

```
try
{
    stmt = conn.createStatement();
    stmt.executeUpdate("create table tabl(col1 byte)");
}
catch ( SQLException e)
{
    System.out.println("Failed to create table ..." + e.getMessage());
}

System.out.println("Trying to insert data using Prepare Statement ...");
try
{
    pstmt = conn.prepareStatement("insert into tabl values (?)");
}
catch (SQLException e)
{
    System.out.println("Failed to Insert into tab:" + e.toString());
}

File file = new File("data.dat");
int fileLength = (int) file.length();
InputStream value = null;
FileInputStream fileinp = null;
int row = 0;
String str = null;
int rc = 0;
ResultSet rs = null;

System.out.println("Inserting data ...\\n");
try
{
    fileinp = new FileInputStream(file);
    value = (InputStream)fileinp;
}
catch (Exception e) {}

try
{
    pstmt.setBinaryStream(1,value,10); //set 1st column
}
catch (SQLException e)
{
    System.out.println("Unable to set parameter");
}

set_execute();
...
public static void set_execute()
{
    try
    {
```

```

        pstmt.executeUpdate();
    }
    catch (SQLException e)
    {
        System.out.println("Failed to Insert into tab:" + e.toString());
        e.printStackTrace();
    }
}

```

The example first creates a **java.io.File** object that represents the operating system file **data.dat**. The example then creates a **FileInputStream** object to read from the **File** object. The **FileInputStream** object is cast to its superclass **InputStream**, which is the expected data type of the second parameter to the **PreparedStatement.setBinaryStream()** method. The **setBinaryStream()** method is executed on the already prepared INSERT statement, which sets the input stream parameter. Finally, the **PreparedStatement.executeUpdate()** method is executed, which actually inserts the contents of the **data.dat** operating system file into the **BYTE** column of the table.

The **TextType.java** program shows how to insert data into a **TEXT** column. It is very similar to inserting into a **BYTE** column, except the method **setAsciiStream()** is used to set the input parameter instead of **setBinaryStream()**.

### *Selecting from BYTE and TEXT Columns*

After you select from a table into a **ResultSet** object, you can use the **ResultSet.getBinaryStream()** and **ResultSet.getAsciiStream()** methods to retrieve a stream of binary or ASCII data from **BYTE** and **TEXT** columns, respectively. Both methods return an **InputStream** object, which can be used to read the data in chunks.

All the data in the returned stream in the current row must be read before you call the **next()** method to retrieve the next row.

The following example from the **ByteType.java** program shows to how select data from a **BYTE** column and print out the data to the standard output:

```
try
{
    stmt = conn.createStatement();
    rs = stmt.executeQuery("Select * from tabl");
    while( rs.next() )
    {
        row++;
        value = rs.getBinaryStream(1);
        System.out.println("\nResult of row #" + row + ", size = "
            + value.available() + " from getAsciiStream(1) ..\n");
        dispValue(value);
    }
}
catch (Exception e) { }
...
public static void dispValue(InputStream in)
{
    int size;
    byte buf;
    int count = 0;
    try
    {
        size = in.available();
        byte ary[] = new byte[size];
        buf = (byte) in.read();
        while(buf!=-1)
        {
            ary[count] = buf;
            count++;
            buf = (byte) in.read();
        }
        System.out.println( new String(ary).trim());
    }
    catch (Exception e)
    {
        System.out.println("Error occur during reading stream ... \n");
    }
}
}
```

The example first puts the result of a **SELECT** statement into a **ResultSet** object. It then executes the method **ResultSet.getBinaryStream()** to retrieve the **BYTE** data into a Java **InputStream** object.

The method **dispValue()**, whose Java code is also included in the example, is used to actually print out the contents of the column to standard output. The **dispValue()** method uses byte arrays and the **InputStream.read()** method to systematically read the contents of the **BYTE** column.

The **TextType.java** program shows how to select data from a TEXT column. It is very similar to selecting from a BYTE column, except the **getAsciiStream()** method is used instead of **getBinaryStream()**.

## Manipulating Informix INTERVAL Data Types

The Informix INTERVAL data type stores a value that represents a span of time. INTERVAL data types are divided into two types: year-month intervals and day-time intervals. A year-month interval can represent a span of years and months, and a day-time interval can represent a span of days, hours, minutes, seconds, and fractions of a second.

You can retrieve and insert Informix INTERVAL data types using the following methods:

- **ResultSet.getString()**
- **PreparedStatement.setString()**

The following example from the **Interval.java** program shows how to insert into and select from the two types of INTERVAL data types:

```
/*
 * Create a table with two interval columns
 */
System.out.println("");
System.out.println("Create a table");
try
{
    cmd = "create table intrvl_tab " +
        "(ym interval year to month, ds interval day to second)";

    int rowcount = stmt.executeUpdate(cmd);
    System.out.println("Create table rowcount = " + rowcount);
}
catch (SQLException e)
{
    System.out.println("ERROR: execution failed - statement: " + cmd);
    System.out.println("ERROR: " + e.getMessage());
    return;
}

/*
 * Insert an interval using a String host variable.
 */
System.out.println("");
System.out.println("Insert an interval using a String host variable");
try
{
```

## Manipulating Informix INTERVAL Data Types

```
String ym = new String("102-11"); // 102 years and 11 months
String ds = new String("5 10:03:55"); // 5 days 10 hours 3 minutes
// and 55 seconds
cmd = "insert into intrvl_tab values (?, ?)";
PreparedStatement pstmt = conn.prepareStatement(cmd);

pstmt.setString(1, ym);
pstmt.setString(2, ds);

int rowcount = pstmt.executeUpdate();
System.out.println("Insert rowcount = " + rowcount);
}
catch (SQLException e)
{
System.out.println("ERROR: execution failed - statement: " + cmd);
System.out.println("ERROR: " + e.getMessage());
return;
}

/*
 * Now fetch the row back
 */
System.out.println("");
System.out.println("Select from intrvl_tab");
try
{
cmd = "select * from intrvl_tab";
ResultSet rs = stmt.executeQuery(cmd);

String ym;
String ds;
while(rs.next())
{
ym = rs.getString(1);
System.out.println("  Select: ym = " + ym);

ds = rs.getString(2);
System.out.println("  Select: ds = " + ds);
}

}
catch (SQLException e)
{
System.out.println("ERROR: execution failed - statement: " + cmd);
System.out.println("ERROR: " + e.getMessage());
return;
}
```

## Manipulating Other Data Types

You can retrieve and insert Informix BOOLEAN data types using the following methods:

- **ResultSet.getBoolean()**
- **PreparedStatement.setBoolean()**

You can retrieve and insert Informix LVARCHAR data types using the following methods:

- **ResultSet.getString()**
- **PreparedStatement.setString()**

You can retrieve and insert opaque data types as long as a user-defined function casts the opaque type to a built-in type.

You can retrieve and insert distinct data types as their base types. For example, here is the SQL statement that defines the distinct type:

```
CREATE DISTINCT TYPE money_type AS NUMERIC(10, 2);
CREATE TABLE disttab (money_col money_type);
```

Here is an example of binding to the base type:

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO disttab " +
    "(money_col) VALUES (?)");
BigDecimal bigdec = new BigDecimal((double)123.45);
pstmt.setBigDecimal(1, bigdec);
pstmt.executeUpdate();
```

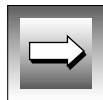
When you bind as the underlying type, Informix JDBC Driver binds as the underlying type on the client side, because the server provides implicit casting between the underlying type and the distinct type.

## Informix-Specific Information About Querying a Database

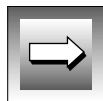
This section describes the Informix-specific information you need to know to use Informix JDBC Driver to query an Informix database and process the results.

The Informix JDBC Driver implementation of the **Statement.execute()** method returns a single **ResultSet** object. This implementation differs from the JDBC API specification, which states that the method can return multiple **ResultSet** objects.

Be sure to always explicitly close a **Statement**, **PreparedStatement**, and **CallableStatement** object by calling the appropriate **close()** method in your Java program when you have finished processing the results of an SQL statement. This closure immediately deallocates the resources that have been allocated to execute your SQL statement. Although the **ResultSet.close()** method closes the **ResultSet** object, it does *not* deallocate the resources allocated to the **Statement**, **PreparedStatement**, or **CallableStatement** objects.



**Important:** For best results, always call **ResultSet.close()** and **Statement.close()** methods to indicate to Informix JDBC Driver that you are done with the statement or result set. Otherwise, your program might not release all its resources on the database server.



**Important:** The same **Statement** or **ResultSet** instance cannot be accessed concurrently across threads. You can, however, share a **Connection** object between multiple threads.

For example, if one thread executes the **Statement.executeQuery()** method on a **Statement** object, and another thread executes the **Statement.executeUpdate()** method on the same **Statement** object, the results of both methods are unexpected and depend on which method was executed last.

Similarly, if one thread executes the method **ResultSet.next()** and another thread executes the same method on the same **ResultSet** object, the results of both methods are unexpected and depend on which method was executed last.



## Example of Sending a Query to an Informix Database

The following example from the **SimpleSelect.java** program shows how to use the **PreparedStatement** interface to execute a **SELECT** statement that has one input parameter:

```
try
{
    PreparedStatement pstmt = conn.prepareStatement("Select * from x where a = ?;");
    pstmt.setInt(1, 11);
    ResultSet r = pstmt.executeQuery();

    while(r.next())
    {
        short i = r.getShort(1);
        System.out.println("Select: column a = " + i);
    }
    r.close();
    pstmt.close();
}
catch (SQLException e)
{
    System.out.println("ERROR: Fetch statement failed: " + e.getMessage());
}
```

The program first uses the **Connection.prepareStatement()** method to prepare the **SELECT** statement with its single input parameter. It then assigns a value to the parameter using the **PreparedStatement.setObject()** method and executes the query with the **PreparedStatement.executeQuery()** method.

The program returns resulting rows in a **ResultSet** object, through which the program iterates with the **ResultSet.next()** method. The program retrieves individual column values with the **ResultSet.getShort()** method, since the data type of the selected column is **SMALLINT**.

Finally, both the **ResultSet** and **PreparedStatement** objects are explicitly closed with the appropriate **close()** method.

For more information on which **getXXX()** methods retrieve individual column values, refer to "Mapping Data Types" on page 2-25.

## Escape Syntax

Valid escape syntax for SQL statements is as follows:

Type of Statement	Escape Syntax
Stored procedure	{call <i>procedure</i> }
Stored procedure	{var = call <i>procedure</i> }
Date	{d 'yyyy-mm-dd' }
Time	{t 'hh:mm:ss' }
Timestamp (Datetime)	{ts 'yyyy-mm-dd hh:mm:ss[.ffff]' }
Function call	{fn <i>func</i> [( <i>args</i> )]}
Escape character	{escape ' <i>escape-char</i> ' }
Outer join	{oj <i>outer-join-statement</i> }

You can put any of the above in an SQL statement. For example:

```
executeUpdate("insert into tabl values( {d '1999-01-01'} )");
```

Everything inside the brackets is converted into a valid Informix SQL statement and returned to the calling function.

## Mapping Data Types

This section discusses mapping issues between data types defined in a Java program and the data types supported by the Informix database server. In particular, it covers the following two topics:

- Mapping between JDBC API data types and Informix data types
- **ResultSet.getXXX()** methods supported by Informix JDBC Driver

## Mapping Between Informix and JDBC Data Types

Since there are variations between the SQL data types supported by each database vendor, the JDBC API defines a set of *generic* SQL data types in the class **java.sql.Types**. Use these JDBC API data types to reference generic SQL types in your Java programs that use the JDBC API to connect to Informix databases.

The following table shows the Informix data type to which each JDBC API data type maps.

JDBC API Data Type	Informix Data Type
BIGINT	INT8
BINARY	BYTE
BIT	Not supported
CHAR	CHAR( <i>n</i> )
DATE	DATE
DECIMAL	DECIMAL
DOUBLE	FLOAT
FLOAT	SMALLFLOAT
INTEGER	INTEGER
LONGVARBINARY	BYTE
LONGVARCHAR	TEXT
NUMERIC	DECIMAL
REAL	SMALLFLOAT
SMALLINT	SMALLINT
TIME	DATETIME
TIMESTAMP	DATETIME

(1 of 2)

JDBC API Data Type	Informix Data Type
TINYINT	SMALLINT
VARBINARY	BYTE
VARCHAR	VARCHAR( <i>m,r</i> )

(2 of 2)

The *LONGVARBINARY* and *LONGVARCHAR* JDBC types can also map to Informix *BLOB* and *CLOB* types, respectively.



**Important:** Informix JDBC Driver maps **java.sql.Timestamp** to the Informix type *DATETIME YEAR TO FRACTION(5)* and **java.sql.Time** to the Informix type *DATETIME HOUR TO SECOND*. Informix *DATETIME* types are very restrictive and are not interchangeable. If you attempt to bind **java.sql.Time** to *DATETIME YEAR TO FRACTION(5)* or **java.sql.Timestamp** to *DATETIME HOUR TO SECOND*, you might get an error from the Informix database server. Any other *DATETIME* qualifiers are not supported.

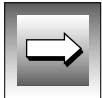
The following table lists mappings between the extended data types supported in Informix Dynamic Server with Universal Data Option and the corresponding Java and JDBC types.

JDBC Type	Java Object Type	Informix Type
java.sql.Types.OTHER	boolean	BOOLEAN
java.sql.Types.SMALLINT	smallint	IfxTypes.IFX_TYPE_BOOL
java.sql.Types.LONGVARCHAR	java.sql.String java.io.inputStream	LVARCHAR IfxTypes.IFX_TYPE_LVARCHAR
java.sql.Types.LONGVARBINARY	java.io.inputStream byte[]	BYTE, BLOB IfxTypes.IFX_TYPE_BYTE
java.sql.Types.LONGVARCHAR	java.io.InputStream java.sql.String	TEXT, CLOB IfxTypes.IFX_TYPE_TEXT

A Java **boolean** object can map to a **smallint** object or an Informix BOOLEAN data type. Informix JDBC Driver attempts to map it according to the column type. However, in cases such as **PreparedStatement** host variable bindings, Informix JDBC Driver cannot access the column types, so the mapping is somewhat limited.

## Supported ResultSet.getXXX() Methods

Use the **ResultSet.getXXX()** methods to transfer data from an Informix database to a Java program that uses the JDBC API to connect to an Informix database. For example, use the **ResultSet.getString()** method to get the data stored in a column of data type LVARCHAR.



**Important:** If you use an expression within an SQL statement—for example, `SELECT mytype::LVARCHAR FROM mytab`—you might not be able to use **ResultSet.getXXX(columnName)** to retrieve the value. Use **ResultSet.getXXX(columnIndex)** to retrieve the value instead.

The following table lists the **ResultSet.getXXX()** methods that Informix JDBC Driver supports. The top heading lists the standard JDBC API data types defined in the **java.sql.Types** class. These translate to specific Informix data types, as shown in the table on page 2-26. The table lists the **getXXX()** methods you can use to retrieve data of a particular JDBC API data type.

An uppercase and bold X indicates the **getXXX()** method Informix recommends you use; a lowercase x indicates other **getXXX()** methods supported by Informix JDBC Driver.

getXXX() Method	JDBC API Data Types from java.sql.Types																					
	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP	BLOB	CLOB	
<b>getBytes()</b>	X	x	x	x	x	x	x	x	x		x <sup>1</sup>	x <sup>1</sup>										
<b>getShort()</b>	x	X	x	x	x	x	x	x	x		x <sup>1</sup>	x <sup>1</sup>										
<b>getInt()</b>	x	x	X	x	x	x	x	x	x		x <sup>1</sup>	x <sup>1</sup>										

(1 of 2)

Supported ResultSet.getXXX() Methods

getXXX() Method	JDBC API Data Types from java.sql.Types																					
	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP	BLOB	CLOB	
getLong()	x	x	x	X	x	x	x	x	x		x <sup>1</sup>	x <sup>1</sup>										
getFloat()	x	x	x	x	X	x	x	x	x		x <sup>1</sup>	x <sup>1</sup>										
getDouble()	x	x	x	x	x	X	X	x	x		x <sup>1</sup>	x <sup>1</sup>										
getBigDecimal()	x	x	x	x	x	x	x	X	X		x	x										
getBoolean()	x	x	x	x	x	x	x	x	x		x	x										
getString()	x	x	x	x	x	x	x	x	x		X	X	x	x	x	x	x	x	x			x
getBytes()													x	X	X	x						x
getDate()											x	x					X		x			
getTime()											x	x						X	x			
getTimestamp()											x	x					x		X			
getAsciiStream()													X	x	x	x						X
getUnicodeStream()																						
getBinaryStream()													x	x	x	X						X
getObject()	x	x	x	x	x	x	x	x	x		x	x	x <sup>2</sup>	x	x	x <sup>2</sup>	x	x <sup>3</sup>	x	x	x	x

Notes:

<sup>1</sup> The column value must match the type of **getXXX()** exactly, or an **SQLException** is raised. If the column value is not within the allowed value range, the **getXXX()** method raises an exception instead of converting the data type. For example, **getBytes(1)** raises an **SQLException** if the column value is 1000.

<sup>2</sup> A byte array is returned.

<sup>3</sup> A **Timestamp** object is returned instead of a **Time** object.

(2 of 2)

The **getXXX()** methods return a null value if the retrieved column value is an SQL null value.

---

## Handling Errors

Use the JDBC API **SQLException** class to handle errors in your Java program. The Informix-specific **com.informix.jdbc.Message** class can also be used outside a Java program to retrieve the Informix error text for a given error number.

### Using the SQLException Class

Whenever an error occurs from either Informix JDBC Driver or the database server, an **SQLException** is raised. Use the following methods of the **SQLException** class to retrieve the text of the error message, the error code, and the **SQLSTATE** value:

- **getMessage()**  
Returns a description of the error. **SQLException** inherits this method from the **java.util.Throwable** class.
- **getErrorCode()**  
Returns an integer value that corresponds to the Informix database server or Informix JDBC Driver error code.
- **getSQLState()**  
Returns a string that describes the **SQLSTATE** value. The string follows the X/Open **SQLSTATE** conventions.

All Informix JDBC Driver errors have error codes of the form -79XXX, such as -79708 Method can't take null parameter.

For a list of Informix database server errors, refer to *Informix Error Messages*. You can find the on-line version of this guide at <http://www.informix.com/answers>.

The following example from the **SimpleSelect.java** program shows how to use the **SQLException** class to catch Informix JDBC Driver or database server errors using a try-catch block:

```
try
{
    PreparedStatement pstmt = conn.prepareStatement("Select * from x where a = ?;");
    pstmt.setInt(1, 11);
    ResultSet r = pstmt.executeQuery();

    while(r.next())
    {
        short i = r.getShort(1);
        System.out.println("Select: column a = " + i);
    }
    r.close();
    pstmt.close();
}
catch (SQLException e)
{
    System.out.println("ERROR: Fetch statement failed: " + e.getMessage());
}
```

## Retrieving Informix Error Message Text

Informix provides the class **com.informix.jdbc.Message** for retrieving the Informix error message text based on the Informix error number. To use this class, call the Java interpreter **java** directly, passing it an Informix error number, as shown in the following example:

```
java com.informix.jdbc.Message 100
```

The example returns the message text for Informix error 100:

```
100: ISAM error: duplicate value for a record with unique key.
```

A positive error number is returned if you specify an unsigned number when using the **com.informix.jdbc.Message** class. This differs from the **finderr** utility, which returns a negative error number for an unsigned number.



---

## Internationalization

Internationalization allows you to develop software independently of the countries or languages of its users and then to localize your software for multiple countries or regions. Informix JDBC Driver extends the Java JDK 1.2 internationalization features by providing access to Informix databases that are based on different locales and code sets.

### JDK 1.1 and 1.2 Internationalization Support

Versions 1.1 and 1.2 of the JDK provide a rich set of APIs for developing global applications. These internationalization APIs are based on the Unicode 2.0 code set and can adapt text, numbers, dates, currency, and user-defined objects to any country's conventions.

The internationalization APIs are concentrated in three packages:

- The **java.text** package contains classes and interfaces for handling text in a locale-sensitive way.
- The **java.io** package contains new classes for importing and exporting non-Unicode character data.
- The **java.util** package contains the **Locale** class, the localization support classes, and new classes for date and time handling.

For more information about JDK internationalization support, consult this Web site:

<http://java.sun.com/products/jdk/1.2/docs/guide/internat/intl.doc.html>



**Warning:** *There is no connection between JDK locales and JDK code sets: you must keep these in agreement. For example, if you select the Japanese locale **ja\_JP**, there is no Java method that tells you that the SJIS code set is the most appropriate.*

### Support for Informix GLS Variables

For detailed information about Informix global language support (GLS), refer to the *Informix Guide to GLS Functionality*.

Internationalization adds several new environment variables to Informix JDBC Driver:

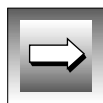
- **DB\_LOCALE**
- **CLIENT\_LOCALE**
- **GL\_DATE**
- **DBDATE**
- **DBCENTURY**

The locale environment variables are available on and optional for Informix servers that support GLS:

- The **DB\_LOCALE** variable specifies the locale of the database. Informix JDBC Driver uses this variable to perform code set conversion between Unicode and the database locale. The server uses **DB\_LOCALE** with **CLIENT\_LOCALE** to establish the server processing locale.
- The **CLIENT\_LOCALE** variable specifies the locale of the client that is accessing the database. Informix JDBC Driver uses this variable only to provide defaults for user-defined formats and to display error messages.

If set, the **CLIENT\_LOCALE** value establishes the server processing locale to provide defaults for user-defined formats like the **GL\_DATE** format, and user-defined types can use it for code-set conversion.

The **GL\_DATE**, **DBDATE**, and **DBCENTURY** variables are described in the following section.



**Important:** The **DB\_LOCALE**, **CLIENT\_LOCALE**, and **GL\_DATE** variables are supported only if the server supports the Informix GLS feature. If these environment variables are set and your application connects to a non-GLS server (server versions earlier than 7.2), a connection exception occurs. If you connect to a non-GLS server, the behavior is the same as for Informix JDBC Driver Version 1.22.

## Support for End-User Formats

The end-user format is the format in which a date appears in a string variable. This section describes the **GL\_DATE**, **DBDATE**, and **DBCENTURY** variables, which specify end-user formats.

***GL\_DATE Variable***

The **GL\_DATE** environment variable specifies the end-user formats of values in DATE columns. This variable is supported in Informix database servers 7.2x and beyond. A **GL\_DATE** format string can contain the following characters:

- One or more white-space characters
- An ordinary character (other than the % symbol or a white-space character)
- A formatting directive, which is composed of the % symbol followed by one or two conversion characters that specify the required replacement

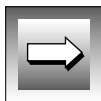
There are two types of formatting directives: non-era-based and era-based. The non-era-based time formatting directives are defined in the following table.

Directive	Replaced By
%a	The abbreviated weekday name as defined in the locale
%A	The full weekday name as defined in the locale
%b	The abbreviated month name as defined in the locale
%B	The full month name as defined in the locale
%C	The century number (the year divided by 100 and truncated to an integer) as a decimal number (00 through 99)
%d	The day of the month as a decimal number (01 through 31). A single digit is preceded by a zero (0)
%D	Same as the %m/%d/%y format
%e	The day of the month as a decimal number (1 through 31). A single digit is preceded by a space
%h	Same as the %b formatting directive
%iy	The year as a two-digit decade (00 through 99). It is the Informix-specific formatting directive for %y.

(1 of 2)

Directive	Replaced By
%iY	The year as a four-digit decade (0000 through 9999). It is the Informix-specific formatting directive for %Y.
%m	The month as a decimal number (01 through 12)
%n	A NEWLINE character
%t	The TAB character
%w	The weekday as a decimal number (0 through 6); 0 represents the locale equivalent of Sunday.
%x	A special date representation that the locale defines
%y	The year as a two-digit decade (00 through 99)
%Y	The year as a four-digit decade (0000 through 9999)
%%	% (to allow % in the format string)

(2 of 2)



**Important:** *GL\_DATE* optional date format qualifiers for field specifications (for example %4m to display a month as a decimal number with a maximum field width of 4) are not supported.

The *GL\_DATE* conversion modifier “O”, which indicates use of alternative digits for alternative date formats, is not supported.

If a date string contains an era-based date, for example, 1998-09-29 A.D., the `GL_DATE` environment variable must specify the era-based format of the value to scan or display. You can use the directives in the following table to indicate era-based date formatting, which the locale defines.

Directive	Description
<code>%EC</code>	Accepts the full or abbreviated era name for scanning; for printing, <code>%EC</code> is replaced by the full name of the base year (period) of the era that the locale defines (same as <code>%C</code> if the locale does not define an era).
<code>%Eg</code>	Accepts the full or abbreviated era name for scanning; for printing, <code>%Eg</code> is replaced by the abbreviated name of the base year (period) of the era that the locale defines (same as <code>%C</code> if the locale does not define an era).
<code>%Ex</code>	Replaced by a special date representation for an era that the locale defines (same as <code>%x</code> if the locale does not define an era).
<code>%Ey</code>	Replaced by the offset from <code>%EC</code> of the era the locale defines. This date is the era year only (same as <code>%y</code> if the locale does not define an era).
<code>%EY</code>	Replaced by the full era year, which the locale defines (same as <code>%Y</code> if the locale does not define an era).

White space or other nonalphanumeric characters must appear between any two formatting directives. If a `GL_DATE` variable format does not correspond to any of the valid formatting directives, errors can result when the server attempts to format the date.

For example, for a U.S. English locale, you can format an internal `DATE` value for 09/29/1998 using the following format:

```
Sep 29, 1998 (Tuesday)
```

To create this format, set the `GL_DATE` environment variable to this value:

```
%b %d, %Y (%A)
```

To insert this date value into a database table that has a `DATE` column, you can perform the following types of inserts:

- **Nonnative SQL (enter the date value exactly as expected by the `GL_DATE` setting).**

```
Statement stmt = conn.createStatement();
String query = "create table tablename (date_col " +
    "date)";
stmt.executeUpdate(query);
query = "insert into tablename values ('Sep 29, " +
    "1998 (Tuesday)')";
stmt.executeUpdate(query);
```

- **Native SQL (enter the date value in the JDBC escape format `yyyy-mm-dd`; the value is converted to the `GL_DATE` format automatically).**

```
Statement stmt = conn.createStatement();
String query = "create table tablename (date_col " +
    "date)";
stmt.executeUpdate(query);
query = "insert into tablename values " +
    "({d '1998-09-29'})";
stmt.executeUpdate(query);
```

To retrieve the formatted `GL_DATE` `DATE` value from the database, call the `getString()` method of the `ResultSet` class, as the following example shows:

```
ResultSet r = stmt.executeQuery("select date_col from " +
    "tablename");
while (r.next())
{
    String datestr = r.getString(1);
}
```

To enter strings that represent dates into database table columns of `CHAR`, `VARCHAR`, or `LVARCHAR`, you can also build date objects that represent the date string value. The date string value must be in `GL_DATE` format.

```
Statement stmt = conn.createStatement();
String query = "create table tablename (char_col char(30))";
stmt.executeUpdate(query);
query = "insert into tablename values ('Sep 29, 1998 " +
    "(Tuesday)')";
stmt.executeUpdate(query);
ResultSet r = stmt.executeQuery("select char_col from " +
    "tablename");
while (r.next())
{
    java.sql.Date dateval = r.getDate(1);
}
```

## ***DBDATE Variable***

The **DBDATE** environment variable specifies the end-user formats of values in DATE columns. End-user formats are used in the following ways:

- When you input DATE values, Informix products use the **DBDATE** environment variable to interpret the input. For example, if you specify a literal DATE value in an INSERT statement, Informix database servers require this literal value to be compatible with the format specified by the **DBDATE** variable.
- When you display DATE values, Informix products use the **DBDATE** environment variable to format the output.

With standard formats, you can specify the following attributes:

- The order of the month, day, and year in a date
- Whether the year is printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year

The format string can include the following characters:

- Hyphen ( - ), dot ( . ), and slash ( / ) are separator characters in a date format. A separator appears at the end of a format string (for example Y4MD-).
- A 0 indicates that no separator is displayed.
- D and M are characters that represent the day and the month.
- Y2 and Y4 are characters that represent the year and the number of digits in the year.

The following format strings are valid standard **DBDATE** formats:

- DMY2
- DMY4
- MDY4
- MDY2
- Y4MD
- Y4DM
- Y2MD
- Y2DM

The separator always goes at the end of the format string (for example, DMY2/). If no separator or an invalid character is specified, the slash (/) character is the default.

For the U.S. ASCII English locale, the default setting for **DBDATE** is Y4MD-, where Y4 represents a four-digit year, M represents the month, D represents the day, and hyphen (-) is the separator (for example, 1998-10-08).

For era-based dates such as EDMY2/, the E indicates a multibyte character that represents the era name (if a locale supports eras).

To insert a date value into a database table with a DATE column, you can perform the following types of inserts (the **DBDATE** value is MDY2-):

- **Nonnative SQL (enter the date value exactly as expected by the **DBDATE** setting).**

```
Statement stmt = conn.createStatement();
String query = "create table tablename (date_col " +
    "date)";
stmt.executeUpdate(query);
query = "insert into tablename values ('08-10-98')";
stmt.executeUpdate(query);
```

- **Native SQL (enter the date value in the JDBC escape format yyyy-mm-dd; the value is converted to the **DBDATE** format automatically).**

```
Statement stmt = conn.createStatement();
String query = "create table tablename (date_col " +
    "date)";
stmt.executeUpdate(query);
query = "insert into tablename values " +
    "({d '1998-08-10'})";
stmt.executeUpdate(query);
```

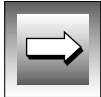
To retrieve the formatted **DBDATE** DATE value from the database, call the **getString** method of the **ResultSet** class. For example:

```
ResultSet r = stmt.executeQuery("select date_col from " +
    "tablename");
while (r.next())
{
    String datestr = r.getString(1);
}
```



To enter strings that represent dates into database table columns of CHAR, VARCHAR, or LVARCHAR, you can build date objects that represent the date string value. The date string value needs to be in **DBDATE** format. For example:

```
Statement stmt = conn.createStatement();
String query = "create table tablename (char_col char(8))";
stmt.executeUpdate(query);
query = "insert into tablename values ('08-10-98')";
stmt.executeUpdate(query);
ResultSet r = stmt.executeQuery("select char_col from " +
    "tablename");
while (r.next())
{
    java.sql.Date dateval = r.getDate(1);
}
```



**Important:** Informix JDBC Driver does not support ALS 6.0, 5.0, or 4.0 era-based formats in the **DBDATE** environment variable.

### ***DBCENTURY Variable***

The **DBCENTURY** environment variable enables you to choose the appropriate four-digit year expansion for DATE and DATETIME values that are defined using a one- or two-digit year. See the *Informix Guide to SQL: Reference* for detailed information about this environment variable.

## **Precedence Rules Regarding DATE Value End-User Formats**

The precedence rules that define how to determine an end-user format for an internal DATE value are listed here:

- If a **DBDATE** format is specified, this format is used.

- If a **GL\_DATE** format is specified, a locale must be determined:
  - If a **CLIENT\_LOCALE** value is specified, it is used in conjunction with the **GL\_DATE** format string to display DATE values.
  - If a **DB\_LOCALE** value is specified but a **CLIENT\_LOCALE** value is not, the **DB\_LOCALE** value is compared with the database locale (read from the **systables** table of the user database) to verify that the **DB\_LOCALE** value is valid. If the **DB\_LOCALE** value is valid, it is used in conjunction with the **GL\_DATE** format string to display DATE values. If the **DB\_LOCALE** value is not valid, the database locale is used in conjunction with the **GL\_DATE** format string.
  - If neither **CLIENT\_LOCALE** nor **DB\_LOCALE** values are specified, the database locale is used in conjunction with the **GL\_DATE** format string to display DATE values.
- If a **CLIENT\_LOCALE** value is specified, the DATE formats conform to the default formats associated with this locale.
- If a **DB\_LOCALE** value is specified but no **CLIENT\_LOCALE** value is specified, the **DB\_LOCALE** value is compared with the database locale to verify that the **DB\_LOCALE** value is valid. If the **DB\_LOCALE** value is valid, the **DB\_LOCALE** default formats are used. If the **DB\_LOCALE** value is not valid, the default formats for dates associated with the database locale are used.
- If neither **CLIENT\_LOCALE** nor **DB\_LOCALE** values are specified, all DATE values are formatted in U.S. English format, Y4MD-.

## Support for Code Set Conversion

Code-set conversion converts character data from one code set (the source code set) to another (the target code set). In a client/server environment, character data might need to be converted from one code set to another if the client and server computers use different code sets to represent the same characters. You must specify code set conversion for the following types of character data:

- SQL data types (CHAR, VARCHAR, NCHAR, NVARCHAR)
- SQL statements
- Database objects such as database names, column names, table names, statement identifier names, and cursor names

- Stored procedure text
- Command text
- Environment variables

Informix JDBC Driver converts character data as it is sent between client and server. The code set (encoding) used for the conversion is specified in the **systables** catalog for the opened database. With this feature, you can specify **DB\_LOCALE** and **CLIENT\_LOCALE** values in the connection properties or URL.

### ***Unicode to Database Code Set***

Java is Unicode based, so Informix JDBC Driver converts data between Unicode and the Informix database code set. The code set conversion value is extracted from the **DB\_LOCALE** value specified at the time the connection is made. If this **DB\_LOCALE** value is incorrect, the database locale (stored in the database **systables** catalog) is used in the connection and in the code set conversion.

The **DB\_LOCALE** value must be a valid Informix locale, with a valid Informix code set name or number as shown in the compatibility table that follows. The following table maps the supported JDK 1.2 encodings to Informix code sets.

Informix Code Set Name	Informix Code Set Number	JDK Code Set
8859-1	819	8859_1
8859-2	912	8859_2
8859-3	57346	8859_3
8859-4	57347	8859_4
8859-5	915	8859_5
8859-6	1089	8859_6
8859-7	813	8859_7
8859-8	916	8859_8
8859-9	920	8859_9

(1 of 2)

Informix Code Set Name	Informix Code Set Number	JDK Code Set
ASCII	364	ASCII
sjis-s	932	SJIS
utf8	57372	UTF8
big5	57352	Big5
CP1250	1250	Cp1250
CP1251	1251	Cp1251
CP1252	1252	Cp1252
CP1253	1253	Cp1253
CP1254	1254	Cp1254
CP1255	1255	Cp1255
CP1256	1256	Cp1256
CP1257	1257	Cp1257
cp949	57356	Cp949
KS5601	57356	Cp949
ksc	57356	Cp949
ujjis	57351	EUC_JP
gb	57357	ISO2022CN_GB
GB2312-80	57357	ISO2022CN_GB
cp936	57357	ISO2022CN_GB

(2 of 2)

You cannot use an Informix locale with a code set where there is no JDK supported encoding. This incorrect usage results in an `Encoding not supported` error message.

If the connection is made but the server returns a warning of a mismatch between the **DB\_LOCALE** value sent and the real value in the database **systables** catalog, the correct database locale is automatically extracted from the **systables** catalog and the client uses the correct JDK encoding for the connection.

The following table shows the supported locales.

Supported Locales				
ar_ae	ar_bh	ar_kw	ar_om	ar_qa
ar_sa	bg_bg	ca_es	cs_cz	da_dk
de_at	de_ch	de_de	el_gr	en_au
en_ca	en_gb	en_ie	en_nz	en_us
es_ar	es_bo	es_cl	es_co	es_cr
es_ec	es_es	es_gt	es_mx	es_pa
es_pe	es_py	es_sv	es_uy	es_ve
fi_fi	fr_be	fr_ca	fr_ch	fr_fr
hr_hr	hu_hu	is_is	it_ch	it_it
iw_il	ja_jp	ko_kr	mk_mk	nl_be
nl_nl	no_no	pl_pl	pt_br	pt_pt
ro_ro	ru_ru	sh_yu	sk_sk	sv_se
th_th	tr_tr	uk_ua	zh_cn	zh_tw

### ***Unicode to Client Code Set***

Because the Unicode code set includes all existing code sets, the Java virtual machine (JVM) must render the character using the platform's local code set. Inside the Java program, you must always use Unicode characters. The JVM on that platform converts input and output between Unicode and the local code set. For example, you specify button labels in Unicode, and the JVM converts the text to display the label correctly. Similarly, when the **getText()** method gets user input from a text box, the client program gets the string in Unicode, no matter how the user entered it.

Never read a text file one byte at a time. Always use the **InputStreamReader()** or **OutputStreamWriter()** methods to manipulate text files. By default, these methods use the local encoding, but you can specify an encoding in the constructor of the class, as follows:

```
InputStreamReader = new InputStreamReader (in, "SJIS");
```

You and the JVM are responsible for getting external input into the correct Java Unicode string. Thereafter, the database locale encoding is used to send the data to and from the server.

### ***Connecting to a Database with Non-ASCII Characters***

If you do not specify the database name at connection time, the connection must be opened with the correct **DB\_LOCALE** value for the specified database.

If **CLOSE DATABASE** and **DATABASE *dbname*** statements are issued, the connection continues to use the original **DB\_LOCALE** value to interpret the database name, so if the **DB\_LOCALE** value of the new database does not match, an error is returned. In this case, the client program must close and reopen the connection with the correct **DB\_LOCALE** value for the new database.

If you supply the database name at connection time, the **DB\_LOCALE** value must be set to the correct database locale.

## Code Set Conversion for TEXT Data Types

Informix JDBC Driver does not automatically convert code sets for TEXT, BYTE, CLOB, and BLOB data types. You can convert the code set for TEXT data by using the `getBytes()`, `getString()`, `InputStreamReader()`, and `OutputStreamWriter()` methods. These methods take a code set parameter that converts to and from Unicode and the specified code set. These methods are covered in detail in Sun's JDK documentation.

Here is sample code that shows how to convert a file from the client code set to Unicode and then from Unicode to the database code set. When you retrieve data from the database, you can use the same approach to convert the data from the database code set to the client code set.

```

...
// File in client encoding
File infile = new File("data.dat");
// New converted file -database encoding
File outfile = new File("data_conv.dat");
...

// Convert data from client encoding to database encoding
try
{
    String from = "SJIS";           // client encoding
    String to = "8859_1";         // database encoding
    convertFile(infile, outfile, from, to);
}
catch (Exception e)
{
    System.out.println("Failed to convert file");
}
...

// Get length of converted file
int fileLength = (int) outfile.length();
fin = new FileInputStream(outfile);
pstmt.setAsciiStream(1, fin, fileLength);
...

public static void convertFile(File infile, File outfile, String from,
    String to) throws IOException
{
    InputStream in = new FileInputStream(infile);
    OutputStream out = new FileOutputStream(outfile);

    Reader r = new BufferedReader(new InputStreamReader(in, from));
    Writer w = new BufferedWriter(new OutputStreamWriter(out, to));

    // Copy characters from input to output. The InputStreamReader converts
    // from the input encoding to Unicode, and the OutputStreamWriter

```

```
// converts from Unicode to the output encoding. Characters that can
// not be represented in the output encoding are output as '?'

char[] buffer = new char[4096];
int len;
while ((len = r.read(buffer)) != -1)
    w.write(buffer, 0, len);
r.close();
w.flush();
w.close();
}
```

---

## Handling Transactions

By default, all new **Connection** objects are in autocommit mode. This means that a **COMMIT** statement is automatically executed after each statement that is sent to the database server. To turn autocommit mode off for a connection, explicitly call the **Connection.setAutoCommit(false)** method.

When autocommit mode is off, Informix JDBC Driver implicitly starts a new transaction when the next statement is sent to the database server. This transaction lasts until the user issues a **COMMIT** or **ROLLBACK** statement. If the user has already started a transaction by executing **setAutoCommit(false)**, and then calls **setAutoCommit(false)** again, the existing transaction continues unchanged. The Java program must explicitly terminate the transaction by issuing either a **COMMIT** or a **ROLLBACK** statement before it drops the connection to the database or the database server.

If the Java program sets autocommit mode on while inside a transaction, Informix JDBC Driver rolls back the current transaction before it actually turns autocommit mode on.

In a database that has been created with logging, if a **COMMIT** statement is sent to the database server (either with the **Connection.commit()** method or directly with an SQL statement) and autocommit mode is on, the error **-255 : Not in transaction** is returned by the database server because there is currently no user transaction started.

In a database created in ANSI mode, explicitly sending a **COMMIT** statement to the database server commits an empty transaction. No error is returned because the database server automatically starts a transaction before it executes the statement if there is no user transaction currently open.



---

## Other Informix Extensions to the JDBC API

This section describes the Informix-specific extensions to the JDBC API not already covered in this guide. These extensions handle information that is specific to Informix databases.

Another Informix extension, the `com.informix.jdbc.Message` class, is fully described in “Handling Errors” on page 2-30.

### Using the Informix SERIAL and SERIAL8 Data Types

Informix JDBC Driver provides support for the Informix SERIAL and SERIAL8 data types via the methods `getSerial()` and `getSerial8()`, which are part of the implementation of the `java.sql.Statement` interface.

Since the SERIAL and SERIAL8 data types do not have an obvious mapping to any JDBC API data types from the `java.sql.Types` class, you must import Informix-specific classes into your Java program to be able to handle SERIAL and SERIAL8 table columns. To do this, add the following import line to your Java program:

```
import com.informix.jdbc.*
```

Use the `getSerial()` and `getSerial8()` methods after an INSERT statement to return the serial value that was automatically inserted into the SERIAL or SERIAL8 column of a table, respectively. The methods return 0 if any of the following conditions are true:

- The last statement was not an INSERT statement.
- The table being inserted into does not contain a SERIAL or SERIAL8 column.
- The INSERT statement has not executed yet.

If you execute the `getSerial()` or `getSerial8()` method after a CREATE TABLE statement, the methods return 1 by default (assuming the new table includes a SERIAL or SERIAL8 column). If the table does not contain a SERIAL or SERIAL8 column, the methods return 0. If you assign a new serial starting number, the methods return that number.

If you want to use the `getSerial()` and `getSerial8()` methods, you must cast the **Statement** or **PreparedStatement** object to **IfxStatement**, the Informix-specific implementation of the **Statement** interface. The following example shows how to perform the cast:

```
PreparedStatement pstmt = conn.prepareStatement ("INSERT INTO ... ");
pstmt.executeUpdate();
int serial_number = ((IfxStatement)pstmt).getSerial();
```

If you want to insert consecutive serial values into a column of data type SERIAL or SERIAL8, specify a value of 0 for the SERIAL or SERIAL8 column in the INSERT statement. When the column is set to 0, the database server assigns the next highest value.

For more detailed information about the Informix SERIAL and SERIAL8 data types, refer to *Informix Guide to SQL: Reference* and *Informix Guide to SQL: Syntax*. You can find the on-line version of both of these guides at <http://www.informix.com/answers>.

## Obtaining Driver Version Information

There are two ways to obtain version information about Informix JDBC Driver: from your Java program, or from the UNIX or Windows command line. The command line method also allows you to obtain the serial number you provided when you installed the driver on your computer.

### To get version information from your Java program

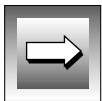
1. Import the Informix package `com.informix.jdbc.*` into your Java program by adding the following line to the import section:

```
import com.informix.jdbc.*;
```
2. Invoke the static method `IfxDriver.getJDBCVersion()`. This method returns a **String** object that contains the complete version of the current Informix JDBC Driver.

An example of a version of Informix JDBC Driver is 1.22.JC1.

The `IfxDriver.getJDBCVersion()` method does not return the serial number you provided during installation of the driver.

**Important:** For Version X.Y of Informix JDBC Driver, the JDBC API methods `Driver.getMajorVersion()` and `DatabaseMetaData.getDriverMajorVersion()` always return the value X. Similarly, the methods `Driver.getMinorVersion()` and `DatabaseMetaData.getDriverMinorVersion()` always return the value Y.



To get the version of Informix JDBC Driver from the command line, enter the following command at the UNIX shell prompt or the Windows command prompt:

```
java com.informix.jdbc.Version
```

The command also returns the serial number you provided when you installed the driver. An example of a serial number is INF#J000000.

---

## Using an HTTP Proxy Server

You might need to use an HTTP proxy server for these reasons:

- **Applets.** Because of security restrictions in Web browsers, if an applet is using Informix JDBC Driver, it can only connect to a database running on the same host as the Web server. This restriction is not always desirable, because both Web servers and database servers can require extensive system resources.
- **Firewalls.** Informix JDBC Driver cannot connect to a database from behind a firewall. The firewall prevents the browser from connecting to the database.

The solution to both these problems is to install Informix JDBC Driver on the same computer as the Java applet and install the HTTP proxy as a middle tier between the Java applet and Informix database machines.

The HTTP proxy feature is not part of the JDBC 2.0 specification. The HTTP proxy is a lightweight servlet that receives HTTP packets from the JDBC client. The proxy extracts SQL requests from the packet and transmits them to a database server. The client (the end user) is unaware of this extra layer.

To specify a proxy, amend the URL statement:

```
CURRENT_JDBC_URL;proxy=web-server-host-name:port-number
```

Before creating the proxy servlet, you must copy two class files, **IfxJDBCProxy.class** and **SessionMgr.class**, to the servlet directory. These two class files reside in the top-level directory **proxy** after the product bundle is installed.

The Web server must support servlets. (For example, the Sun and Apache Java Web servers do.) Add the Web servlet to the Web server by specifying the class of the proxy servlet.

#### To add a proxy servlet to a Java Web server

1. Go to the URL `http://server-host-name:port-number` using a Web browser. The Java Web server administrator page appears.
2. Enter `admin` as the user followed by the administrator password to log on to the Web server administrator management menu.
3. Select **Web Service**→**Servlets**→**Add** from the menu.
4. Specify the JDBC servlet name and class. Name the servlet for the JDBC proxy **IfxJDBCProxy**.

This procedure makes the Web server aware of the proxy servlet. The servlet must allow for some configuration, such as the ability to specify initialization parameters. Example parameters are maximum number of connections, idle time-out, possibly a default port and host, and even a properties file.

If the proxy servlet is invoked from an applet, the applet loads the servlet (if the servlet is not loaded) when the applet sends a message to the servlet.

The following Web sites offer more information about proxy servlets:

- <http://jserv.javasoft.com/index.html>
- <http://www.javasoft.com>
- <http://www.sun.com/java>
- <http://java.apache.org>

Other ways to use Informix JDBC Driver in a multiple-tier environment are as follows:

- **Remote method invocation (RMI).** Informix JDBC Driver resides on an application server that is a middle tier between the Java applet or application and Informix database machines. An example of RMI is included with Informix JDBC Driver; see Appendix A, “Sample Code Files” for details.
- **Other communication protocols, such as CORBA.** Informix JDBC Driver resides on an application server that is a middle tier between the Java applet or application and Informix database machines.

---

## Restrictions and Limitations

The following JDBC API methods are not supported by Informix JDBC Driver and cannot be used in a Java program that connects to an Informix database:

- **Connection.setCatalog()**
- **Connection.setReadOnly()**
- **Connection.isReadOnly()**
- **Statement.setMaxFieldSize()**
- **Statement.setQueryTimeout()**
- **Statement.cancel()**
- **PreparedStatement.setUnicodeStream()**
- **CallableStatement.registerOutParameter()**
- **ResultSet.getUnicodeStream()**

The following JDBC API methods behave differently than specified by the JavaSoft specification:

- **Statement.execute()**  
Method returns a single result set.
- **PreparedStatement.execute()**  
Method returns a single result set.
- **CallableStatement.execute()**  
Method returns a single result set.
- **ResultSetMetaData.getTableName()**  
Method always returns the value "".
- **ResultSetMetaData.getSchemaName()**  
Method always returns the value "".
- **ResultSetMetaData.getCatalogName()**  
Method always returns the value "".
- **ResultSetMetaData.isReadOnly()**  
Method always returns FALSE.

- **ResultSetMetaData.isWriteable()**  
Method always returns `TRUE`.
- **ResultSetMetaData.isDefinitelyWriteable()**  
Method always returns `TRUE`.

---

# Troubleshooting

In This Chapter . . . . .	3-3
Debugging Your JDBC API Program. . . . .	3-3
Using the Debug Version of the Driver. . . . .	3-3
Turning on Tracing . . . . .	3-4
Performance Issues . . . . .	3-5
Using the FET_BUF_SIZE Environment Variable . . . . .	3-6
Memory Management of Large Objects . . . . .	3-6
Reducing Network Traffic . . . . .	3-8





## In This Chapter

This chapter provides troubleshooting tips for Informix JDBC Driver. It covers the following topics:

- “Debugging Your JDBC API Program”
- “Performance Issues”

---

## Debugging Your JDBC API Program

If your Java program contains JDBC API programming errors, you might want to use the debug version of Informix JDBC Driver instead of the optimized version to try to find where the errors occur in your program.

### Using the Debug Version of the Driver

The debug version of Informix JDBC Driver, called **ifxjdbc-g.jar**, is exactly the same as the optimized version (called **ifxjdbc.jar**), except that it has been compiled with the **-g** option instead of the **-O** option.

The difference in compilation options also means that the debug version of Informix JDBC Driver has been embedded with tracing information. When you use the debug version of the driver, you can turn on tracing and get a better idea of what the JDBC API portion of your Java program is actually doing.

For instructions on how to use the debug version of Informix JDBC Driver in a Java application or Java applet, refer to “Using the Driver in an Application” on page 1-13 or “Using the Driver in an Applet” on page 1-14, respectively.

## Turning on Tracing

Trace output consists of the following two kinds of information:

- General information from Informix JDBC Driver
- Informix native SQLI protocol messages sent between your Java program and the Informix database server

To turn on tracing, specify the environment variables **TRACE**, **TRACEFILE**, **PROTOCOLTRACE**, and **PROTOCOLTRACEFILE** in the database URL or the property list when you establish a connection to an Informix database or database server.

The following table describes the tracing environment variables.

Environment Variable	Description
TRACE	Traces general information from Informix JDBC Driver. Can be set to one of the following levels: <ul style="list-style-type: none"><li>■ 0. Tracing not enabled. This is the default value.</li><li>■ 1. Traces the entry and exit points of methods.</li><li>■ 2. Same as Level 1, except generic error messages are also traced.</li><li>■ 3. Same as Level 2, except data variables are also traced.</li></ul>
TRACEFILE	Specifies the full pathname of the operating system file on the client computer to which the <b>TRACE</b> messages are written.

(1 of 2)

Environment Variable	Description
PROTOCOLTRACE	<p>Traces the SQLI protocol messages sent between your Java program and the Informix database server.</p> <p>Can be set to the following levels:</p> <ul style="list-style-type: none"> <li>■ 0. Protocol tracing not enabled. This is the default value.</li> <li>■ 1. Traces message IDs.</li> <li>■ 2. Same as Level 1, except the data in the message packets is also traced.</li> </ul>
PROTOCOLTRACFILE	<p>Specifies the full pathname of the operating system file on the client computer to which the <b>PROTOCOLTRACE</b> messages are written.</p>

(2 of 2)

The following example of a database URL specifies the highest level of protocol tracing and sets tracing output to the operating system file **/tmp/trace.out**:

```
String url = "jdbc:informix-
sql://123.45.67.89:1533:informixserver=myserver;user=rdtest;password=test;
PROTOCOLTRACE=2;PROTOCOLTRACFILE=/tmp/trace.out";
```

For more information on establishing a connection to an Informix database or database server using a database URL or a property list, refer to “Establishing a Connection” on page 2-3.

## Performance Issues

This section describes issues that might affect the performance of your queries: the **FET\_BUF\_SIZE** environment variable, memory management of the Informix TEXT and BYTE data types, memory management of the Informix BLOB and CLOB data types, and reducing network traffic.

## Using the FET\_BUF\_SIZE Environment Variable

When a SELECT statement is sent from a Java program to an Informix database, the returned rows, or tuples, are stored in a tuple buffer in Informix JDBC Driver. The default size of the tuple buffer is the larger of the returned tuple size or 4096 bytes.

You can use the Informix **FET\_BUF\_SIZE** environment variable to override the default size of the tuple buffer. Increasing the size of the tuple buffer can reduce network traffic between your Java program and the database, often resulting in better performance of queries.

**FET\_BUF\_SIZE** can be set to any positive integer less than or equal to 32,767. If the **FET\_BUF\_SIZE** environment variable is set, and its value is larger than the default tuple buffer size, the tuple buffer size is set to the value of **FET\_BUF\_SIZE**.

There are times, however, when increasing the size of the tuple buffer can actually degrade the performance of queries. This could happen if your Java program has many active connections to a database or if the swap space on your computer is limited. If this is true for your Java program or computer, you might not want to use the **FET\_BUF\_SIZE** environment variable to increase the size of the tuple buffer.

For more information on setting Informix environment variables, see “Establishing a Connection” on page 2-3.

## Memory Management of Large Objects

Whenever a large object (a BYTE, TEXT, BLOB, or CLOB data type) is fetched from the server, the data is either cached into memory or stored in a temporary file (if it exceeds the memory buffer). A JDBC applet can cause a security violation if it tries to create a temporary file on the local computer. In this case, the entire large object must be stored in memory.

You can specify how large object data is stored using a user-defined environment variable, **LOBCACHE**, that you include as one of the name-value pairs in the URL syntax:

- To set the maximum number of bytes allocated in memory to hold the data, set the **LOBCACHE** value to that number of bytes. If the data size exceeds the **LOBCACHE** value, the data is stored in a temporary file. If a security violation occurs during creation of this file, the data is stored in memory.
- To always store the data in a file, set the **LOBCACHE** value to 0. In this case, if a security violation occurs, Informix JDBC Driver makes no attempt to store the data in memory.
- To always store the data in memory, set the **LOBCACHE** value to a negative number. If the required amount of memory is not available, Informix JDBC Driver throws the **SQLException** message `Out of Memory`.

If the **LOBCACHE** size is invalid or not defined, the default size is 4096.

You can set the **LOBCACHE** value through the URL, as follows:

```
URL = "jdbc:158.58.9.37:711test:user=guest;password=iamaguest;  
informixserver=oltapsh;lobcache=4096";
```

The preceding example stores the large object in memory if the size is 4096 bytes or fewer. If the large object exceeds 4096 bytes, Informix JDBC Driver tries to create a temporary file. If a security violation occurs, memory is allocated for the entire large object. If that fails, the driver throws an **SQLException** message.

Here is another example:

```
URL = "jdbc:informix-sqli://icarus:7110/testdb:user=guest:passwd=whoknows;  
informixserver=olserv01;lobcache=0";
```

The preceding example uses a temporary file for storing the fetched large object.

Here is a third example:

```
URL = "jdbc:informix-sqli://icarus:7110/testdb:user=guest:passwd=whoknows;  
informixserver=olserv01;lobcache=-1";
```

The preceding example always uses memory to store the fetched large object.

For programming information on how to use the **TEXT**, **BYTE**, **CLOB**, and **BLOB** data types in a Java program, refer to “Manipulating Informix Large Object Data Types” on page 2-15.

## Reducing Network Traffic

The two environment variables **OPTOFC** and **IFX\_AUTOFREE** can be used to reduce network traffic when you close **Statement** and **ResultSet** objects.

Set **OPTOFC** to 1 to specify that the **ResultSet.close()** method does not require a network round-trip if all the qualifying rows have already been retrieved in the client's tuple buffer. The database server automatically closes the cursor after all the rows have been retrieved.

Informix JDBC Driver might or might not have additional rows in the client's tuple buffer before the next **ResultSet.next()** method is called. Therefore, unless Informix JDBC Driver has received all rows from the database server, the **ResultSet.close()** method might still require a network round-trip when **OPTOFC** is set to 1.

Set **IFX\_AUTOFREE** to 1 to specify that the **Statement.close()** method does not require a network round-trip to free the server cursor resources if the cursor has already been closed in the database server.

The database server automatically frees the cursor resources right after the cursor is closed, either explicitly by the **ResultSet.close()** method or implicitly by the **OPTOFC** environment variable.

When the cursor resources have been freed, the cursor can no longer be referenced.

For examples of how to use the **OPTOFC** and **IFX\_AUTOFREE** environment variables, see the **autofree.java** and **optofc.java** demonstration examples. In these examples, the variables are set with the **Properties.put()** method.

For more information on setting Informix environment variables, refer to "Establishing a Connection" on page 2-3.

# Sample Code Files

## UNIX

This appendix lists and describes the code examples provided with Informix JDBC Driver.

The main examples are located in the following directories:

- ***\$JDBCLOCATION/demo/basic***
- ***\$JDBCLOCATION/demo/clob-blob***
- ***\$JDBCLOCATION/demo/udt-distinct***

*JDBCLOCATION* refers to the directory where you installed Informix JDBC Driver.

Another set of examples is located in the directory ***\$JDBCLOCATION/demo/stores7***. A README file in the **demo** directory explains the various demonstration files and how to execute them.

An RMI example is located in the directory ***\$JDBCLOCATION/demo/rmi***. A README file in the **demo** directory explains how to execute the example. ♦

## Windows

The main examples are located in the following directories:

- ***%JDBCLOCATION%\demo\basic***
- ***%JDBCLOCATION%\demo\clob-blob***
- ***%JDBCLOCATION%\demo\udt-distinct***

*JDBCLOCATION* refers to the directory where you installed Informix JDBC Driver.

Another set of examples is located in the directory `%JDBCLOCATION%\demo\stores7`. A README file in the **demo** directory explains the various demonstration files and how to execute them.

An RMI example is located in the directory `%JDBCLOCATION%\demo\rmi`. A README file in the **demo** directory explains how to execute the example. ♦

The following table lists the files in the **basic** directory. For each code example, the table displays the name of the Java program and a brief description of what the program does.

Demo Program Name	Description
autofree.java	Shows how to use the <b>AUTOFREE</b> environment variable.
ByteType.java	Shows how to insert into and select from a table that contains a column of data type <b>BYTE</b> .
CreateDB.java	Creates a database called <b>testDB</b> .
DBCENTURYSelect.java	Shows how to retrieve a date string representation that has a four-digit year expansion based on the <b>DBCENTURY</b> property value from the URL string.
DBConnection.java	Creates connections to both a database and a database server.
DBDATESelect.java	Shows how to retrieve a date object and a date string representation from the database based on the <b>DBDATE</b> property value from the URL string.
DBMetaData.java	Shows how to retrieve information about a database with the <b>DatabaseMetaData</b> interface.
DropDB.java	Drops a database called <b>testDB</b> .
GLDATESelect.java	Shows how to retrieve a date object and a date string representation from the database based on the <b>GL_DATE</b> property value from the URL string.
Interval.java	Shows how to insert and select Informix <b>INTERVAL</b> data.
LOCALESelect.java	Shows how to retrieve a date object and a date string representation from the database based on the <b>CLIENT_LOCALE</b> property value from the URL string.
MultiRowCall.java	Shows how to return multiple rows in a stored procedure call.
OptimizedSelect.java	Shows how to use the <b>FET_BUF_SIZE</b> environment variable to adjust the Informix JDBC Driver tuple buffer size.

(1 of 2)



Demo Program Name	Description
optofc.java	Shows how to use the <b>OPTOFC</b> environment variable.
PropertyConnection.java	Shows how to specify connection environment variables via a property list.
RSMetaData.java	Shows how to retrieve information about a result set with the <b>ResultSet-MetaData</b> interface.
SimpleCall.java	Shows how to call a stored procedure.
SimpleConnection.java	Shows how to connect to a database or database server.
SimpleSelect.java	Shows how to send a simple SELECT query to the database server.
TextType.java	Shows how to insert into and select from a table that contains a column of data type TEXT.

(2 of 2)

The following table lists the files in the **clob-blob** directory. For each code example, the table displays the name of the Java program and a brief description of what the program does.

Demo Program Name	Description
demo1.java	Shows how to create two tables with BLOB and CLOB columns and compare the data.
demo2.java	Shows how to create one table with BYTE and TEXT columns and a second table with BLOB and CLOB columns and how to compare the data.
demo3.java	Shows how to create one table with BLOB and CLOB columns and a second table with BYTE and TEXT columns and how to compare the data.
demo4.java	Shows how to create two tables with BYTE and TEXT columns and compare the data.
demo5.java	Shows how to store data from a file into a BLOB table column.

The following table lists the files in the **udt-distinct** directory. For each code example, the table displays the name of the Java program and a brief description of what the program does.

Demo Program Name	Description
createDB.java	Creates a database that the other <b>udt-distinct</b> demonstration files use.
createTypes.java	Shows how to create opaque and distinct types in the database.
distinct_d1.java	Shows how to create a distinct type.
dropDB.java	Drops the database that the other <b>udt-distinct</b> demonstration files use.
udt_d1.java	Shows how to create a fixed-length opaque type.

---

# Glossary

- applet** A program created with Java classes that is not intended to be run on its own, but rather to be embedded in another application, such as a browser.
- autocommit mode** Mode in which a COMMIT statement is automatically executed after each statement sent to the database server.
- BLOB** A smart large object data type that stores any kind of binary data, including images. The database server performs no interpretation on the contents of a BLOB column.
- See also *smart large object*.
- blobpage** The unit of disk allocation within a blobspace. The size of a blobpage is determined by the DBA and can vary from blobpage to blobpage.
- blobspace** A logical collection of chunks that is used to store TEXT and BYTE data.
- built-in data type** A fundamental data type defined by the database server: for example, INTEGER, CHAR, or SERIAL8.
- BYTE** A built-in data type for a simple large object that stores any type of binary data and can be as large as  $2^{31}$  bytes.
- cast** A mechanism that the database server uses to convert data from one data type to another. The server provides built-in casts that it performs automatically. Users can create both implicit and explicit casts.
- See also *explicit cast*, *implicit cast*.

<b>cast function</b>	A function that is used to implement an implicit or explicit cast. A cast function performs the necessary operations for conversion between two data types. It must be registered as a cast with the CREATE CAST statement before it can be used.
<b>CLASSPATH</b>	An environment variable that tells the Java virtual machine (JVM) and other applications where to find the Java class libraries used in a Java program.
<b>CLOB</b>	A smart large object data type that stores blocks of text items, such as ASCII or PostScript files.  See also <i>smart large object</i> .
<b>code set</b>	A character set of one or more natural-language alphabets with symbols for digits, punctuation, and diacritical marks. Each character set has at least one code set, which maps its characters to unique bit patterns. ASCII, ISO8559-1, Microsoft 1252, and EBCDIC are examples of code sets for the English language.
<b>concurrency</b>	The ability of two or more processes to access the same database simultaneously.
<b>connection</b>	An association between an application and a database environment, created by a CONNECT or DATABASE statement. Database servers can also have connections to one another.  See also <i>explicit connection, implicit connection</i> .
<b>CORBA</b>	(Common Object Request Broker Architecture) The CORBA 2.0 specification describes a convention called Object Request Broker (ORB), the infrastructure for distributed-object computing. CORBA enables client applications to communicate with remote objects and invoke operations statically or dynamically.
<b>database URL</b>	URL passed to the <b>DriverManager.getConnection()</b> method that specifies the subprotocol (the database connectivity mechanism), the database or database server identifier, and a list of properties that can include Informix environment variables.
<b>data type</b>	See <i>built-in data type, extended data type</i> .

<b>dbspace</b>	A logical collection of one or more chunks of contiguous disk space within which you store databases and tables. Because chunks represent specific regions of disk space, the creators of databases and tables can control where their data is physically located by placing databases or tables in specific dbspaces. Large objects are stored in sbspaces.
<b>delimiter</b>	The boundary of an input field, or the terminator for a database column or row. Some files and prepared objects require a semicolon (;), comma (,), pipe ( ), space, or tab delimiters between statements.
<b>distinct data type</b>	A data type that is created with the CREATE DISTINCT TYPE statement. A distinct data type is based on an existing opaque, built-in, or distinct data type, known as its source type. The distinct data type has the same internal storage representation as its source type, but it has a different name. To compare a distinct data type with its source type requires an explicit cast. A distinct data type inherits all routines that are defined on its source type.
<b>explicit cast</b>	A cast that requires a user to specify the CAST AS keyword or cast operator (::) to convert data from one data type to another. An explicit cast requires a function if the internal storage representations of the two data types are not equivalent.
<b>explicit connection</b>	A connection made to a database environment that uses the CONNECT statement.  <i>See also <a href="#">implicit connection</a>.</i>
<b>extended data type</b>	A term used to refer to data types that are not built-in, such as opaque data types and distinct data types.
<b>fundamental data type</b>	A data type that cannot be broken into smaller pieces by the database server using SQL statements: for example, built-in data types and opaque data types.
<b>Global Language Support (GLS)</b>	An application environment that allows Informix application-programming interfaces (APIs) and database servers to handle different languages, cultural conventions, and code sets. Developers use the GLS libraries to manage all string, currency, date, and time data types in their code. Using GLS, you can add support for a new language, character set, and encoding by editing resource files, without access to the original source code, and without rebuilding the client software.

<b>host variable</b>	A C or COBOL program variable that is referenced in an embedded statement. A host variable is identified by the dollar sign ( \$ ) or colon ( : ) that precedes it.
<b>implicit cast</b>	A cast that the database server automatically performs to convert data from one data type to another.  See also <i>explicit cast</i> .
<b>implicit connection</b>	A connection made using a database statement (DATABASE, CREATE DATABASE, START DATABASE, DROP DATABASE).  See also <i>explicit connection</i> .
<b>IP address</b>	Unique ID of every computer on the Internet. The format consists of four numerical strings separated by dots, such as 123.45.67.89.
<b>jar utility</b>	A JavaSoft utility that creates Java archive, or JAR, files. JAR is a platform-independent file format that aggregates many files into one.
<b>keyword</b>	A word that has meaning to a programming language. In Informix SQL, keywords are shown in syntax diagrams in all uppercase letters. They must be used in SQL statements exactly as shown in the syntax although they can be in either uppercase or lowercase letters.
<b>large object</b>	A data object that exceeds 255 bytes in length. A large object is logically stored in a table column but physically stored independently of the column, because of its size. Large objects can contain non-ASCII data. The Universal Data Option recognizes two kinds of large objects: simple large objects (TEXT, BYTE) and smart large objects (CLOB, BLOB).  See also <i>simple large object</i> , <i>smart large object</i> .
<b>locale</b>	A set of files that define the native-language behavior of the program at run-time. The rules are usually based on the linguistic customs of the region or the territory. The locale can be set through an environment variable that dictates output formats for numbers, currency symbols, dates, and time as well as collation order for character strings and regular expressions.  See also <i>Global Language Support (GLS)</i> .
<b>LVARCHAR</b>	A built-in data type that stores varying-length character data greater than 256 bytes. It is used for input and output casts for opaque data types. LVARCHAR supports code-set order for comparisons of character data.

<b>metadata</b>	Data about data. Metadata provides information about data in the database or used in the application. Metadata can be data attributes, such as name, size, and data type, or descriptive information about data.
<b>opaque data type</b>	A fundamental data type of a predefined fixed or variable length whose internal structure is not accessible through SQL statements.. Opaque data types are created with the SQL statement CREATE OPAQUE TYPE. Support functions must always be defined for opaque types.
<b>RMI</b>	(Remote Method Invocation) A method for creating distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts.
<b>servlet</b>	An extension method for many common protocols, especially HTTP. Servlets are modules that run inside request/response oriented servers. Servlets are similar to applets in that their classes might be dynamically loaded, either across the network or from local storage. However, servlets differ from applets in that they lack a graphical interface.
<b>simple large object</b>	A large object that is stored in a blob space, is not recoverable, and does not obey transaction isolation modes. Simple large objects include TEXT and BYTE data types.  See also <i>TEXT</i> , <i>BYTE</i> .
<b>smart large object</b>	A large object that: <ul style="list-style-type: none"> <li>■ is stored in an sbspace, a logical storage area that contains one or more chunks.</li> <li>■ has read, write, and seek properties similar to a UNIX file.</li> <li>■ is recoverable.</li> <li>■ obeys transaction isolation modes.</li> <li>■ can be retrieved in segments by an application.</li> </ul> Smart large objects include CLOB and BLOB data types.
<b>SQLSTATE</b>	A variable that contains status values about the outcome of SQL statements.
<b>support functions</b>	The functions that the database server automatically invokes to process a data type.

The database server uses a support function to perform operations (such as converting to and from the internal, external, and binary representations of the type) on opaque data types.

An index access method uses a support function in an operator class to perform operations (such as building or searching) on an index.

<b>sysmaster database</b>	A master database created and maintained by every Informix database server. The <b>sysmaster</b> database contains the ON-Archive catalog tables and system monitoring interface (SMI) tables. Informix recommends you do not modify this database.
<b>system catalog</b>	A group of database tables that contain information about the database itself, such as the names of tables or columns in the database, the number of rows in a table, the information about indexes and database privileges, and so on.
<b>system-defined cast</b>	A cast that is built into the database server. A system-defined cast performs automatic conversions between different built-in data types.
<b>TEXT</b>	A built-in data type for a simple large object that stores text data and can be as large as $2^{31}$ bytes.
<b>tuple buffer</b>	The section of Informix JDBC Driver memory that stores the retrieved rows from a SELECT statement.



---

# Error Messages

- 79700    Method not supported  
Informix JDBC Driver does not support this JDBC method.
- 79702    Can't create new object  
The software could not allocate memory for a new **String** object.
- 79703    Row/column index out of range  
The row or column index is out of range. Compare the index to the number of rows and columns expected from the query to ensure that it is within range.
- 79704    Can't load driver  
Informix JDBC Driver could not create an instance of itself and register it in the **DriverManager** class. The rest of the **SQLException** text describes what failed.
- 79705    Incorrect URL format  
The URL you have submitted is invalid. Informix JDBC Driver does not recognize the syntax. Check the syntax and try again.
- 79708    Can't take null input  
The string you have provided is null. Informix JDBC Driver does not understand null input in this case. Check the input string to ensure that it has the proper value.

- 79709      Error in date format
- The expected input is a valid date string in the following format: *yyyy-mm-dd*. Check the date and verify that it has a four-digit year, followed by a valid two-digit month and two-digit day. The delimiter must be a hyphen ( - ).
- 79710      Syntax error in SQL escape clause
- Invalid syntax was passed to a JDBC escape clause. Valid JDBC escape clause syntax is demarcated by curly braces and a keyword, for example: *{keyword syntax}*. Check the JDBC 2.0 documentation from Sun Microsystems for a list of valid escape clause keywords and syntax.
- 79711      Error in time format
- An invalid time format was passed to a JDBC escape clause. The escape clause syntax for time literals has the following format: *{t 'hh:mm:ss'}*.
- 79712      Error in timestamp format
- An invalid timestamp format was passed to a JDBC escape clause. The escape clause syntax for timestamp literals has the following format: *{ts 'yyyy-mm-dd hh:mm:ss.f...'}.*
- 79713      Incorrect number of arguments
- An incorrect number of arguments was passed to the scalar function escape syntax. Here is the correct syntax: *{fn function(arguments)}*. Verify that the correct number of arguments was passed to the function.
- 79714      Type not supported
- You have specified a data type that is not supported by Informix JDBC Driver. Check your program to make sure the data type used is among those supported by the driver.
- 79715      Syntax error
- Invalid syntax was passed to a JDBC escape clause. Valid JDBC escape clause syntax is demarcated by curly braces and a keyword: *{keyword syntax}*. Check the JDBC 2.0 documentation from Sun Microsystems for a list of valid escape clause keywords and syntax.

- 79716 System or internal error
- An operating or runtime system error or a driver internal error occurred. The accompanying message describes the problem.
- 79726 Null SQL statement
- The SQL statement passed in was null. Check the SQL statement string of your program to make sure it contains a valid statement.
- 79727 Statement was not prepared
- The SQL statement was not prepared properly. If you use host variables (for example, `insert into mytab values (?, ?);`) in your SQL statement, you must use **connection.prepareStatement()** to prepare the SQL statement before you can execute it.
- 79729 Method cannot take argument
- The method does not take an argument. Refer to your Java API specification or the appropriate section of this guide to make sure you are using the method properly.
- 79730 Connection not established
- A connection was not established. You must obtain the connection by calling the **DriverManager.getConnection()** method first.
- 79731 MaxRows out of range
- You have specified an out-of-range **maxRow** value. Make sure you specify a value between 0 and **Integer.MAX\_VALUE**.
- 79732 Illegal cursor name
- The cursor name specified is illegal. Make sure the string passed in is not null or empty.
- 79733 No active result
- The statement does not contain an active result. Check your program logic to make sure you have called the **executeXXX()** method before you attempt to refer to the result.

- 79734    **INFORMIXSERVER** has to be specified
- INFORMIXSERVER** is a property required for connecting to an Informix database. You can specify it in the URL or as part of a **Properties** object that is passed to the **connect()** method.
- 79735    Can't instantiate protocol
- An internal error occurred during a connection attempt. Call Informix Technical Support.
- 79736    No connection/statement establish yet
- There is no current connection or statement. Check your program to make sure a connection was properly established or a statement was created.
- 79737    No meta data
- There is no metadata available for this SQL statement. Make sure the statement generates a result set before you attempt to use it.
- 79738    No such column name
- The column name specified does not exist. Make sure the column name is correct.
- 79739    No current row
- The cursor is not properly positioned. You must first position the cursor within the result set by using a method such as **resultset.next()**, **resultset.beforefirst()**, **resultset.first()**, or **resultset.absolute()**.
- 79740    No statement created
- There is no current statement. Make sure the statement was properly created.
- 79741    Can't convert to
- There is no data conversion possible from the column data type to the one specified. The actual data type is appended to the end of this message. Review your program logic to make sure that the conversion you have asked for is supported. Refer to "Mapping Data Types" on page 2-26 for the data mapping matrix.

- 79742 Can't convert from
- No data conversion is possible from the data type you specified to the column data type. The actual data type is appended to the end of this message. Check your program logic to make sure that the conversion you have asked for is supported. Refer to "Mapping Data Types" on page 2-26 for the data mapping matrix.
- 79743 Cannot load the specified IfxProtocol class
- This message occurs when Informix JDBC Driver cannot create a new instance of the **Connection** class when connecting to the Informix server. Check the **SQLException** message for more details.
- 79744 Transactions not supported
- The user has tried to call **commit()** or **rollback()** on a database that does not support transactions, or has tried to set **autoCommit** to false on a non-logging database. Verify that the current database has the correct logging mode and review the program logic.
- 79745 Read only mode not supported
- Informix does not support read-only mode.
- 79746 No Transaction Isolation on non-logging db's
- Informix does not support setting the transaction isolation level on non-logging databases.
- 79747 Invalid transaction isolation level
- If the server could not complete the rollback, this error occurs. See the rest of the **SQLException** message for more details about why the rollback failed.
- This error also occurs if an invalid transaction level is passed to **setTransactionIsolation()**. The valid values are:
- TRANSACTION\_READ\_UNCOMMITTED
  - TRANSACTION\_READ\_COMMITTED
  - TRANSACTION\_REPEATABLE\_READ
  - TRANSACTION\_SERIALIZABLE

- 79748    Can't lock the connection
- Informix JDBC Driver normally locks the connection object just before beginning the data exchange with the server. The driver could not obtain the lock. Only one thread at a time should use the connection object.
- 79749    Number of input values does not match number of question marks
- The number of bind variables that you set using the **PreparedStatement.setXXX()** methods in this statement does not match the number of ? placeholders that you wrote into the statement. Locate the text of the statement and verify the number of placeholders, then check the calls to **PreparedStatement.setXXX()**.
- 79750    Method only for queries
- The **Statement.executeQuery(String)** and **PreparedStatement.executeQuery()** methods should only be used if the statement is a SELECT statement. For other statements, use the **Statement.execute(String)**, **Statement.executeBatch()**, **Statement.executeUpdate(String)**, **Statement.getUpdateCount()**, **Statement.getResultSet()**, or **PreparedStatement.executeUpdate()** method.
- 79755    Object is null.
- The object passed in is null. Check your program logic to make sure your object reference is valid.
- 79756    must start with 'jdbc'
- The first token of the URL must be the keyword JDBC (case insensitive). For example:
- ```
URL: jdbc:informix-sqli://myachine:1234/mydatabase:user=me:password=secret
```
- 79757    Invalid sub-protocol
- The current valid subprotocol supported by Informix is **informix-sqli**.
- 79758    Invalid ip address
- When you connect to an Informix server via an IP address, the IP address must be valid. A valid IP address is set of four numbers between 0 and 255, separated by dots ( . ): for example, 127.0.0.1.

**-79759 Invalid port number**

The port number must be a valid four-digit number, as follows:

```
URL: jdbc:informix-sqli://mymachine:1234/mydatabase:user=me:password=secret
```

In this example, 1234 is the port number.

**-79760 Invalid database name**

This statement contains the name of a database in some invalid format.

The maximum length for database names and cursor names depends on the version of the database server. In 7.x, 8.x, and 9.1x versions of the Informix database server, the maximum length is 18 characters.

For INFORMIX-SE, database names should be no longer than 10 characters (fewer in some host operating systems).

Both database and cursor names must begin with a letter and contain only letters, numbers, and underscore characters. In the 6.0 and later versions of the database server, database and cursor names can begin with an underscore.

In MS-DOS systems, filenames can be a maximum of eight characters plus a three-character extension.

**-79761 Invalid Property format**

The URL accepts property values in key=value pairs. For example, `user=informix:password=informix` adds the key=value pairs to the list of properties that are passed to the connection object. Check the syntax of the key=value pair for syntax errors. Make sure there is only one = sign; that there are no spaces separating the key, value, or =; and that key=value pairs are separated by one colon (:), again with no spaces.

**-79762 Attempt to connect to a non 5.x server**

When connecting to a Version 5.x server, the user must set the URL property USE5SERVER to any non-NULL value. If a connection is then made to a version 6 or later server, this exception is thrown. Verify that the version of the server is correct and modify the URL as needed.

- 79771     Input value is not valid
- The input value is not accepted for this data type. Make sure this is a valid input for this data type.
- 79774     Unable to create local file
- Large object data read from the server can be stored either in memory or in a local file. If the LOBCACHE value is 0 or the large object size is greater than the LOBCACHE value, the large object data from the server is always stored in a file. In this case, if a security exception occurs, Informix JDBC Driver makes no attempt to store the large object into memory and throws this exception.
- 79782     Method can be called only once
- Make sure methods like **Statement.getUpdateCount()** and **Statement.getResultSet()** are called only once per result.
- 79783     Encoding or code set not supported
- The encoding or code set entered in the **DB\_LOCALE** or **CLIENT\_LOCALE** variable is not valid. Check “Internationalization” on page 2-33 for valid code sets.
- 79784     Locale not supported
- The locale entered in the **DB\_LOCALE** or **CLIENT\_LOCALE** variable is not valid. Check “Internationalization” on page 2-33 for valid locales.
- 79785     Unable to convert JDBC escape format date string to localized date string
- The JDBC escape format for date values must be specified in this format: {d 'yyyy-mm-dd'}. Verify that the JDBC escape date format specified is correct. Verify the **DBDATE** and **GL\_DATE** settings for the correct date string format if either of these was set to a value in the connection URL string or property list.



- 79786      Unable to build a Date object based on localized date string representation
- The localized date string representation specified in a CHAR, VARCHAR, or LVARCHAR column is not correct, and a date object cannot be built based on the year, month, and day values. Verify that the date string representation conforms to the **DBDATE** or **GL\_DATE** date formats if either one of these is specified in a connection URL string or property list. If neither **DBDATE** or **GL\_DATE** is specified but a **CLIENT\_LOCALE** or **DB\_LOCALE** is explicitly set in a connection URL string or property list, verify that the date string representation conforms to the JDK short default format (**DateFormat.SHORT**).
- 79788      User name must be specified
- The user name is required to establish a connection with Informix JDBC Driver. Make sure you pass in `user=your_user_name` as part of the URL or one of the properties.
- 79789      Server does not support GLS variables DB\_LOCALE, CLIENT\_LOCALE or GL\_DATE
- These variables can only be used if the server supports GLS. Check the documentation for your server version and omit these variables if they are not supported.
- 79797      DBDATE setting must be at least 4 characters and no longer than 6 characters.
- This error occurs because the **DBDATE** format string that is passed to the server either has too few characters or too many. To fix the problem, verify the **DBDATE** format string with the user documentation and make sure that the correct year, month, day, and possibly era parts of the **DBDATE** format string are correctly identified.
- 79798      A numerical year expansion is required after 'Y' character in DBDATE string.
- This error occurs because the **DBDATE** format string has a year designation (specified by the character **Y**), but there is no character following the year designation to denote the numerical year expansion (2 or 4). To fix the problem, modify the **DBDATE** format string to include the numerical year expansion value after the **Y** character.

- 79799 An invalid character is found in the DBDATE string after the 'Y' character.
- This error occurs because the **DBDATE** format string has a year designation (specified by the character Y), but the character following the year designation is not a 2 or 4 (for two-digit years and four-digit years, respectively). To fix the problem, modify the **DBDATE** format string to include the required numerical year expansion value after the Y character. Only a 2 or 4 character should immediately follow the Y character designation.
- 79800 No 'Y' character is specified before the numerical year expansion value.
- This error occurs because the **DBDATE** format string has a numerical year expansion (2 or 4 to denote two-digit years or four-digit years, respectively), but the year designation character (Y) was not found immediately before the numerical year expansion character specified. To fix the problem, modify the **DBDATE** format string to include the required Y character immediately before the numerical year expansion value requested.
- 79801 An invalid character is found in DBDATE format string.
- This error occurs because the **DBDATE** format string has a character that is not allowed. To fix the problem, modify the **DBDATE** format string to only include the correct date part designations: year (Y), numerical year expansion value (2 or 4), month (M), and day (D). Optionally, you can include an era designation (E) and a default separator character (hyphen, dot, or slash) which is specified at the end of the **DBDATE** format string. Refer to the user documentation for further information on correct **DBDATE** format string character designations.
- 79802 Not enough tokens are specified in the string representation of a date value.
- This error occurs because the date string specified does not have the minimum number of tokens or separators needed to form a valid date value (composed of year, month, and day numerical parts). For example, 12/15/98 is a valid date string representation with the slash character as the separator or token. But 12/1598 is not a valid date string representation, because there are not enough separators or tokens. To fix the problem, modify the date string representation to include a valid format for separating the day, month, and year parts of a date value.

-79803 Date string index out of bounds during date format parsing to build Date object.

This error occurs because there is not a one-to-one correspondence between the date string format required by **DBDATE** or **GL\_DATE** and the actual date string representation you defined. For example, if **GL\_DATE** is set to `%b %D %y` and you specify a character string of `Oct`, there is a definite mismatch between the format required by **GL\_DATE** and the actual date string. To fix the problem, modify the date string representation of the **DBDATE** or **GL\_DATE** setting so that the date format specified matches one-to-one with the required date string representation.

-79804 No more tokens are found in **DBDATE** string representation of a date value.

This error occurs because the date string specified does not have any more tokens or separators needed to form a valid date value (composed of year, month, and day numerical parts) based on the **DBDATE** format string. For example, `12/15/98` is a valid date string representation when **DBDATE** is set to `MDY2/`. But `12/1598` is not a valid date string representation, because there are not enough separators or tokens. To fix the problem, modify the date string representation to include a valid format for separating the day, month, and year parts of a date value based on the **DBDATE** format string setting.

-79805 No era designation found in **DBDATE/GL\_DATE** string representation of date value.

This error occurs because the date string specified does not have a valid era designation as required by the **DBDATE** or **GL\_DATE** format string setting. For example, if **DBDATE** is set to `Y2MDE-`, but the date string representation specified by the user is `98-12-15`, this is an error because there is no era designation at the end of the date string value. To fix the problem, modify the date string representation to include a valid era designation based on the **DBDATE** or **GL\_DATE** format string setting. In the above example, a date string representation of `98-12-15 AD` would probably suffice depending on the locale.

- 79806 Numerical day value can not be determined from date string based on DBDATE.
- This error occurs because the date string specified does not have a valid numerical day designation as required by the DBDATE format string setting. For example, if DBDATE is set to Y2MD-, but the date string representation you specify is 98-12-blah, this is an error, because blah is not a valid numerical day representation. To fix the problem, modify the date string representation to include a valid numerical day designation (1-31) based on the DBDATE format string setting.
- 79807 Numerical month value can not be determined from date string based on DBDATE.
- This error occurs because the date string specified does not have a valid numerical month designation as required by the DBDATE format string setting. For example, if DBDATE is set to Y2MD-, but the date string representation you specify is 98-blah-15, this is an error, because blah is not a valid numerical month representation. To fix the problem, modify the date string representation to include a valid numerical month designation (1-12) based on the DBDATE format string setting.
- 79808 Not enough tokens specified in %D directive representation of date string.
- This error occurs because the date string specified does not have the correct number of tokens or separators needed to form a valid date value based on the GL\_DATE %D directive (*mm/dd/yy* format). For example, 12/15/98 is a valid date string representation based on the GL\_DATE %D directive, but 12/1598 is not a valid date string representation, because there are not enough separators or tokens. To fix the problem, modify the date string representation to include a valid format for the GL\_DATE %D directive.
- 79809 Not enough tokens specified in %x directive representation of date string.
- This error occurs because the date string specified does not have the correct number of tokens or separators needed to form a valid date value based on the GL\_DATE %x directive (format required is based on day, month, and year parts, and the ordering of these parts is determined by the specified locale). For example, 12/15/98 is a valid date string representation based on the GL\_DATE %x directive for the U.S. English locale, but 12/1598 is not a valid date string representation, because there are not enough separators or tokens. To fix the problem, modify the date string representation to include a valid format for the GL\_DATE %x directive based on the locale.

# Index

## A

Accessing a database remotely 2-51  
 ANSI compliance  
   level Intro-10  
 APPLET tag 1-15  
 Applets and database access 2-51  
 ARCHIVE attribute of APPLET  
   tag 1-15  
 Autocommit 2-48  
 AUTOFREE environment  
   variable A-2  
 autofree.java example  
   program 3-8, A-2  
 Automatically freeing the  
   cursor 3-8

## B

BLOB data type  
   caching 2-16, 3-6  
   examples for  
     data inserts and updates 2-16  
     data retrieval 2-19  
   extensions for 2-15  
 Boldface type Intro-6  
 BOOLEAN data type 2-23, 2-29  
 Browsers 1-14  
 BYTE data type  
   caching 3-6  
   examples for  
     data inserts and updates 2-16  
     data retrieval 2-19  
   extensions for 2-15  
 ByteType.java example  
   program 2-18, 2-20, A-2

## C

Caching large objects 3-6  
 CallableStatement interface 2-15  
 cancel() method 2-53  
 Catalogs  
   Informix JDBC Driver  
     interpretation 2-15  
   systables 2-43, 2-45  
 Classes  
   IfxDriver 2-4  
   IfxJDBCProxy 2-51, 2-52  
   Locale 2-33  
   Message 2-32  
   Properties 2-10  
   ResultSet 2-38, 2-40  
   SessionMgr 2-51  
   SQLException 2-30, 2-32  
   Version 2-51  
 CLASSPATH environment  
   variable 1-13  
 Class.forName() method 2-4  
 Client hosts, specifying the locale  
   of 2-34  
 CLIENT\_LOCALE environment  
   variable 2-11, 2-34, 2-42  
 CLOB data type  
   caching 2-16, 3-6  
   examples for  
     data inserts and updates 2-16  
     data retrieval 2-19  
   extensions for 2-15  
 close() method 2-24  
 Code sets  
   conversion of 2-42, 2-47  
   synchronizing with locales 2-33  
   table of 2-43

Comment icons Intro-7  
 Compliance with industry standards Intro-10  
 Concurrency, and multiple threads 2-24  
 Connection interface 2-25, 2-48  
 Connections  
   creating 2-4, 2-8  
   establishing 2-3  
   to a database with non-ASCII characters 2-46  
 Contents of Informix JDBC Driver 1-6  
 CORBA 2-52  
 CreateDB.java example program A-2  
 createDB.java example program A-4  
 createTypes.java example program A-4  
 Creating a connection 2-4, 2-8  
 Cursors, automatically freeing 3-8

---

## D

Data types  
   BLOB 2-15, 3-6  
   BOOLEAN 2-23, 2-29  
   BYTE 2-15, 3-6  
   CLOB 2-15, 3-6  
   DATETIME 2-28  
   distinct 2-23  
   INTERVAL 2-21  
   LVARCHAR 2-23, 2-28  
   mapping between Informix and JDBC API 2-26  
   opaque 2-23  
   SERIAL 2-49  
   SERIAL8 2-49  
   TEXT 2-15, 3-6  
 Database server name, setting in database URLs 2-7  
 DatabaseMetaData interface 2-14, 2-50  
 Databases  
   names of, setting in database URLs 2-6  
   querying 2-15  
   remote access options 2-51  
   specifying the locale of 2-34  
   URL 2-5  
   with non-ASCII characters 2-46  
 Dates  
   DBDATE formats of 2-39  
   eras in 2-37  
   formatting directives for 2-35, 2-37  
   inserting values 2-37, 2-40  
   native SQL formats of 2-38, 2-40  
   nonnative SQL formats of 2-38, 2-40  
   precedence rules for end-user formats 2-41  
   represented by strings 2-38  
   retrieving values 2-38, 2-40  
   support for end-user formats 2-34  
 DATETIME type 2-28  
 DBANSIWARN environment variable 2-11  
 DBCENTURY environment variable 2-11, 2-41  
 DBCENTURYSelect.java example program A-2  
 DBConnection.java example program 2-8, A-2  
 DBDATE environment variable 2-11, 2-39, 2-41  
 DBDATESelect.java example program A-2  
 DBMetaData.java example program A-2  
 DBSPACETEMP environment variable 2-11  
 DBUPSPACE environment variable 2-11  
 DB\_LOCALE environment variable 2-11, 2-34, 2-42  
 Deallocating resources 2-24  
 Debugging 3-3  
 Default locale Intro-5  
 DELIMITENT environment variable 2-11  
 demo1.java example program A-3  
 demo2.java example program A-3  
 demo3.java example program A-3  
 demo4.java example program A-3  
 demo5.java example program A-3

Directives, formatting, for dates 2-35, 2-37  
 Distinct data type 2-23  
 distinct\_d1.java example program A-4  
 Driver interface 2-50  
 DriverManager interface 1-5, 2-4, 2-8, 2-10  
 DropDB.java example program A-2  
 dropDB.java example program A-4

---

## E

End-user formats for dates  
   precedence rules for 2-41  
   support for 2-34  
 Environment variables Intro-6  
   AUTOFREE A-2  
   CLASSPATH 1-13  
   CLIENT\_LOCALE 2-11, 2-34, 2-42  
   DBANSIWARN 2-11  
   DBCENTURY 2-11, 2-41  
   DBDATE 2-11, 2-39, 2-41  
   DBSPACETEMP 2-11  
   DBUPSPACE 2-11  
   DB\_LOCALE 2-11, 2-34, 2-42  
   DELIMITENT 2-11  
   FET\_BUF\_SIZE 2-11, 3-6, A-2  
   GL\_DATE 2-12, 2-35, 2-42  
   IFX\_AUTOFREE 2-12, 3-8  
   INFORMIXCONRETRY 2-12  
   INFORMIXCONTIME 2-12  
   INFORMIXOPCACHE 2-12  
   INFORMIXSERVER 2-7, 2-8, 2-12  
   INFORMIXSTACKSIZE 2-12  
   LOBCACHE 2-13, 2-16, 3-6  
   NODEFDAC 2-13  
   OPTCOMPIND 2-13  
   OPTOFC 2-13, 3-8, A-3  
   PATH 2-13  
   PDQPRIORITY 2-14  
   PLCONFIG 2-14  
   PROTOCOLTRACE 3-4  
   PROTOCOLTRACEFILE 3-4  
   PSORT\_DBTEMP 2-14  
   PSORT\_NPROCS 2-14

specifying 2-7, 2-9  
 supported 2-11  
 TRACE 3-4  
 TRACEFILE 3-4  
 USEV5SERVER 2-14  
 en\_us.8859-1 locale Intro-5  
 Eras in date formats 2-37  
 Errors 2-31, 2-32  
 Escape syntax 2-26  
 Establishing a connection 2-3  
 Examples  
   autofree.java 3-8, A-2  
   ByteType.java 2-18, 2-20, A-2  
   CreateDB.java A-2  
   createDB.java A-4  
   createTypes.java A-4  
   DBCENTURYSelect.java A-2  
   DBConnection.java 2-8, A-2  
   DBDATESelect.java A-2  
   DBMetaData.java A-2  
   demo1.java A-3  
   demo2.java A-3  
   demo3.java A-3  
   demo4.java A-3  
   demo5.java A-3  
   distinct\_d1.java A-4  
   DropDB.java A-2  
   dropDB.java A-4  
   GLDATESelect.java A-2  
   Interval.java 2-21, A-2  
   large object data types 2-16, 2-19  
   LOCALESelect.java A-2  
   MultiRowCall.java A-2  
   OptimizedSelect.java A-2  
   optofc.java 2-10, 3-8, A-3  
   PropertyConnection.java A-3  
   RSMetaData.java A-3  
   SimpleCall.java A-3  
   SimpleConnection.java A-3  
   SimpleSelect.java A-3  
   TextType.java 2-19, 2-21, A-3  
   udt\_d1.java A-4  
 executeQuery() method 2-25  
 executeUpdate() method 2-8, 2-19  
 execute() method 2-24, 2-53

## F

FET\_BUF\_SIZE environment  
   variable 2-11, 3-6, A-2  
 File interface 2-19  
 FileInputStream interface 2-19  
 Files  
   IfxJDBCProxy.class 1-7, 2-51  
   java.io 2-33  
   java.text 2-33  
   java.util 2-33  
   SessionMgr.class 1-7, 2-51  
 Firewalls and database access 2-51  
 Formatting directives for  
   dates 2-35, 2-37

## G

getBoolean() method 2-23  
 getBytes() method 2-47  
 getCatalogName() method 2-53  
 getCatalogs() method 2-15  
 getConnection() method 2-4, 2-8,  
   2-10  
 getErrorCode() method 2-31  
 getJDBCVersion() method 2-50  
 getMajorVersion() method 2-50  
 getMessage() method 2-31  
 getMinorVersion() method 2-50  
 getSchemaName() method 2-53  
 getSchemas() method 2-15  
 getSerial8() method 2-49  
 getSerial() method 2-49  
 getString() method 2-21, 2-23, 2-38,  
   2-40, 2-47  
 getTableName() method 2-53  
 getText() method 2-46  
 getUnicodeStream() method 2-53  
 getXXX() method 2-25, 2-29  
 GLDATESelect.java example  
   program A-2  
 Global Language Support  
   (GLS) Intro-5, 2-33  
 GL\_DATE environment  
   variable 2-12, 2-35, 2-42

## H

Host names, setting in database  
   URLs 2-6  
 HTTP proxy 2-51

## I

Icons  
   Important Intro-7  
   platform Intro-7  
   Tip Intro-7  
   Warning Intro-7  
 IfxDriver class 2-4  
 ifxjdbc-g.jar file 1-6, 1-14, 3-3  
 IfxJDBCProxy class 2-51, 2-52  
 IfxJDBCProxy.class file 1-7, 2-51  
 ifxjdbc.jar file 1-6, 1-14  
 IFX\_AUTOFREE environment  
   variable 2-12, 3-8  
 Important paragraphs, icon  
   for Intro-7  
 Industry standards, compliance  
   with Intro-10  
 Informix JDBC Driver  
   contents of 1-6  
   installing interactively 1-7  
   installing silently 1-10  
   loading 2-4  
   overview of 1-6  
   registering 2-4  
   tracing 3-4  
   uninstalling 1-12  
   using debug version of 3-3  
   using in an applet 1-14, 2-4  
   using in an application 1-13  
 INFORMIXCONRETRY  
   environment variable 2-12  
 INFORMIXCONTIME  
   environment variable 2-12  
 INFORMIXOPCACHE  
   environment variable 2-12  
 INFORMIXSERVER environment  
   variable 2-7, 2-8, 2-12  
 INFORMIXSTACKSIZE  
   environment variable 2-12  
 InputStream interface 2-16

InputStreamReader() method 2-46, 2-47

Inserting date values 2-37, 2-40

Installing Informix JDBC  
 Driver 1-7, 1-10

Interfaces  
 CallableStatement 2-15  
 Connection 2-25, 2-48  
 DatabaseMetaData 2-14, 2-50  
 Driver 2-50  
 DriverManager 1-5, 2-4, 2-8, 2-10  
 File 2-19  
 FileInputStream 2-19  
 InputStream 2-16  
 PreparedStatement 2-15, 2-25  
 ResultSet 2-15, 2-24, 2-25, 2-29, 3-8  
 ResultSetMetaData 2-15  
 Statement 2-8, 2-15, 3-8  
 Types 2-27, 2-49

Internationalization 2-33 to 2-48

INTERVAL data type 2-21

Interval.java example  
 program 2-21, A-2

IP address, setting in database  
 URL 2-6

isDefinitelyWritable()  
 method 2-54

ISO 8859-1 code set Intro-5

isReadOnly() method 2-53

isWritable() method 2-54

## J

JAR files  
 ifxjdbc-g.jar 1-6, 1-14, 3-3  
 ifxjdbc.jar 1-6, 1-14

jar utility 1-14

Java virtual machine (JVM) 1-13

javac, Java compiler 1-6

JavaSoft 1-3, 1-14

java.io file 2-33

java.text file 2-33

java.util file 2-33

JDBC API 1-3, 1-4

JDBC driver, general 1-5

## L

Limitations and restrictions of  
 Informix JDBC Driver 2-53

Loading Informix JDBC Driver 2-4

LOBCACHE environment  
 variable 2-13, 2-16, 3-6

Locale class 2-33

Locales  
 assumptions about Intro-5  
 client, specifying 2-34  
 database, specifying 2-34  
 synchronizing with code sets 2-33  
 table of 2-45

LOCALESelect.java example  
 program A-2

Localization 2-33

LVARCHAR data type 2-23, 2-28

## M

Mapping between Informix and  
 JDBC API data types 2-26

Message class 2-32

Metadata, accessing database 2-14

Methods  
 cancel() 2-53  
 Class.forName() 2-4  
 close() 2-24  
 executeQuery() 2-25  
 executeUpdate() 2-8, 2-19  
 execute() 2-24, 2-53  
 getBoolean() 2-23  
 getBytes() 2-47  
 getCatalogName() 2-53  
 getCatalogs() 2-15  
 getConnection() 2-4, 2-8, 2-10  
 getErrorCode() 2-31  
 getJDBCVersion() 2-50  
 getMajorVersion() 2-50  
 getMessage() 2-31  
 getMinorVersion() 2-50  
 getSchemaName() 2-53  
 getSchemas() 2-15  
 getSerial8() 2-49  
 getSerial() 2-49  
 getString() 2-21, 2-23, 2-38, 2-40, 2-47

getTableName() 2-53

getText() 2-46

getUnicodeStream() 2-53

getXXX() 2-25, 2-29

InputStreamReader() 2-46, 2-47

isDefinitelyWritable() 2-54

isReadOnly() 2-53

isWritable() 2-54

next() 2-19, 2-25

OutputStreamWriter() 2-46, 2-47

prepareStatement() 2-25

put() 2-10

registerDriver() 2-4

registerOutParameter() 2-53

setAsciiStream() 2-17

setAutoCommit() 2-48

setBinaryStream() 2-17

setBoolean() 2-23

setCatalog() 2-53

setMaxFieldSize() 2-53

setObject() 2-25

setQueryTimeout() 2-53

setReadOnly() 2-53

setString() 2-21, 2-23

setUnicodeStream() 2-53

MultiRowCall.java example  
 program A-2

## N

Name-value pairs of database  
 URL 2-7

Native SQL date formats 2-38, 2-40

next() method 2-19, 2-25

NODEFDAC environment  
 variable 2-13

Nonnative SQL date formats 2-38, 2-40

## O

ODBC 1-5

Opaque data type 2-23

OPTCOMPIND environment  
 variable 2-13

OptimizedSelect.java example  
 program A-2



OPTOFC environment  
 variable 2-13, 3-8, A-3  
 optofc.java example program 2-10,  
 3-8, A-3  
 OutputStreamWriter()  
 method 2-46, 2-47

---

## P

Passwords  
 URL syntax of 2-7  
 PATH environment variable 2-13  
 PDQPRIORITY environment  
 variable 2-14  
 Performance 3-6  
 Platform icons Intro-7  
 PLCONFIG environment  
 variable 2-14  
 Port numbers, setting in database  
 URL 2-6  
 Precedence rules for date  
 formats 2-41  
 PreparedStatement interface 2-15,  
 2-25  
 prepareStatement() method 2-25  
 Properties class 2-10  
 Property lists 2-10  
 PropertyConnection.java example  
 program A-3  
 PROTOCOLTRACE environment  
 variable 3-4  
 PROTOCOLTRACEFILE  
 environment variable 3-4  
 Proxy server 2-51  
 PSORT\_DBTEMP environment  
 variable 2-14  
 PSORT\_NPROCS environment  
 variable 2-14  
 put() method 2-10

---

## Q

Querying the database 2-15

---

## R

registerDriver() method 2-4

Registering Informix JDBC  
 Driver 2-4  
 registerOutParameter()  
 method 2-53  
 Remote database access 2-51  
 Remote method invocation  
 (RMI) 2-52, A-1, A-2  
 Restrictions and limitations of  
 Informix JDBC Driver 2-53  
 ResultSet class 2-38, 2-40  
 ResultSet interface 2-15, 2-24, 2-25,  
 2-29, 3-8  
 ResultSetMetaData interface 2-15  
 Retrieving  
 database names 2-15  
 date values 2-38, 2-40  
 Informix error message text 2-32  
 user names 2-15  
 version information 2-50  
 RMI 2-52, A-1, A-2  
 RSMetaData.java example  
 program A-3

---

## S

Schemas, Informix JDBC Driver  
 interpretation 2-15  
 SERIAL data type 2-49  
 SERIAL8 data type 2-49  
 Servlets 2-51, 2-52  
 SessionMgr class 2-51  
 SessionMgr.class file 1-7, 2-51  
 setAsciiStream() method 2-17  
 setAutoCommit() method 2-48  
 setBinaryStream() method 2-17  
 setBoolean() method 2-23  
 setCatalog() method 2-53  
 setMaxFieldSize() method 2-53  
 setObject() method 2-25  
 setQueryTimeout() method 2-53  
 setReadOnly() method 2-53  
 setString() method 2-21, 2-23  
 Setting  
 autocommit 2-48  
 properties 2-10  
 the CLASSPATH environment  
 variable 1-13, 1-14  
 setUnicodeStream() method 2-53

setup.class class file 1-6, 1-8, 1-9,  
 1-10, 1-11  
 SimpleCall.java example  
 program A-3  
 SimpleConnection.java example  
 program A-3  
 SimpleSelect.java example  
 program A-3  
 Specifying  
 environment variables 2-7, 2-9  
 the client locale 2-34  
 the database locale 2-34  
 SQL date formats  
 native 2-38, 2-40  
 nonnative 2-38, 2-40  
 SQLException class 2-30, 2-32  
 Statement interface 2-8, 2-15, 3-8  
 Strings, representing dates  
 using 2-38  
 Supported environment  
 variables 2-11  
 Supported getXXX() methods 2-29  
 Syntax of Database URL 2-5  
 sysmaster database 2-14  
 systables catalog  
 and code set conversion 2-43,  
 2-45  
 and metadata 2-15

---

## T

TEXT data type  
 caching 3-6  
 code set conversion for 2-47  
 examples for  
 data inserts and updates 2-16  
 data retrieval 2-19  
 extensions for 2-15  
 TextType.java example  
 program 2-19, 2-21, A-3  
 Threads, multiple, and  
 concurrency 2-24  
 Tip icons Intro-7  
 TRACE environment variable 3-4  
 TRACEFILE environment  
 variable 3-4  
 Tracing 3-4  
 Transactions, handling 2-48

Tuple buffer 2-11, 3-6  
Types interface 2-27, 2-49

---

## U

udt\_d1.java example program A-4  
Unicode  
    and internationalization  
        APIs 2-33  
    and the client code set 2-46  
    and the database code set 2-43  
Uninstalling Informix JDBC  
    Driver 1-12  
URLs  
    database 2-5  
    for a proxy server 2-51  
User names, setting in database  
    URLs 2-7  
USEV5SERVER environment  
    variable 2-14  
Using debug version of Informix  
    JDBC Driver 3-3  
Using Informix JDBC Driver  
    in an applet 1-14, 2-4  
    in an application 1-13  
Using INFORMIX-OnLine 5.x  
    database servers 2-14  
Using INFORMIX-SE 5.x database  
    servers 2-14  
Using the Informix serial data  
    types 2-49  
Utilities, jar 1-14

---

## V

Version class 2-51  
Version, of Informix JDBC  
    Driver 2-50

---

## W

Warning icons Intro-7

---

## X

X/Open compliance level Intro-10