IBM Informix ODBC Driver Programmer's Manual

Version 3.82
March 2003
Part No. CT1UFNA

Note:
Before using this information and the product it supports, read the information in the appendix entitled "Notices."

Table of Contents

Chapter 3 Data Types

Chapter 4 Working with Smart Large Objects

Chapter 5 Working with Rows and Collections

Chapter 6 Client Functions

Chapter 8 Error Messages

Introduction

In This Introduction

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

About This Manual

This manual is a user guide and reference manual for IBM Informix ODBC Driver, which is the Informix implementation of the Microsoft Open Database Connectivity (ODBC) interface, Version 3.0. This manual explains how to use the IBM Informix ODBC Driver application programming interface (API) to access an Informix database and interact with an Informix database server.

Types of Users

This manual is written for C programmers who use IBM Informix ODBC Driver to access Informix databases.

This manual assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational or object-relational databases, or exposure to relational database concepts
- C programming language

If you have limited experience with relational databases, SQL, or your operating system, refer to the *Getting Started Guide* for your database server for a list of supplementary titles.

Software Dependencies

This manual assumes that you are using IBM Informix ODBC Driver, Version 3.82, on either UNIX, Windows NT, Windows 95, or Windows 98.

In places where this manual presents database server-specific information, this information applies to one of the following database servers:

- IBM Informix Dynamic Server, Version 7.3
- IBM Informix Extended Parallel Server, Version 8.3
- IBM Informix Dynamic Server, Version 9.3

If you are using a database server that is not listed here, see your release notes for information about client behavior on your database server.

Assumptions About Your Locale

IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation and representation of numeric data, currency, date, and time is brought together in a single environment, called a Global Language Support (GLS) 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 *IBM Informix GLS User's Guide*.

Demonstration Databases

The demonstration files that you receive with IBM Informix ODBC Driver contain demonstration source programs and files that create, populate, and drop the **odbc_demodb** database. You can find the demonstration files in the following locations.

UNIX

On UNIX (Solaris), the ODBC demonstration files are located at:

```
$INFORMIXDIR/demo/clidemo
```

♦

Windows

In Windows, the ODBC demonstration files are located at:

```
%INFORMIXDIR%\demo\odbcdemo
```

♦

The **readme.txt** file in each of these locations describes each of the demonstration files and its purpose. Some of the demonstration program source code is also provided as example text in the manual.

New Features

IBM Informix ODBC Driver, Version 3.82, is a new product that replaces IBM Informix CLI, Version 2.8. IBM Informix ODBC Driver implements the Microsoft Open Database Connectivity (ODBC) Version 3.0 standard.

New features in IBM Informix ODBC Driver include:

- Fetch array size
- Large object automation
- Deferred-PREPARE
- Insert cursor
- Auto freeing of cursors
- Using Stored Procedure Language (SPL) output parameters
- Manual Data-Source Name (DSN) Migration
- DSN Migration with the DSN Migrate graphical user interface (GUI)
- Default user-defined type (UDT) fetch type

- Advanced tab options
- Support for third-party applications
- Case-sensitive catalog functions
- Message transfer optimization (OPTMSG)

UNIX

IBM Informix Client Software Developer's Kit, Version 2.8, supports silent installation on UNIX. The installation script includes additional options that allow you to override version checking, so that the installation can proceed without further user interaction. ♦

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.

The following conventions are discussed:

- Typographical conventions
- Icon conventions
- Sample-code conventions

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, 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.
italics ***italics*** `italics`	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.
boldface ***boldface***	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` *`monospace`*	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.

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.*

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
	***Warning:***	Identifies paragraphs that contain vital instructions, cautions, or critical information
	***Important:***	Identifies paragraphs that contain significant information about the feature or operation that is being described
	***Tip:***	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Feature, Product, and Platform Icons

Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
Core	Identifies information that relates to ODBC interface conformance The Core level is the minimum level to which all ODBC drivers are expected to conform.
GLS	Identifies information that relates to the IBM Informix Global Language Support (GLS) feature
IDS	Identifies information that is specific to IBM Informix Dynamic Server

(1 of 2)

Icon	Description
Level 1	Identifies information that relates to ODBC interface conformance Level 1 includes the core standards compliance level functionality and some additional features, including scrollable cursors, bookmarks, and positioned updates and deletes.
Level 2	Identifies information that relates to ODBC interface conformance Level 2 includes the Level 1 standards compliance level functionality and some additional features, including bookmark update, delete, and refresh; stored procedure support; catalog functions for primary and foreign keys; and multicatalog support.
UNIX	Identifies information that is specific to UNIX platforms
Windows	Identifies information that is specific to Windows NT, Windows 95, and Windows 98 environments

(2 of 2)

These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the information that applies to the indicated feature, product, or platform ends at the next heading at the same or higher level. A ♦ symbol indicates the end of feature-, product-, or platform-specific information that appears within one or more paragraphs within a section.

Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single IBM Informix application development tool. If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO sales_demo
⋮

DELETE FROM customer
   WHERE customer_num = 121
⋮

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using DB-Access, you must delimit multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement.

Tip: *Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.*

For detailed directions on using SQL statements for a particular application development tool or client product, see the manual for your product.

Additional Documentation

For additional information, you might want to refer to the following types of documentation:

- Online manuals
- Online help
- Error message files
- Documentation notes, release notes, and machine notes
- Related reading

Online Manuals

You can obtain the online manuals at the IBM Informix Online Documentation site at
http://www.ibm.com/software/data/informix/pubs/library/.

Windows

Online Help

IBM Informix online help, provided with each graphical user interface (GUI), displays information about those interfaces and the functions that they perform. Use the help facilities that each GUI provides to display the online help.

Error Message Files

IBM Informix software products provide ASCII files that contain all of the error messages and their corrective actions. For a detailed description of these error messages, refer to *IBM Informix Error Messages* at the IBM Informix Online Documentation site at
http://www.ibm.com/software/data/informix/pubs/library/.

UNIX

To read the error messages on UNIX, you can use the following commands.

Command	Description
finderr	Displays error messages online
rofferr	Formats error messages for printing

♦

Windows

To read error messages and corrective actions on Windows NT, use the **Informix Error Messages** utility. To display this utility, choose **Start→Programs→Informix** from the task bar. ♦

Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following sections describe the online files that supplement the information in this manual. Please examine these files before you begin using your database server and client products. They contain vital information about application and performance issues.

UNIX

On UNIX platforms, the following online files appear in the **$INFORMIXDIR/release/en_us/0333** directory.

Online File	Purpose
csdk_odbc_docnotes_3.82.html **csdk_odbc_docnotes_3.82.txt**	The documentation notes file describes features that are not covered in the manual or that were modified since publication.
csdk_unix_release_notes_3.82.html **csdk_unix_release_notes_3.82.txt**	The release notes file describes feature differences from earlier versions of IBM Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
csdk_machine_notes_3.82.txt	The machine notes file describes any special actions that you must take to configure and use IBM Informix products on your computer. Machine notes are named for the product described.

♦

Windows

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix** from the Task Bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals and information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of IBM Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.The release notes file for IBM Informix Client Software Developer's Kit includes information about database server compatibility.

Machine notes do not apply to Windows environments. ♦

Related Reading

The following publications provide additional information about the topics that this manual discusses. For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your *Getting Started Guide*.

- IBM Informix Client Software Developer's Kit, Version 2.8
- Microsoft ODBC 3.5 Software Development Kit and Programmer's Reference (ISBN Number 1-57231-516-4)

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.

IBM Welcomes Your Comments

To help us with future versions of our manuals, let us know about any corrections or clarifications that you would find useful. Include the following information:

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

Send electronic mail to us at the following address:

docinf@us.ibm.com

This address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact Customer Services.

Chapter

1

Overview of IBM Informix ODBC Driver

In This Chapter

This chapter introduces IBM Informix ODBC Driver, Version 3.82, and describes its advantages and architecture. The chapter also describes conformance, isolation and lock levels, libraries, and environment variables.

What Is IBM Informix ODBC Driver?

Open Database Connectivity (ODBC) is a specification for a database Application Programming Interface (API). Microsoft ODBC, Version 3.0, is based on the Call Level Interface specifications from X/Open and the International Standards Organization/International Electromechanical Commission (ISO/IEC). ODBC supports SQL statements with a library of C functions. An application calls these functions to implement ODBC functionality.

ODBC applications enable you to perform the following operations:

- Connect to and disconnect from data sources
- Retrieve information about data sources
- Retrieve information about IBM Informix ODBC Driver
- Set and retrieve IBM Informix ODBC Driver options
- Prepare and send SQL statements
- Retrieve SQL results and process the results dynamically
- Retrieve information about SQL results and process the information dynamically

ODBC lets you allocate storage for results before or after the results are available. This feature lets you determine the results and the action to take without the limitations that predefined data structures impose.

ODBC does not require a preprocessor to compile an application program.

IBM Informix ODBC Driver Features

IBM Informix ODBC Driver implements the Microsoft Open Database Connectivity (ODBC) Version 3.0 standard. The IBM Informix ODBC Driver product also provides the following features and functionality:

- Data Source Name (DSN) migration
- UNIX — Driver Manager Replacement Module, which supports compatibility between ODBC 2.x applications and the ODBC driver, Version 3.82. ♦
- Windows — Microsoft Transaction Server (MTS), which is an environment that lets you develop, run, and manage scalable, component-based Internet and intranet server applications. MTS performs the following tasks:
  - Manages system resources, including processes, threads, and database connections, so that your application can scale to many simultaneous users
  - Manages server component creation, execution, and deletion
  - Automatically initiates and controls transactions to make your application reliable
  - Implements security so that unauthorized users cannot access your application
  - Provides tools for configuration, management, and deployment ♦
- IDS — Extended data types, including rows and collections ♦
- Long identifiers
- Limited support of bookmarks
- GLS data types
- Extensive error detection
- Unicode support
- XA support

IDS

Support for Extended Data Types

IBM Informix ODBC Driver supports the following extended data types:

- Collection (LIST, MULTISET, SET)
- DISTINCT
- OPAQUE (fixed, unnamed)
- Row (named, unnamed)
- Smart large object (BLOB, CLOB)
- Client functions to support some of the extended data types

Support for GLS Data Types

IBM Informix ODBC Driver supports the following GLS data types:

- NCHAR
- NVCHAR

For more information on data types, see Chapter 3, "Data Types."

Extended Error Detection

IBM Informix ODBC Driver detects the following types of errors:

- ISAM
- XA

Additional Values for Some ODBC Function Arguments

IBM Informix ODBC Driver supports additional values for some ODBC function arguments including:

IDS

- *fDescType* values for **SQLColAttributes**
  - SQL_INFX_ATTR_FLAGS
  - SQL_INFX_ATTR_EXTENDED_TYPE_ALIGNMENT
  - SQL_INFX_ATTR_EXTENDED_TYPE_CODE
  - SQL_INFX_ATTR_EXTENDED_TYPE_NAME
  - SQL_INFX_ATTR_EXTENDED_TYPE_OWNER
  - SQL_INFX_ATTR_SOURCE_TYPE_CODE
- *fInfoType* return value for **SQLGetInfo**
  - SQL_INFX_LO_PTR_LENGTH
  - SQL_INFX_LO_SPEC_LENGTH ♦
- SQL_INFX_LO_STAT_LENGTH
- *fOption* value for **SQLGetConnectOption** and **SQLSetConnectOption**: SQL_INFX_OPT_LONGID

Windows

- *fOption* value for **SQLGetConnectOption** and **SQLSetConnectOption:** SQL_ATTR_ENLIST_IN_DTC ♦

ODBC Component Overview

ODBC with the IBM Informix ODBC Driver can include the following components:

- Driver manager

  An application can link to a driver manager, which links to the driver specified by the data source. The driver manager also checks parameters and transitions. On most UNIX platforms, the ODBC Driver Manager can be purchased from a third-party vendor. The Data Direct ODBC driver manager is shipped with the CSDK bundle for Sun Solaris 32 bit & IBM AIX 32-bit platforms.

  On Microsoft Windows platforms, the ODBC Driver Manager is a part of the OS.

- IBM Informix ODBC Driver

  The driver provides an interface to Informix database server. Applications can use the driver in the following configurations:

  - to link to the ODBC driver manager
  - to link to the Driver Manager Replacement & the driver
  - to link to the driver directly

- Data sources

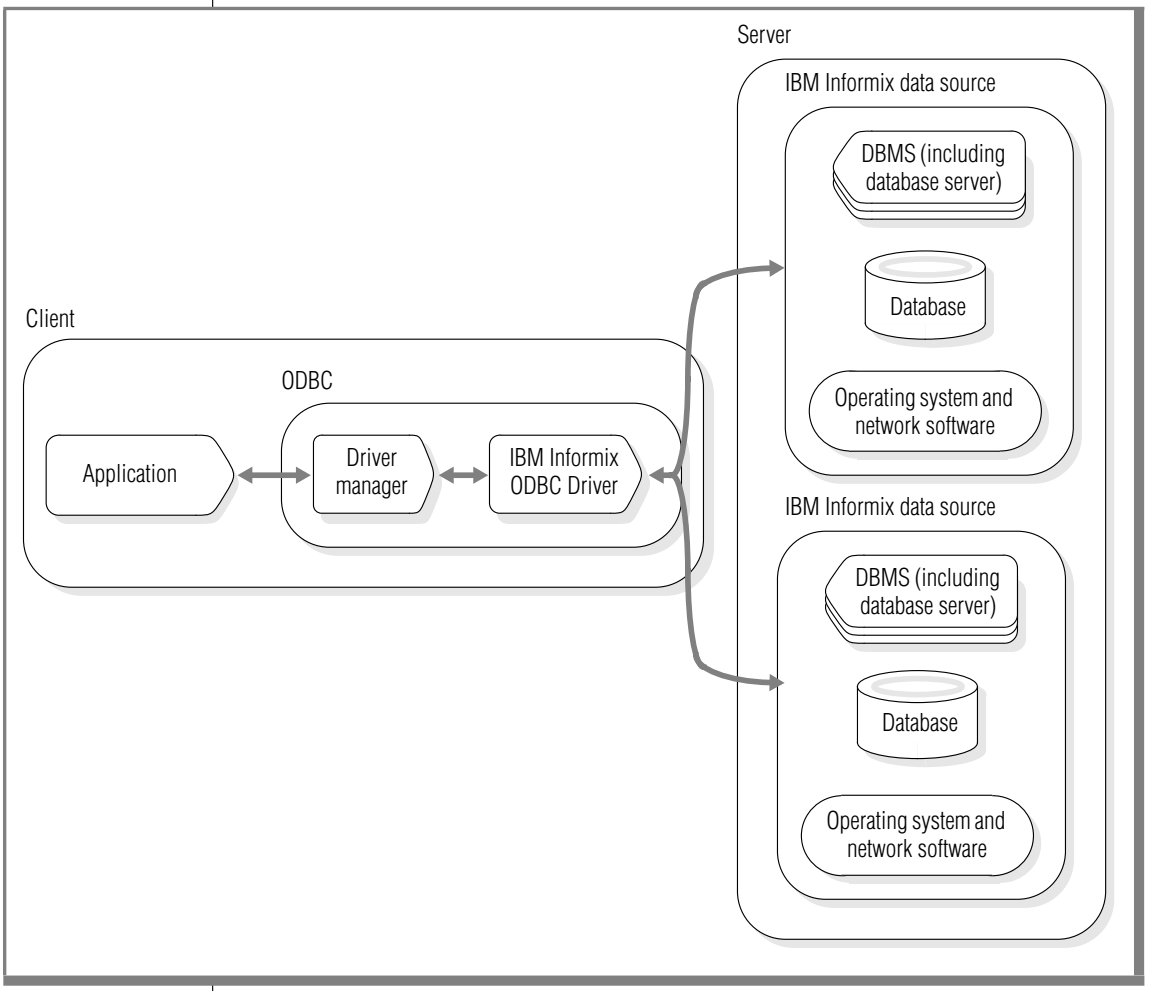
  The drive provides access to the following data sources:

  - database management systems (DBMS), including a database server
  - databases
  - operating systems and network software required for accessing the database

IBM Informix ODBC Driver with a Driver Manager

Figure 1-1 shows the software architecture when a driver manager is included in the system. In such a system, the driver and driver manager act like a single unit that processes function calls.

Figure 1-1
IBM Informix ODBC Driver with a Driver Manager

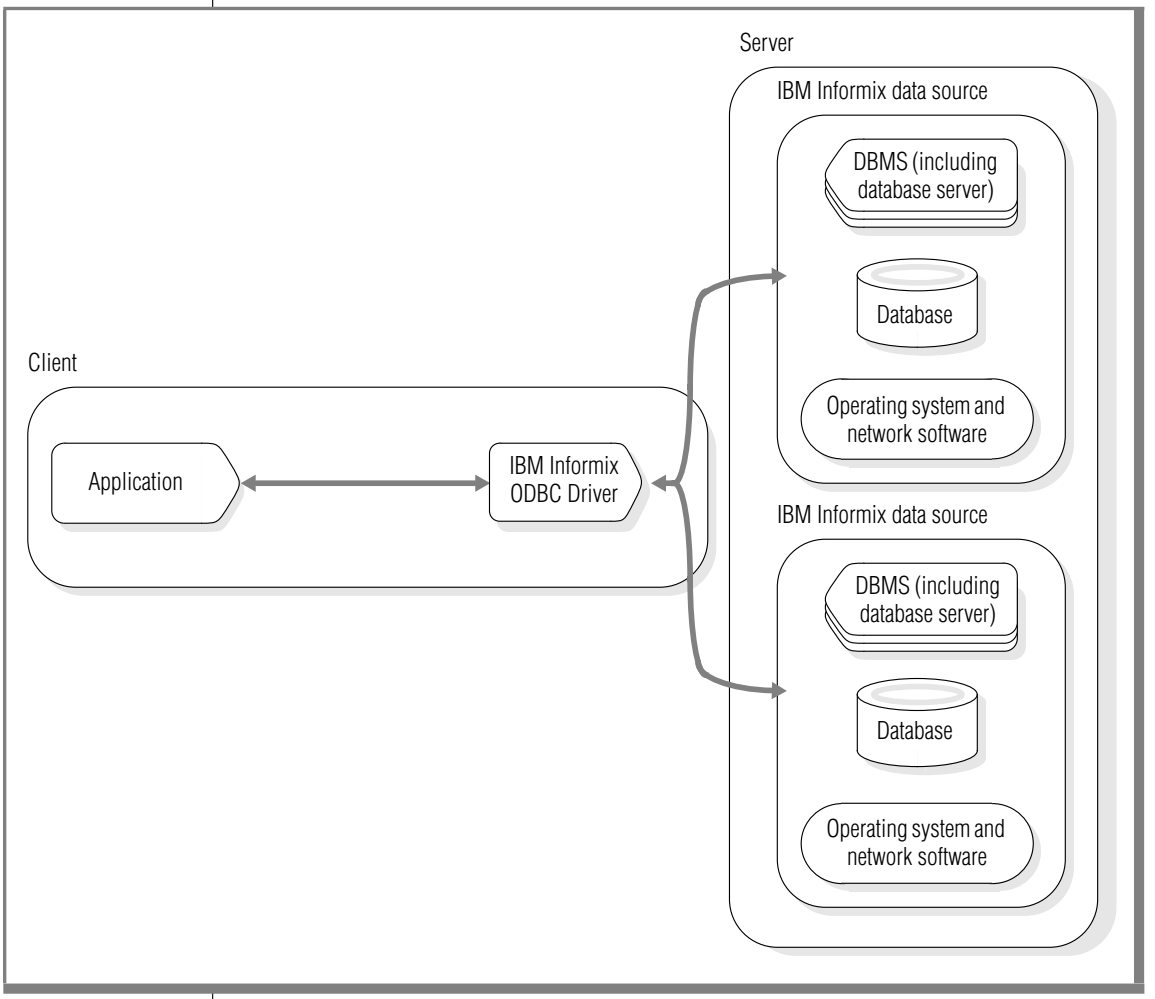


UNIX

IBM Informix ODBC Driver Without a Driver Manager

Figure 1-2 shows an application using IBM Informix ODBC Driver without a driver manager. In this case, the application must link to the IBM Informix ODBC Driver library. For more information, see "Libraries" on page 1-14.

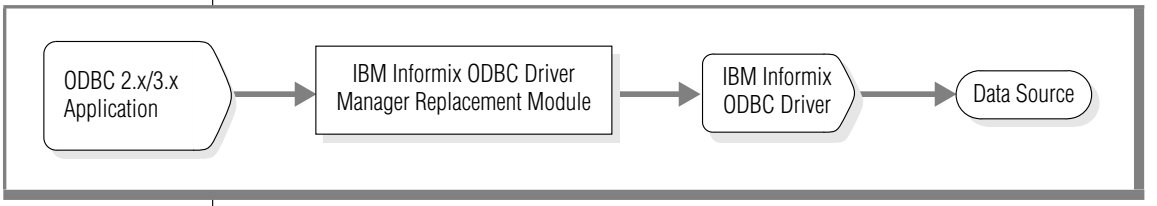
Figure 1-2
IBM Informix ODBC Driver Without a Driver Manager



IBM Informix ODBC Driver with the DMR

IBM Informix ODBC Driver includes a Driver Manager Replacement (DMR) library. The DMR replaces the driver manager on platforms where no driver manager is available. Figure 1-3 shows an ODBC configuration with the DMR.

Figure 1-3
Architecture of the Driver Manager Replacement Module



Applications that are linked directly to the ODBC 3.5 driver and the DMR do not require the ODBC Driver Manager.

In addition to supporting ODBC 3.5 features, the DMR supports compatibility between ODBC 2.x applications and Version 3.82 of the IBM Informix ODBC Driver. To be compatible with ODBC 2.x applications, the application must link to Version 3.82 of IBM Informix ODBC Driver through the DMR or through the ODBC 3.5 driver manager.

You cannot use the IBM Informix DMR to connect to non-Informix data sources. The DMR does not support connection pooling. The DMR does not map between Unicode and ANSI APIs.

Using IBM Informix ODBC Driver

IBM Informix ODBC Driver includes the following components:

- Environment variables
- Header files
- Data types
- Libraries

Environment Variables

The following table describes environment variables that you must set for the driver. For more information about environment variables, see the *IBM Informix Guide to SQL: Reference*.

Environment Variable	Description
INFORMIXDIR	Full path of the directory where the IBM Informix Client Software Developer's Kit is installed. On Windows platforms, **INFORMIXDIR** is a registry setting rather than an environment variable. It is set during installation.
PATH	Directories that are searched for executable programs. Your **PATH** setting must include the path to your **$INFORMIXDIR/bin** directory.
DBCENTURY (optional)	Controls the setting of year values. **DBCENTURY** affects a client program when a user issues a statement that contains a date or datetime string that specifies only the last two digits of the year. For example: `insert into datetable ( datecol ) values ("01/01/01");` The database server stores the date specified in this statement as either 01-01-1901 or 01-01-2001, depending on the **DBCENTURY** value on the client.
DBDATE or **GL_DATE** (optional)	**DBDATE** and **GL_DATE** control the interpretation of dates. For example, you can specify whether the date format is *mm-dd-yyyy* or *yyyy-mm-dd*.

UNIX

Setting Environment Variables on UNIX

If you set these environment variables at the command line, you must reset them whenever you log onto your system. If you set these environment variables in a file, they are set automatically when you log onto your system.

IBM Informix ODBC Driver provides a sample setup file called **setup.odbc** in **$INFORMIXDIR/etc**. You can use this file to set environment variables for the driver. The following table describes the environment variables that are in **setup.odbc**.

Environment Variable	Description
INFORMIXDIR	Full path of the directory where IBM Informix Client Software Developer's Kit is installed
INFORMIXSQLHOSTS	This value is optional. It specifies the directory that contains **sqlhosts**. By default, **sqlhosts** is in **$INFORMIXDIR/etc**. Set **INFORMIXSQLHOSTS** if you want **sqlhosts** to be in a different directory.
ODBCINI	This value is optional. You can use it to specify an alternative location for the **odbc.ini** file. The default location is your home directory.

Windows

Setting Environment Variables in Windows

If you set the environment variables at the command line, you must reset them whenever you log into your Windows environment. If you set them in the Windows registry, however, they are set automatically when you log in. IBM Informix ODBC Driver stores environment variables in the following location in the Windows registry:

```
\HKEY_CURRENT_USERS\Software\Informix\Environment
```

In a Windows environment you must use **setnet32.exe**, or a tool that will update the registry correctly, to set environment variables that IBM Informix dynamic link libraries (DLLs), such as **iclit09b.dll**, use.

You can use environment variables as required by your development environment. For example, the compiler needs to know where to find include files. To specify the location of include files, set the environment variable **INFORMIXDIR** (or some other environment variable) and then set the include path to **INFORMIXDIR\incl\cli**.

Header Files

You can use the **sql.h** and **sqlext.h** header files, which are part of the Microsoft compiler, to run IBM Informix ODBC Driver. To run Informix extensions, include the **infxcli.h** file, which is installed in **INFORMIXDIR/incl/cli**. This file defines IBM Informix ODBC Driver constants and types, and provides function prototypes for the IBM Informix ODBC Driver functions. If you include the **infxcli.h** file, it automatically includes the **slq.h** and **sqlext.h** files.

The **sql.h** and **sqlext.h** header files contain definitions of the C data types. For more information about data types, see Chapter 3.

You should include **xa.h** in XA ODBC applications.

Data Types

A column of data stored on a data source has an SQL data type. IBM Informix ODBC Driver maps Informix-specific SQL data types to ODBC SQL data types, which are defined in the ODBC SQL grammar. (The driver returns these mappings through **SQLGetTypeInfo**. It also uses the ODBC SQL data types to describe the data types of columns and parameters in **SQLColAttributes** and **SQLDescribeCol**).

Each SQL data type corresponds to an ODBC C data type. By default, the driver assumes that the C data type of a storage location corresponds to the SQL data type of the column or parameter to which the location is bound. If the C data type of a storage location is not the *default* C data type, the application can specify the correct C data type with the *TargetType* argument for **SQLBindCol**, the *fCType* argument for **SQLGetData**, and the *ValueType* argument in **SQLBindParameter**. Before the driver returns data from the data source, it converts the data to the specified C data type. Before the driver sends data to the data source, it converts the data from the specified C data type to the SQL data type.

The Informix data type names differ from the Microsoft ODBC data type names. For information about these differences, refer to Chapter 3, "Data Types" and to the appendix on data types in the *Microsoft ODBC 3.0 Programmer's Reference*.

Libraries

UNIX

The installation procedure installs the following libraries into **INFORMIXDIR/lib/cli**. In each data source specification section in the **odbc.ini** file, set the driver value indicating the full path to one of the following library filenames.

Filename	Description
libifcli.a or **libcli.a**	Static version for single (nonthreaded) library
libifcli.so or **iclis09b.so**	Shared version for single (nonthreaded) library
libthcli.a	Static version for multithreaded library
libthcli.so or **iclit09b.so**	Shared version for multithreaded library
libifdrm.so or **idmrs09a.so**	Shared library for DMR (thread safe)

For more information about the **odbc.ini** file, see "The odbc.ini File" on page 2-6.

If you do not use a driver manager, your application needs to link to either the static or the shared version of the IBM Informix ODBC Driver libraries.

The following compile command links an application to the thread-safe version of the IBM Informix ODBC Driver libraries:

```
cc ... -L$INFORMIXDIR/lib/cli -lifdmr - lthcli
```

♦

Windows

The installation procedure installs the following libraries into **INFORMIXDIR\lib**.

Filename	Description
iclit09b.lib	Enables linking directly to the driver without the use of a driver manager
iregt07b.lib	Allows linking directly to **iregt07b.dll**

♦

The following compile command links an application to the thread-safe version of the IBM Informix ODBC Driver libraries:

```
cl ... -L$INFORMIXDIR/lib/cli iclit09b.lib
```

If you use a driver manager, you must link your application to the driver manager library only, as the following example shows:

```
cl odbc32.lib
```

IBM Informix ODBC Driver requires a Version 3.0 driver manager.

Using the IBM Informix ODBC Driver API

An application uses the IBM Informix ODBC Driver API to make a connection to a data source, send SQL statements to a data source, process result data dynamically, and terminate a connection. The driver enables your application to perform the following steps:

1. Connect to the data source.

   You can connect to the data source through a DSN connection, or you can use DSN-less connection strings. Specify the data-source name and any additional information needed to complete the connection.
2. Process one or more SQL statements:
   a. Place the SQL text string in a buffer. If the statement includes parameter markers, set the parameter values.
   b. If the statement returns a result set, either assign a cursor name for the statement or let the driver assign one.
   c. Either prepare the statement or submit it for immediate execution.
   d. If the statement creates a result set, you can inquire about the attributes of the result set, such as the number of columns and the name and type of a specific column. For each column in the result set, assign storage and fetch the results.
   e. If the statement causes an error, retrieve error information from the driver and take the appropriate action.
3. End any transaction by committing it or rolling it back.
4. Terminate the connection when the application finishes interacting with the data source.

Every IBM Informix ODBC Driver function name starts with the prefix SQL. Each function accepts one or more arguments. Arguments are defined as input (to the driver) or output (from the driver).

Figure 1-4 on page 1-17 shows the basic function calls that an application makes even though an application generally calls other functions also.

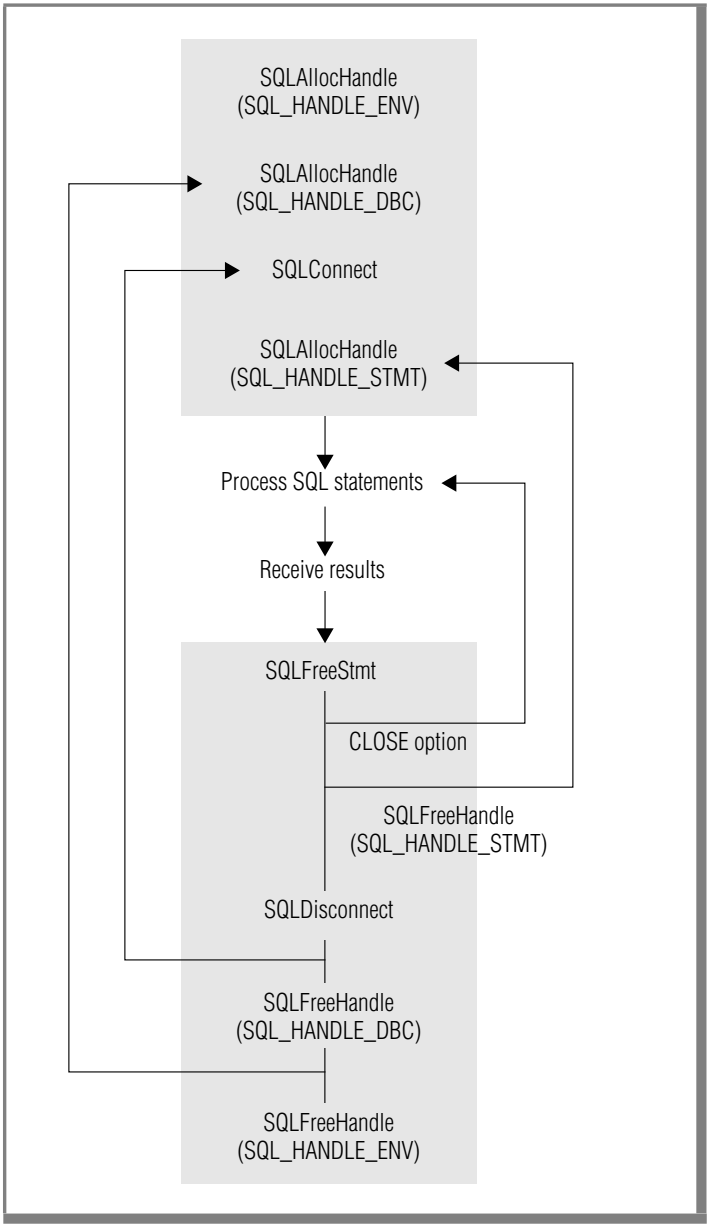


Figure 1-4 *Sample Listing of Function Calls That an IBM Informix ODBC Driver Application Makes*

Environment, Connection, and Statement Handles

When an application requests it, the driver and the driver manager allocate storage for information about the environment, each connection, and each SQL statement. The driver returns a handle for each of these allocations to the application, which uses one or more handles in each call to a function.

The IBM Informix ODBC Driver API uses the following types of handles:

- **Environment handles**. Environment handles identify memory storage for global information, including the valid connection handles and the current active connection handle. The environment handle is an HENV variable type. An application uses one environment handle. It must request this handle before it connects to a data source.
- **Connection handles**. Connection handles identify memory storage for information about particular connections. A connection handle is an HDBC variable type. An application must request a connection handle before it connects to a data source. Each connection handle is associated with the environment handle. However, the environment handle can be associated with multiple connection handles.
- **Statement handles**. Statement handles identify memory storage for information about SQL statements. A statement handle is an HSTMT variable type. An application must request a statement handle before it submits SQL requests. Each statement handle is associated with exactly one connection handle. However, each connection handle can be associated with multiple statement handles.

Buffers

An application passes data to the driver in an input buffer. The driver returns data to the application in an output buffer. The application must allocate memory for both input and output buffers. If the application uses the buffer to retrieve string data, the buffer must contain space for the null termination byte.

Some functions accept pointers to buffers that are used later by other functions. The application must ensure that these pointers remain valid until all applicable functions have used them. For example, the argument *rgbValue* in **SQLBindCol** points to an output buffer where **SQLFetch** returns the data for a column.

Input Buffers

An application passes the address and length of an input buffer to the driver. The length of the buffer must be one of the following values:

- A length greater than or equal to zero

  This value is the actual length of the data in the input buffer. For character data, a length of zero indicates that the data is an empty (zero length) string. A length of zero is different from a null pointer. If the application specifies the length of character data, the character data does not need to be null-terminated.

- SQL_NTS

  This value specifies that a character data value is null-terminated.

- SQL_NULL_DATA

  This value tells the driver to ignore the value in the input buffer and use a NULL data value instead. It is valid only when the input buffer provides the value of a parameter in an SQL statement.

For character data that contains embedded null characters, the operation of IBM Informix ODBC Driver functions is undefined; for maximum interoperability, it is better not to use them. Informix database servers treat null characters as end-of-string markers or as indicators that no more data exists.

Unless it is specifically prohibited in a function description, the address of an input buffer can be a null pointer. In such cases, the value of the corresponding buffer-length argument is ignored. For more information about input buffers, see "Converting Data from SQL to C" on page 3-29.

Output Buffers

An application passes the following arguments to the driver so that the driver can return data in an output buffer:

- The address of the output buffer, to which the driver returns the data

  Unless it is specifically prohibited in a function description, the address of an output buffer can be a null pointer. In such cases, the driver does not return anything in the buffer and, in the absence of other errors, returns SQL_SUCCESS.

  If necessary, the driver converts data before returning it. The driver always null-terminates character data before returning it.

- The length of the buffer

  The driver ignores this value if the returned data has a fixed length in C, as with an integer, real number, or date structure.

- The address of a variable in which the driver returns the length of the data (the length buffer)

  The returned length of the data is SQL_NULL_DATA if the data is a null value in a result set. Otherwise, the returned length of the data is the number of bytes of data that are available to return. If the driver converts the data, the returned length of the data is the number of bytes that remain after the conversion; for character data, it does not include the null-termination byte that the driver adds.

If the output buffer is too small, the driver attempts to truncate the data. If the truncation does not cause a loss of significant data, the driver returns the truncated data in the output buffer, returns the length of the available data (as opposed to the length of the truncated data) in the length buffer, and returns SQL_SUCCESS_WITH_INFO. If the truncation causes a loss of significant data, the driver leaves the output and length buffers untouched and returns SQL_ERROR. The application calls **SQLGetDiagRec** to retrieve information about the truncation or the error.

For more information about output buffers, see "Converting Data from SQL to C" on page 3-29.

SQLGetInfo Argument Implementation

The following table describes the Informix implementation of **SQLGetInfo** arguments for IBM Informix ODBC Driver. For more information on **SQLGetInfo**, see the *Microsoft ODBC 3.0 Programmer's Reference* and "SQLGetInfo" on page 8-59.

Argument Name	Informix Implementation
SQL_ACTIVE_ENVIRONMENTS	IBM Informix driver does not have a limit on number of active environments. Zero is always returned.
SQL_AGGREGATE_FUNCTIONS	IBM Informix driver will return all aggregate functions that the database server supports.
SQL_ASYNC_MODE	IBM Informix driver will return SQL_AM_NONE.
SQL_ATTR_METADATA_ID	Supported for GetInfo and PutInfo
SQL_BATCH_ROW_COUNT	IBM Informix driver will return bitmask zero.
SQL_BATCH_SUPPORT	IBM Informix driver will return bitmask zero.
SQL_CA1_POS_DELETE	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POS_POSITION	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POS_REFRESH	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POS_UPDATE	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POSITIONED_DELETE	A DELETE WHERE CURRENT OF SQL statement is supported when the cursor is a forward-only cursor. (An SQL-92 entry-level-conforming driver will always return this option as supported.)
SQL_CA1_POSITIONED_UPDATE	An UPDATE WHERE CURRENT OF SQL statement is supported when the cursor is a static-only cursor. (An SQL-92 entry-level-conforming driver will always return this option as supported.)

(1 of 3)

Argument Name	Informix Implementation
SQL_CA1_LOCK_NO_CHANGE	A LockType argument of SQL_LOCK_NO_CHANGE is supported in a call to SQLSetPos when the cursor is a static-only cursor.
SQL_CA1_SELECT_FOR_UPDATE	A SELECT FOR UPDATE SQL statement is supported when the cursor is a forward-only cursor. (An SQL-92 entry-level-conforming driver will always return this option as supported.)
SQL_CATALOG_NAME	IBM Informix driver will return 'Y'
SQL_COLLATION_SEQ	INTERSOLV DataDirect ODBC Driver returns InfoValuePtr (unmodified)
SQL_DDL_INDEX	Returns bitmask SQL_DL_CREATE_INDEX | SQL_DL_DROP_INDEX
SQL_DESCRIBE_PARAMETER	Returns 'N'; parameters cannot be described. (This is because the latest Informix database servers support function overloading such that multiple functions with the same name can accept different parameter types.)
SQL_DIAG_DYNAMIC_FUNCTION	Returns empty string
SQL_DROP_TABLE	Returns bitmask SQL_DT_DROP_TABLE | SQL_DT_CASCADE | SQL_DT_RESTRICT
SQL_DROP_VIEW	Returns bitmask SQL_DV_DROP_TABLE | SQL_DV_CASCADE | SQL_DV_RESTRICT
SQL_INDEX_KEYWORDS_	SQL_LLK_ASC | SQL_LK_DESC
SQL_INSERT_STATEMENT	Returns bitmask SQL_IS_INSERT_LITERALS | SQL_ INSERT_SEARCHED | SQL_IS_SELECT_INTO
SQL_MAX_DRIVER_CONNECTIONS	Returns zero
SQL_MAX_IDENTIFIER_LEN	Returns different values, depending on database server capability
SQL_ODBC_INTERFACE_CONFORMANCE	Returns SQL_OIC_CORE
SQL_PARAM_ARRAY_ROW_COUNTS	Returns SQL_PARC_NO_BATCH

Argument Name	Informix Implementation
SQL_PARAM_ARRAY_SELECTS	Returns SQL_PAS_NO_SELECT
SQL_SQL_CONFORMANCE	Returns SQL_OSC_CORE
SQL_SQL92_FOREIGN_KEY_DELETE_RULE	Returns bitmask zero
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	Returns bitmask zero
SQL_SQL92_GRANT	Returns bitmask zero
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	Returns bitmask zero
SQL_SQL92_PREDICATES	Returns bitmask zero
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	Returns bitmask zero
SQL_SQL92_REVOKE	SQL_SR_CASCADE | SQL_SR_RESTRICT
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	Returns bitmask zero
SQL_SQL92_STRING_FUNCTIONS	Returns bitmask zero
SQL_SQL92_VALUE_EXPRESSIONS	Returns bitmask zero
SQL_STANDARD_CLI_CONFORMANCE	Returns bitmask SQL_SCC_XOPEN_CLI_ VERSION1 | SQL_SCC_ISO92_CLI
SQL_STATIC_CURSOR_ATTRIBUTES1	Scrollable only
SQL_STATIC_CURSOR_ATTRIBUTES2	Scrollable only
SQL_XOPEN_CLI_YEAR	Returns string “1995”

(3 of 3)

GLS

Global Language Support

IBM Informix products can support many languages, cultures, and code sets. Global Language Support (GLS) provides support for all language- and culture-specific information. The following table describes how to set the GLS options depending on your platform.

Platform	How to Set GLS Options
UNIX	Specify the GLS options in the **odbc.ini** file. For more information, see "To configure a new user DSN or system DSN" on page 2-18.
Windows	Specify the GLS options in the IBM Informix ODBC Driver DSN Setup dialog box. For more information, see "To configure a new user DSN or system DSN" on page 2-18.

The rest of this section describes the GLS options for IBM Informix ODBC Driver. For more information about GLS and locales, see the *IBM Informix GLS User's Guide*. Please refer to Chapter 9, "Unicode," for detail on Unicode support.

Client Locale

Description:	Locale and codeset that the application runs in
Format:	**locale.codeset@modifier**
odbc.ini field for UNIX:	CLIENT_LOCALE
Default value for UNIX:	**en_us.8859-1**
Default value for Windows:	**en_us.1252**

Database Locale

Description:	Locale and codeset that the database was created in
Format:	**locale.codeset@modifier**
odbc.ini field for UNIX:	DB_LOCALE
Default value for UNIX:	**en_us.8859-1**
Default value for Windows:	**en_us.1252**

Translation Library

Description:	Performs the codeset conversion
Format:	Path to the file for the library. The translation DLL must follow the ODBC standard for translation libraries. For more information, see the *Microsoft ODBC 3.0 Programmer's Reference*.

odbc.ini field for UNIX:	TRANSLATIONDLL
Default value for UNIX:	**$INFORMIXDIR/lib/esql/igo4a304.xx** where *xx* is platform-specific extension for shared library
Default value for Windows:	**igo4n304.dll**

Translation Option

Description:	Option for a non-IBM Informix translation library
Format:	Determined by the vendor
odbc.ini field for UNIX:	TRANSLATION_OPTION
Default value for Windows:	Determined by the vendor

Important: *Do not set this option for an IBM Informix translation library. An IBM Informix translation library determines the translation option based on the client locale and database locale values.*

VMB Character

Description:	Varying multibyte character length reporting option that specifies how to set *pcbValue* when *rgbValue* (the output area) is not large enough for the code-set-converted data. The possible values are: **Estimate**. IBM Informix ODBC Driver makes a worst-case estimate of the storage space needed to return the data. **Exact**. IBM Informix ODBC Driver writes the code-set-converted data to disk until all the data is converted. Because this option can degrade performance, it is recommended that you do not use this option unless your application does not work with **Estimate**. When you use a multibyte code set (in which characters vary in length from 1 to 4 bytes) for either the database or client locale, the length of a character string or simple large object (TEXT) in the database locale does not indicate the length of the string after it is converted to the client locale.
Possible values for UNIX:	`0` = **Estimate** `1` = **Exact**
Possible values for Windows:	**Estimate** **Exact**
odbc.ini field for UNIX:	VMBCHARLENEXACT
Default value for UNIX:	**Estimate**
Default value for Windows:	**Estimate**

X/Open Standard Interface

In addition to the standard ODBC functions, the IBM Informix ODBC Driver also supports the following functions:

- _fninfx_xa_switch: Function for acquiring the xa_switch structure defined by Informix RM
- IFMX_SQLGetXaHenv: Function for obtaining the environment handle associated with an XA Connection
- IFMX_SQLGetXaHdbc: Function for obtaining the database handle associated with an XA Connection
- xa_open function takes an xa_info parameter. TheIBM Informix ODBC Driver uses this xa_info to establish a XA connection

  The format of xa_info is as follows:

  ```
  <appilcationtoken>|<DSN name>
  ```

  The application token is a unique number the application generates for each xa_open request. It must use the same application token as parameter to IFMX_SQLGetXaHenv & IFMX_SQLGetXaHdbc to get the associated environment & database handles.

For information on programming with XA, refer to *TP/XA Programmer's Manual*.

Partially Supported and Unsupported ODBC Features

IBM Informix ODBC Driver supports partial implementation of the following ODBC features:

- Transaction processing
- ODBC cursors
- ODBC bookmarks
- **SQLBulkOperations**

Transaction Processing

IBM Informix ODBC Driver implementation of transaction isolation levels and transaction modes is slightly different from the Microsoft ODBC implementation of these features. The following sections describe the implementation of transaction isolation levels and transaction modes in IBM Informix ODBC Driver.

Transaction Isolation Levels

The following table lists the transaction isolation levels that IBM Informix ODBC Driver supports for these Informix database servers.

Database Servers	Transaction Isolation Levels
Dynamic Server	■ SQL_TXN_READ_COMMITTED ■ SQL_TXN_READ_UNCOMMITTED ■ SQL_TXN_SERIALIZABLE
IBM Informix Extended Parallel Server	■ SQL_TXN_READ_COMMITTED ■ SQL_TXN_READ_UNCOMMITTED ■ SQL_TXN_SERIALIZABLE Cursor stability. To use this isolation level, call **SQLSetConnectOption()** with the *fOption* value set to `1040` and *vParam* set to `1`.

The default transaction isolation level is SQL_TXN_READ_COMMITTED. To change the transaction isolation level, call **SQLSetConnectOption()** with an *fOption* value of SQL_TXN_ISOLATION.

For more information about transaction isolation levels, see the SQL_DEFAULT_TXN_ISOLATION and SQL_TXN_ISOLATION_OPTION descriptions in the *Microsoft ODBC 3.0 Programmer's Reference*.

Transaction Modes

The default transaction mode is auto-commit.

To change the transaction mode to manual commit

1. Enable transaction logging for your database server.

   For information about transaction logging, see your *Administrator's Guide*.
2. Call **SQLSetConnectOption()** with SQL_AUTOCOMMIT set to SQL_AUTOCOMMIT_OFF.

ODBC Cursors

IBM Informix ODBC Driver supports static and forward cursors but not dynamic and keyset-driven cursors.

For more information about cursors, see the*Microsoft ODBC 3.0 Programmer's Reference*.

ODBC Bookmarks

A *bookmark* is a value that identifies a row of data. IBM Informix ODBC Driver supports bookmarks with **SQLFetchScroll** and **SQLExtendedFetch** and does not support them with **SQLBulkOperations**. IBM Informix ODBC Driver supports bookmarks to the following extent:

- Uses only variable length bookmarks.
- SQL_DESC_OCTET_LENGTH is set to 4 for bookmark columns.
- A bookmark is an integer that contains the row number within the row set, starting with 1.

- Bookmarks persist only as long as the cursor remains open.
- **SQLFetchScroll**, using SQL_FETCH_BOOKMARK for the fetch orientation argument, is fully supported.
- **SQLBulkOperations** will not update the bookmark column for SQL_ADD.

For more information about ODBC bookmarks, see the *Microsoft ODBC 3.0 Programmer's Reference*.

SQLBulkOperations

IBM Informix ODBC Driver supports only the SQL_ADD argument of **SQLBulkOperations**.

SQLDescribeParam

SQLDescribeParam is an ODBC API which returns metadata for the parameters of a query.

Prior to IBM Informix ODBC Driver, Version 3.82, this API returned metadata only for parameters in an INSERT or UPDATE statement.Parameters in the WHERE clause of an UPDATE statement required the IFX_UPDDESC environment variable to be set. For parameters in other statement, **SQLDescribeParam** returned the "Driver Does Not Support This Function" error.

IDS

When connecting to IBM Informix Dynamic Server, Version 9.40 with IBM Informix ODBC Driver Version 3.82, **SQLDescribeParam** will provide metadata for parameters of all statement types.

Important: *The SQLDescribeParam API returns "SQL_UNKNOWN" if the API is called to get information about an expression value or parameter embedded inside another procedure.* ♦

Unsupported Microsoft ODBC Driver Features

IBM Informix ODBC Driver does not support implementation of the following ODBC features:

- Asynchronous communication mode
- Concurrency checking
  - SQL_CA2_OPT_ROWVER_CONCURRENCY
  - SQL_CA2_OPT_VALUES_CONCURRENCY
- CONVERT scalar functions
- Cursor simulation features:
  - SQL_CA2_CRC_APPROXIMATE
  - SQL_CA2_CRC_EXACT
  - SQL_CA2_SIMULATE_NON_UNIQUE
  - SQL_CA2_SIMULATE_TRY_UNIQUE
  - SQL_CA2_SIMULATES_UNIQUE
- Dynamic cursor attributes
- Installer DLL
- SQL functions: **SQLSetStmtAttr**

Chapter

2

Configuring Data Sources

In This Chapter

This chapter explains how to configure a data source (DSN) on UNIX and Windows for IBM Informix ODBC Driver, Version 3.82. After you install the driver, you must configure your DSN before you can connect to it.

UNIX

Configuring a DSN on UNIX

This section provides information on driver specifications and DSN specifications, and describes the following DSN configuration files:

- **sqlhosts**
- **odbcinst.ini**
- **odbc.ini**

The configuration files provide information, such as driver attributes, that the driver uses to connect to DSNs. To modify these files, use a text editor. The section also provides examples of driver and DSN specifications.

The sqlhosts File

This file consists of connection information. It contains an entry for each Informix database server. For information about **sqlhosts**, see your *Administrator's Guide*.

The odbcinst.ini File

Installed ODBC drivers use the **odbcinst.ini** sample file, which is located in **$INFORMIXDIR/etc/odbcinst.ini**. To create your **odbcinst.ini** file, copy the **odbcinst.ini** sample file to your home directory as **$HOME/.odbcinst.ini** (note the added dot at the beginning of the filename). This file has entries for all the installed drivers on your computer. Update this file when you install a new driver or a new version of a driver. The following table describes section items in the **$HOME/.odbcinst.ini** file.

Section	Description	Status
ODBC Drivers	List of names of all the installed ODBC drivers	Optional
ODBC Driver Specifications	List of driver attributes and values	Optional

ODBC Drivers

This section provides information on ODBC drivers.

The following example illustrates information about drivers:

```
[ODBC Drivers]
driver_name1=Installed
driver_name2=Installed
⋮
```

The following example illustrates information about installed drivers:

```
[ODBC Drivers]
INFORMIX 2.8 32 BIT=Installed
INFORMIX 3.81 32 BIT=Installed
```

Driver Specifications

Each installed driver has a properties section under the name of the driver. The following example illustrates a driver-specification format:

```
[driver name1]
Driver=driver_library_path
Setup=setup/driver_library_path
APILevel=api_level_supported
ConnectFunctions=connectfunctions
DriverODBCVer=odbc_version
FileUsage=file_usage
SQLLevel=sql_level
⋮
```

The following example illustrates information about driver specifications:

```
[INFORMIX 3.81 32]
Driver=/vobs/tristarm/odbc/iclis09b.so
Setup=/vobs/tristarm/odbc/iclis09b.so
APILevel=1
ConnectFunctions=YYY
DriverODBCVer=03.81
FileUsage=0
SQLLevel=1
```

The following table describes the keywords that are in the driver-specification section.

Keywords	Description	Status
API Level	ODBC interface conformance level that the driver supports 0=None 1=Level 1 supported 2=Level 2 supported	Required
ConnectFunctions	Three-character string that indicates whether the driver supports **SQLConnect**, **SQLDriverConnect**, and **SQLBrowseConnect**	Required
DriverODBCVer	Character string with the version of ODBC that the driver supports	Required
Driver	Driver library path	Required

(1 of 2)

Keywords	Description	Status
FileUsage	Number that indicates how a file-based driver directly treats files in a DSN	Required
Setup	Setup library	Required
SQLLevel	Number that indicates the SQL-92 grammar that the driver supports	Required

(2 of 2)

For a detailed description of the Driver Specification section, see the *Microsoft ODBC 3.0 Programmer's Reference*.

The odbc.ini File

The **odbc.ini** file is a sample data-source configuration information file. For the location of **odbc.ini** file, see the release notes. To create this file, copy **odbc.ini** to your home directory as **$HOME/.odbc.ini** (note the added dot at the beginning of the filename). Every DSN to which your application connects must have an entry in this file. The following table describes the sections in **$HOME/.odbc.ini.**

Section	Description	Status
ODBC Data Sources	This section lists the DSNs and associates them with the name of the driver. You need to provide this section only if you use an ODBC driver manager from a third-party vendor.	Required
Data Source Specification	Each DSN listed in the ODBC Data Sources section has a Data-Source Specification section that describes the DSN.	Required
ODBC	This section lists ODBC tracing options.	Optional

ODBC Data Sources

Each entry in the ODBC Data Sources section lists a DSN and the driver name. The *data_source_name* value is any name that you choose. It is like an envelope that contains all relevant connection information about the DSN.

The following example illustrates an ODBC data-source format:

```
[ODBC Data Sources]
data_source_name=INFORMIX ODBC 3.81 Driver
⋮
```

The following example defines two DSNs called EmpInfo and CustInfo:

```
[ODBC Data Sources]
EmpInfo=IINFORMIX ODBC 3.81 Driver
CustInfo=INFORMIX-CLI 2.8 Driver
```

Data-Source Specification

Each DSN in the data-sources section has a data-source specification section.

The following example illustrates a data-source specification format:

```
[data_source_name]
Driver=driver_path
Description=data_source_description
Database=database_name
LogonID=user_id
pwd=user_password
Server=database_server
CLIENT_LOCALE=application_locale
DB_LOCALE=database_locale
TRANSLATIONDLL=translation_path
CURSORBEHAVIOR=cursor_behavior
DefaultUDTFetchType=default_UDT_Fetch_type
ENABLESCROLLABLECURSORS=enable_scroll_cursors
ENABLEINSERTCURSORS=enable_insert_cursors
OPTIMIZEAUTOCOMMIT=optimize_auto_commit
NEEDODBCTYPESONLY=need_odbc_types_only
OPTOFC=open_fetch_close_optimization
REPORTKEYSETCURSORS=report_keyset_cursors
FETCHBUFFERSIZE=fetchbuffer_size
DESCRIBEDECIMALFLOATPOINT=describe_decimal_as_float
USESERVERDBLOCALE=use_server_dblocale
DONOTUSELVARCHAR=do_not_use_lvarchar
REPORTCHARCOLASWIDECHARCOL=char_col_as_widechar_col
[ODBC]
   UNICODE=unicode_type
```

The following table describes the keywords that are in the data-source specification section and the order that they appear in each section.

Keywords	Description	Status
data_source_name	Data source specified in the Data Sources section	Required
driver_path	Path for the driver Set this value to the complete pathname for the driver library. For more information on the library directory and filenames, see the release notes.	Required
data_source_description	Description of the DSN Configured for a single user or for system users.	Optional
database_name	Database to which the DSN connects by default	Required
user_id	User identification or account name for access to the DSN	Optional
user_password	Password for access to the DSN	Optional
database_server	Informix database server on which *database_name* resides	Required
application_locale (GLS only)	Client locale. Default value: **en_us.8859-1**	Optional
database_locale (GLS only)	Database locale. Default value: **en_us.8859-1**	Optional
translation_path (GLS only)	DLL that performs code-set conversion; default value: **$INFORMIXDIR/lib/esql/ig04a304.xx,** where **xx** represents a platform-specific file extension	Optional
cursor_behavior_name	Flag for cursor behavior when a commit or rollback transaction is called. Possible values are: ■ `0`=close cursor ■ `1`=preserve cursor Default value: `0`	Optional

(1 of 5)

Keywords	Description	Status
default_UDT_Fetch_type	Default UDT fetch type. Default value: SQL_C_BINARY Possible values are: ■ SQL_C_BINARY ■ SQL_C_CHAR	Optional

Enable_scroll_cursors	If this option is activated, the IBM Informix ODBC Driver supports only scrollable, static cursors.
	Available only as a connection option:
	`SQL_INFX_ATTR_ENABLE_SCROLL_CRUSORS`
	or as a connection attribute string:
	`EnableScrollableCursors`
	Default value is: 0 (disabled)
		Optional

Enable_insert_cursors	Reduces the number of network messages sent to and from the server by buffering the inserted rows used with arrays or parameters and insert statements. This option improves the performance of bulk insert operations.
	Available as both a connection and statement option:
	`SQL_INFX_ATTR_ENABLE_INSERT_CURSORS`
	or as a connection attribute string:
	`EnableInsertCursors`
	Default value is: 0
		Optional

optimize_auto_commit	Defers automatic commit operations while cursors remain open. This option can reduce database communication when the application is using non-ANSI logging databases.
	Available as a connection option:
	`SQL_INFX_ATTR_OPTIMIZE_AUTOCOMMIT`
	or as a connection attribute string:
	`OptimizeAutoCommit`
	Default value is: 1 (enabled)
		Optional

(2 of 5)

Keywords	Description	Status

open_fetch_close_optimization	Causes the driver to buffer the open, fetch, and close cursor messages to the server. This option eliminates on or more message cycles when you use **SQLPrepare**, **SQLExecute**, and **SQLFetch** statements to fetch data with a cursor.
	Only available as a connection option:
	`SQL_INFX_ATTR_OPTOFC`
	or as a connection attribute string:
	`OPTOFC`
	Default is: 0 (disabled)
		Optional

Report_keyset_cursors	Causes the driver to report (via **SQLGetInfo**) that is supports forward-only, static, and keyset-driver cursors even though the driver only supports forward-only and static cursors. This option is used to enable dynaset-type functions, such as MicroSoft Visual Basic, which require drivers that support keyset-driven cursors.
	Also available as connection option:
	`SQL_INFX_ATTR_REPORT_KEYSET_CURSORS`
	or as a connection attribute string:
	`ReportKeysetCursors`
	Default is: 0 (disabled)
		Optional

fetchbuffer_size	Size of a fetch buffer in bytes. Available as connection attribute string: `FETCHBUFFERSIZE` Default is: 4096	Optional

(3 of 5)

Keywords	Description	Status
describe_decimal_as_float	Describes all floating-point decimal columns as: ■ Float(SQL_REAL) or ■ Float(SQL_DOUBLE) A floating-point decimal column is a column that was created without a scale, for example DECIMAL(12). Some prepackaged applications such as Visual Basic can not properly format Decimal columns that do not have a fixed scale. To use these applications, you must enable this option or redefine the column with a fixed scale. Enabling this option has the disadvantage that SQL_DECIMAL is an exact numeric data type while SQL_REAL and SQL_DOUBLE are approximate numeric data types. SQL_DECIMAL with a precision or 8 or less will be described as SQLREAL. With a precision greater than 8, it is described as SQL_DOUBLE. Available as connection attribute string: `DESCRIBEDECIMALFLOATPOINT` Default is: 0 (disabled)	Optional
use_server_dblocale	Users server database locale. Available as a connection attribute string: `USERSERVERDBLOCALE` Default is: 0 (disabled)	Optional
do_not_use_lvarchar	If enabled, the **SQLGetTypeInfo** does not report LVARCHAR as a supported type (DATA_TYPE) of SQL_VARCHAR. Some applications will use LVARCHAR instead of VARCHAR, even in columns that are less than 256 bytes. The minimum number of bytes transmitted for LVARCHAR is higher than VARCHAR. A large number of LVARCHAR columns can result in the rowset size exceeding the maximum. *Enable this option only if your SQL_VARCHAR columns are less than 256 bytes.* Available as a connection attribute string: `DONOTUSELVARCHAR` Default is: 0 (disabled)	Optional

Keywords	Description	Status
char_col_as_widechar_col	Causes SQLDescribeCol to report character columns as wide character columns as follows: ■ SQL_CHAR is reported as SQL_WCHAR ■ SQL_VARCHAR is reported as SQL_WVARCHAR ■ SQL_LONGVARCHAR is reported as SQL_WLONGVARCHAR Available as a connection attribute string: `REPORTCHARCOLASWIDECHARCOL` Default is: 0 (disabled)	Optional
unicode_type	Indicates the type of Unicode used by an application. This attribute is applies to UNIX applications only and is set in the ODBC section of the **odbc.ini** file. The following considerations apply: ■ Application on Unix not linking to Data Direct ODBC driver manager should set this to UCS-4 ■ Applications on IBM AIX with version lower than 5L should set this to UCS-2 ■ Applications using Data Direct driver manager do not need to set this attribute. Default is: UTF-8 For more information on using Unicode in an ODBC application, see Chapter 9, "Unicode."	Required

(5 of 5)

The following example shows the configuration for a DSN called EmpInfo:

```
[EmpInfo]
Driver=/informix/lib/cli/iclis09b.so
Description=Demo data source
Database=odbc_demo
LogonID =admin
pwd=tiger
Server=ifmx_91
CLIENT_LOCALE=en_us.8859-1
DB_LOCALE=en_us.8859-1
TRANSLATIONDLL=/opt/informix/lib/esql/igo4a304.so
```

The following example shows the configuration for a DSN called Informix 9:

```
[Informix9]
Driver=/work/informix/lib/cli/iclis09b.so
Description= Informix 9.x ODBC Driver
LogonID=user1
pwd=tigress4
Database=odbc_demo
ServerName=my_server
```

If you specify a null `LogonID` or `pwd`, the following error occurs:

```
Insufficient connect information supplied
```

Tip: *You can establish a connection to a DSN with null values for* `LogonID` *and* `pwd` *if the local Informix database server is on the same computer where the client is located. In this case, the current user is considered a trusted user.*

A sample data source, with no `LogonID` and `pwd`, where the server and client are on the same computer, might look like the following example:

```
Driver=/work/informix/lib/cli/iclis09b.so
Description=Informix 9.x ODBC Driver
LogonID=
pwd=tiger
Database=odbc_demo
ServerName=ifmx_server
```

ODBC Section

The values in the ODBC section of **odbc.ini** specify ODBC tracing options. These options are set through the Tracing tab of the ODBC Data Source Administrator dialog box on Windows.

The following example illustrates an ODBC section specification format:

```
[ODBC]
TRACE=1
TRACEFILE=/WORK/ODBC/ODBC.LOG
TRACEDLL=IDMRS09A.SO
UNICODE=UCS-4
```

Setting the $ODBCINI Environment Variable

By default, IBM Informix ODBC Driver uses configuration information found in the **$HOME/.odbc.ini** file. If you want to provide access to your DSN by system users, modify the path in the **$ODBCINI** environment variable to point to another configuration file that also contains the configuration information found in the **$HOME/.odbc.ini** file. Then change the configuration file permissions to allow read access for system users. Do not change the permissions to the **$HOME/.odbc.ini** file.

In the following example, the configuration filename is **myodbc.ini**:

```
setenv ODBCINI /work/myodbc.ini
```

The .netrc File

The **.netrc** file contains data for logging into a remote database server over the network. For information on the **.netrc** file, see the UNIX man pages. Create the **.netrc** file in the home directory where the client computer initiates the connection. Set the **.netrc** file permissions for the user to deny read access by the group and others.

To connect to a remote database server, create entries in the **.netrc** file for the `LogonID` and `pwd` required to autoconnect to the data source. To establish a connection to a remote data source, the ODBC driver first reads the `LogonID` and `pwd` from the data source entry in the **$HOME/.odbc.ini** file. If the **$HOME/.odbc.ini** file does not specify a `LogonID` and `pwd`, the ODBC driver searches the **$HOME/.netrc** file.

For example, to allow an autologin to the computer called **ray** using the login name **log8in** with password **mypassword**, your **.netrc** file should contain the following line:

```
machine ray login log8in password mypassword
```

Windows

Configuring a DSN in Windows

In Windows environments, IBM Informix ODBC Driver provides a GUI to configure DSNs.

To configure a DSN:

- Choose a procedure to modify your DSN:
  - Choose the User DSN option to restrict access to one user.
  - Choose the System DSN option to restrict access to system users.
  - Choose the File DSN option to allow access to all users on a network.
- Enter DSN-configuration values to create a new DSN, such as the data-source name, the database server name, and the database locale.

For a description of values, see Figure 2-1 on page 2-15 and Figure 2-2 on page 2-16. Values are shown in the order that they appear in each section.

Tip: *To find out what kind of DSN you have, follow the steps in "To reconfigure an existing DSN" on page 2-23, click the* ***General*** *tab and read the contents of the* ***Description*** *text box.*

You can also use Microsoft ODBC, Version 2.5 or later, to configure a DSN.

To configure an existing DSN, see "To reconfigure an existing DSN" on page 2-23.

Figure 2-1
Required DSN Values

Required Values	Description
Data Source Name	DSN to access This value is any name that you choose. **Data Source Name** is like an envelope that contains all relevant connection information about the DSN.
Database Name	Name of the database to which the DSN connects by default
Host Name	Computer on which **Server** resides

(1 of 2)

Required Values	Description
Protocol	Protocol used to communicate with **Server** Once you have added a DSN, the pull-down menu will display the available choices.
Server Name	Informix database server on which **Database** resides
Service	Informix database server process that runs on your **Host** computer Confirm the service name with your system administrator or database administrator.

(2 of 2)

Figure 2-2
Optional DSN Values

Optional Values	Description
Client Locale	Default value: **en_us.1252**
Database Locale	Default value: **en_us.1252**
Description	Any kind of information, such as version number and service
Options	General information, such as password settings For more information on this value, see the sqlhosts information in your *Administrator's Guide*.
Password	Password for access to the DSN
Translation Library	Dynamic linked library (DLL) that performs code-set conversion; default value: **$INFORMIXDIR\bin\ig04n304.dll**

(1 of 2)

Optional Values	Description
User Id	User identification or account name for access to the DSN
Translation Option	Option for a non-IBM Informix translation library Varying multibyte character length reporting option that specifies how to set *pcbValue* when *rgbValue* (the output area) is not large enough for the code-set-converted data Possible values: ■ 0=Estimate ■ 1=Exact Default value: 0
Cursor Behavior	Flag for cursor behavior when a commit or rollback transaction is called Possible values are: ■ 0=close cursor ■ 1=preserve cursor Default value: 0

(2 of 2)

To configure a new user DSN or system DSN

1. Choose **Start→Settings→Control Panel**.
2. Double-click **ODBC** to open the ODBC Data Source Administrator dialog box.

   To configure a user DSN, go to step 3. To configure a system DSN, click the **System DSN** tab and go to step 3. All subsequent steps are the same to configure either a user DSN or a system DSN.
3. Click **Add**.

   The Create New Data Source dialog box appears.
4. Double-click **INFORMIX 3.81 32 BIT** on the Create New Data Source wizard.

   The **General** page for the IBM Informix ODBC Driver Setup dialog box appears.
5. Enter the values in the **General** page, as the following example shows:

   - Data Source Name: `odbc33int`
   - Description: `file DSN 3.81 on turbo`

   For a description of the values, see "Required DSN Values" on page 2-15 and "Optional DSN Values" on page 2-16.

Important: *Do not click* ***OK*** *after you enter the values on this page. If you click* ***OK*** *before you enter all the values, you get an error message.*

6. Click the **Connection** tab to display the **Connection** page and enter the values, as the following example shows:
   - Server Name: `ol_clipper` (or use the pull-down menu to choose a server that is on the sqlhosts registry. If you use the pull-down menu, the ODBC application sets the Host Name, Service, Protocol, and Options values.)
   - Host Name: `clipper`
   - Service: `turbo`
   - Protocol: `olsoctcp` (or use the pull-down menu to choose a protocol)
   - Options: `csm=(SPWDCSM)`
   - Database Name: `odbc_demo` (or use the pull-down menu to find a database name)
   - User Id: *myname*
   - Password: `*******`

     To save the values you chose and verify that your DSN connects successfully, click Apply & Test Connection. An ODBC Message dialog box appears. The box tells you if your connection was successful or, if it was not, tells you which Connection-tab value is incorrect.
7. Click the **Environment** tab to display the **Environment** page and enter the values, as the following example shows:
   - Client Locale: `en_US.CP1252`
   - Database Locale: `en_US.CP1252`
   - Use Server Database Locale: if checkbox is checked, database locale value is set to the server locale. If the checkbox is unchecked, the database locale is set to the default locale, `en_US.CP1252.`
   - Translation Library: `INFORMIXDIR\lib\esql\ig04n304.dll`
   - Translation Option: `0`
   - Cursor Behavior: `0 - Close`
   - VMB Character: `0 - Estimate`
   - Fetch Buffer Size: `4096`

8. Click the **Advanced** tab to display the **Advanced** page and click all applicable boxes.

   - **Auto Commit Optimization**

     This option defers automatic commit operations while cursors remain open and can reduce database communication when the application is using non-ANSI logging databases. This option is available only as a connection option:

     SQL_INFX_ATTR_OPTIMIZE_AUTOCOMMIT

     or as a connection attribute string:

     "OptimizeAutoCommit"

     The default is: `1` (enabled).

   - **Open-Fetch-Close Optimization**

     This option causes the driver to buffer the open, fetch, and close cursor messages to the server. In addition, this option eliminates one or more message round trips when you use **SQLPrepare**, **SQLExecute**, and **SQLFetch** statements to fetch data with a cursor. This option is available only as a connection option:

     SQL_INFX_ATTR_OPTOFC

     or as a connection attribute string:

     "OPTOFC"

     The default is: `0` (disabled).

   - **Insert Cursors**

     This option reduces the number of network messages sent to and from the server by buffering the inserted rows that are used with arrays of parameters and insert statements. This option can greatly improve the performance of bulk insert operations, and is available as both connection and statement options:

     SQL_INFX_ATTR_ENABLE_INSERT_CURSORS.

     or as a connection attribute string:

     "EnableInsertCursors"

     The default is: `0` (disabled).

- **Scrollable Cursor**

  If this option is activated, IBM Informix ODBC Driver, Version 3.82, supports only scrollable, static cursors. This option is available only as a connection option:

  SQL_INFX_ATTR_ENABLE_SCROLL_CURSORS

  or as a connection attribute string:

  "EnableScrollableCursors"

  The default is: 0 (disabled).

- **Report KeySet Cursors**

  This option causes the driver to report (through **SQLGetInfo**) that it supports forward-only, static, and keyset-driven cursor types, although the driver only supports forward-only and static cursors. When you set this option, the driver enables dynaset-type functions, such as Microsoft's Visual Basic. These functions require drivers that support keyset-driven cursor types. This option is also available as a connection attribute:

  SQL_INFX_ATTR_REPORT_KEYSET_CURSORS

  or as a connection attribute string:

  "ReportKeysetCursors"

  The default is: 0 (disabled).

- **Report Standard ODBC Types Only**

  If you activate this feature, the driver causes **SQLGetTypeInfo** to map all occurrences of user-defined types (UDTs) as follows:

Blob	**SQL_LONGVARBINARY**
Clob	**SQL_LONGVARBINARY**
Multiset	**SQL_C_CHAR/SQL_C_BINARY**
Set	**SQL_C_CHAR/SQL_C_BINARY**
List	**SQL_C_CHAR/SQL_C_BINARY**
Row	**SQL_C_CHAR/SQL_C_BINARY**

The driver maps multiset, set, row, and list data types to **SQL_C_CHAR** or **SQL_C_BINARY**, which is the default UDT-FetchType attribute set.

If you enable this feature, you must assign connection and statement attributes to standard ODBC types, LO Automatic, and set DefaultUDTFetchType to **SQL_C_CHAR** features.

The default is: 0 (disabled).

- **Describe decimal floating point as SQL_REAL / SQL_DOUBLE**

  This option describes all floating-point decimal columns as Float (SQL_REAL or SQL_DOUBLE). A floating-point decimal column is a column that was created without a scale, ex: DECIMAL(12). Some prepackaged applications such as Visual Basic can not properly format Decimal columns that do not have a fixed scale. To use these applications you must enable this option or re-define the column with a fixed scale.

  There is a disadvantage to enabling this option however, SQL_DECIMAL is an exact numeric data type while SQL_REAL and SQL_DOUBLE are approximate numeric data types. A SQL_DECIMAL with a precision of 8 or less will be described as SQL_REAL, with a precision greater than 8 it is SQL_DOUBLE.

  The default is: 0 (disabled).

- **Do not use LVARCHAR**

  Causes SQLGetTypeInfo to not report LVARCHAR as a supported type of DATA_TYPE of SQL_VARCHAR.

  Some application such as MS Access97 will use LVARCHAR instead of VARCHAR even for columns that are less than 256 bytes long. The minimum # of bytes transmitted for LVARCHAR is higher than for Varchar and a large number of LVARCHAR columns can result in the rowset size exceeding the maximum. Enable this option only if your SQL_VARCHAR columns are less than 256 bytes in length.

  The default is: 0 (disabled)

- **Report CHAR Columns As Wide CHAR Columns**

  Causes SQLDescribeCol to report char columns as wide char columns. SQL_CHAR column is reported as SQL_WCHAR , SQL_VARCHAR as SQL_WVARCHAR and SQL_LONGVARCHAR column as SQL_WLONGVARCHAR

  The default is: 0 (disabled)

1. To check your connection to the database server, click **Test Connection**.
2. Click **OK** to return to the ODBC Data Source Administrator dialog box and to update the DSN information in the appropriate files.

When your application connects to this DSN, the values that you entered are the default entries for the DSN connection.

To remove a DSN

1. Follow steps 1 and 2 in "To configure a new user DSN or system DSN" on page 2-18.
2. Click **Remove** in the ODBC Data Source Administrator dialog box.

   The 32-bit ODBC Administrator dialog box appears.
3. Click **Yes** to remove the DSN and return to the ODBC Data Source Administrator dialog box.

To reconfigure an existing DSN

1. Follow steps 1 and 2 in "To configure a new user DSN or system DSN" on page 2-18.
2. Click **Configure** to display the IBM Informix ODBC Driver Setup dialog box.

   Enter the new configuration values in the corresponding text boxes and click **OK** to return to the ODBC Data Source Administrator dialog box.

After you complete these steps, you will connect to the DSN.

To configure a file DSN

1. Choose **Start→Settings→Control Panel**.
2. Double-click the ODBC icon to open the ODBC Data Source Administrator dialog box.
3. Click the **File DSN** tab to display the **File DSN** page.

   Choose the File DSN option to allow access to the DSN to all users on a network. For a description of values, see Figure 2-1 and Figure 2-2 on page 2-16.
4. Click **Add**.

   The Create New Data Source wizard appears.
5. Select **INFORMIX 3.81 32 BIT** from the driver list and click **Next** to display the Create New Data Source Setup wizard, which contains a file data source text box.
6. If you know the name of the date source file, type the name into the text box, click **Next** to display the completed Create New Data Source wizard, and go to Step 9.

   If you do not know the name of the file, click **Browse** to display the Save As dialog box and enter the values, as the following example shows:

   - File Name: `File_DSN`
   - Save as type: `ODBC File Data Sources`

   Select a filename or type a filename in the **File_name** text box.
7. Click **Save** to display the Create New Data Source wizard, which displays information about the data source name.
8. Click **Next** to display the completed Create New Data Source wizard.
9. Click **Finish** to display the IBM Informix Connect dialog box. For a description of the values, see Figure 2-1 and Figure 2-2 on page 2-16. For **Advanced** tab values, see page 2-20.
10. Click **OK** to save the values and display the ODBC Data Source Administrator dialog box.

    The name of the data file that you chose or typed in step 6 appears in the text box.

After you add or change DSN-configuration information, the driver updates the appropriate Windows registry to reflect the specified values. To be compatible with other IBM Informix connectivity products, the driver stores the DSN-configuration information in the Windows registry.

To create logs of calls to the drivers

1. Click the **Tracing** tab to display the **Tracing** page, as Figure 2-3 shows.

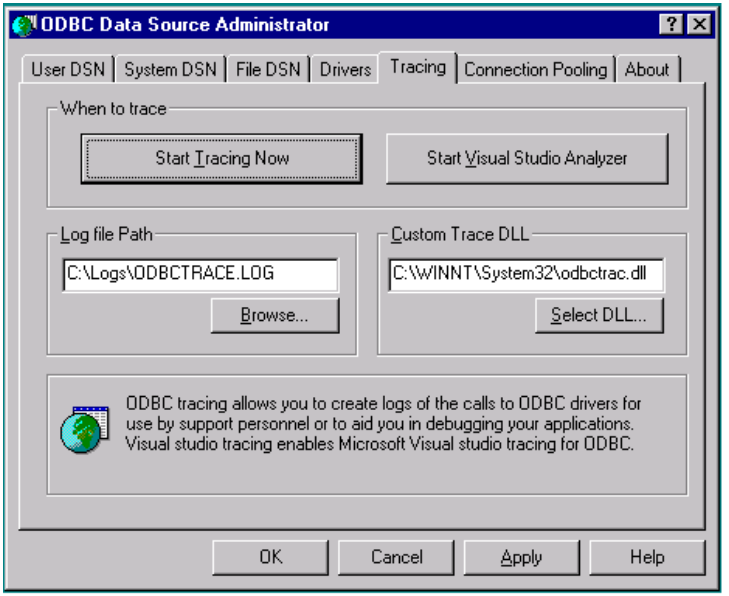


Figure 2-3
Tracing Page

2. Select **Start Tracing Now** to turn tracing on.
3. To enter an existing log file, click **Browse** to display the Select ODBC Trace File dialog box.
4. Enter the filename in the **File_name** text box and click **Save** to return to the **Tracing** page.

5. To select a custom trace dynamic link library (DLL), click **Select DLL** to display the Select a custom trace dll dialog box, and enter the values, as the following example shows:
   - File name: `test2_dsn`
   - Files of type: `Dynamic link libraries(*.dll)`

   Choose a file or type a filename in the **File_name** text box.
6. Click **Open** to display the Tracing page.
7. Click **OK** to save the changes.

After you complete these steps, you will connect to the DSN.

Making a Connection Without DSN

The following table lists the connection string keywords that can be used in making a connection without DSN:

Keyword	Short Versions
DRIVER	DRIVER
DSN	DSN
FILEDSN	FILEDSN
UID	UID
DATABASE	DB
HOST	HOST
PWD	PWD
SERVER	SRVR
SERVICE	SERV
PROTOCOL	PRO
CLIENT_LOCALE	CLOC
DB_LOCALE	DLOC

(1 of 2)

Keyword	Short Versions
TRANSLATIONDLL	TDLL
TRANSLATIONOPTION	TOPT
CONNECTDATABASE	CONDB
EXCLUSIVE	XCL
CURSORBEHAVIOR	CURB
SAVEFILE	SAVEFILE
OPTIONS	OPT
DESCRIPTION	DESC
ENABLESCROLLABLECURSORS	SCUR
ENABLEINSERTCURSORS	ICUR
OPTIMIZEAUTOCOMMIT	OAC
OPTOFC	OPTOFC
NEEDODBCTYPESONLY	ODTYP
REPORTKEYSETCURSORS	RKC
FETCHBUFFERSIZE	FBC
DESCRIBEDECIMALFLOATPOINT	DDFP
DONOTUSELVARCHAR	DNL
REPORTCHARCOLASWIDECHARCOL	RCWC

(2 of 2)

Chapter

3

Data Types

In This Chapter

This chapter discusses the following topics:

- Kinds of data types
- SQL data types
- C data types
- Transferring data
- Converting data

Data Types

The following table describes the data types that IBM Informix ODBC Driver supports.

Data Type	Description	Example
Informix SQL data type	Data types that your Informix database server uses	CHAR(*n*)
Informix ODBC Driver SQL data type	Data types that correspond to the Informix SQL data types	SQL_CHAR
Standard C data type	Data types that your C compiler defines	unsigned char
Informix ODBC Driver typedef	Typedefs that correspond to the standard C data types	UCHAR
Informix ODBC Driver C data type	Data types that correspond to the standard C data types	SQL_C_CHAR

SQL Data Types

For detailed information about the Informix SQL data types, see the *IBM Informix Guide to SQL: Reference*, the *IBM Informix Guide to SQL: Tutorial*, and *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Standard SQL Data Types

The following table lists the standard Informix SQL data types and their corresponding Informix ODBC Driver data types.

Informix SQL Data Type	Informix ODBC Driver SQL Data Type (fSqlType)	Description
BOOLEAN (IDS)	SQL_BIT	't' or 'f'
BYTE	SQL_LONGVARBINARY	Binary data of variable length
CHAR(*n*), CHARACTER(*n*)	SQL_CHAR	Character string of fixed length *n* ($1 \le n \le 32{,}767$)
CHARACTER VARYING(*m*, *r*)	SQL_VARCHAR	Character string of variable length with maximum length *m* ($1 \le m \le 255$) and minimum amount of reserved space *r* ($0 \le r < m$)
DATE	SQL_DATE	Calendar date
DATETIME	SQL_TIMESTAMP	Calendar date and time of day
DEC(*p*, *s*), DECIMAL(*p*, *s*)	SQL_DECIMAL	Signed numeric value with precision *p* and scale *s* ($1 \le p \le 32$; $0 \le s \le p$)
DOUBLE PRECISION	SQL_DOUBLE	Signed numeric value with the same characteristics as the standard C **double** data type
FLOAT	SQL_DOUBLE	Signed numeric value with the same characteristics as the standard C **double** data type

(1 of 4)

Informix SQL Data Type	Informix ODBC Driver SQL Data Type (fSqlType)	Description
INT, INTEGER	SQL_INTEGER	Signed numeric value with precision 10, scale 0, and range *n* (-2,147,483,647 ≤ *n* ≤ 2,147,483,647)
INT8 (IDS)	SQL_BIGINT	Signed numeric value with precision 10, scale 0, and range *n* ($-(2^{63} - 1) \leq n \leq 2^{63} - 1$)
INTERVAL MONTH(*p*)	SQL_INTERVAL_MONTH	Number of months between two dates; *p* is the interval leading precision.
INTERVAL YEAR(*p*)	SQL_INTERVAL_YEAR	Number of years and months between two dates; *p* is the interval leading precision.
INTERVAL YEAR(*p*) TO MONTH	SQL_INTERVAL_YEAR_TO_MONTH	Number of years and months between two dates; *p* is the interval leading precision.
INTERVAL DAY(*p*)	SQL_INTERVAL_DAY	Number of days between two dates; *p* is the interval leading precision.
INTERVAL HOUR(*p*)	SQL_INTERVAL_HOUR	Number of hours between two date times; *p* is the interval leading precision.
INTERVAL MINUTE(*p*)	SQL_INTERVAL_MINUTE	Number of minutes between two date/times; *p* is the interval leading precision.
INTERVAL SECOND(*p*,*q*)	SQL_INTERVAL_SECOND	Number of seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision.
INTERVAL DAY(*p*) TO HOUR	SQL_INTERVAL_DAY_TO_HOUR	Number of days/hours between two date/times; *p* is the interval leading precision.
INTERVAL DAY(*p*) TO MINUTE	SQL_INTERVAL_DAY_TO_MINUTE	Number of days/hours/minutes between two date/times; *p* is the interval leading precision.

(2 of 4)

Informix SQL Data Type	Informix ODBC Driver SQL Data Type (fSqlType)	Description
INTERVAL DAY(*p*) TO SECOND(*q*)	SQL_INTERVAL_DAY_TO_SECOND	Number of days/hours/minutes/seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision.
INTERVAL HOUR TO MINUTE	SQL_INTERVAL_HOUR_TO_MINUTE	Number of hours/minutes between two date/times; *p* is the interval leading precision.
INTERVAL HOUR(*p*) TO SECOND(*q*)	SQL_INTERVAL_HOUR_TO_SECOND	Number of hours/minutes/seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision.
INTERVAL MINUTE(*p*) TO SECOND(*q*)	SQL_INTERVAL_MINUTE_TO_SECOND	Number of minutes/seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision.
LVARCHAR (IDS)	SQL_VARCHAR	Character string of variable length with length *l* ($255 \leq l \leq 2048$)
MONEY(*p*, *s*)	SQL_DECIMAL	Signed numeric value with precision *p* and scale *s* ($1 \leq p \leq 32$; $0 \leq s \leq p$)
NUMERIC	SQL_NUMERIC	Signed, exact, numeric value with precision *p* and scale *s* ($1 \leq p \leq 15$; $0 \leq s \leq p$)
REAL	SQL_REAL	Signed numeric value with the same characteristics as the standard C **float** data type
SERIAL	SQL_INTEGER	Sequential INTEGER
SERIAL8 (IDS)	SQL_BIGINT	Sequential INT8
SMALLFLOAT	SQL_REAL	Signed numeric value with the same characteristics as the standard C **float** data type

(3 of 4)

Informix SQL Data Type	Informix ODBC Driver SQL Data Type (fSqlType)	Description
SMALLINT	SQL_SMALLINT	Signed numeric value with precision 5, scale 0, and range n ($-32{,}767 \le n \le 32{,}767$)
TEXT	SQL_LONGVARCHAR	Character string of variable length
VARCHAR(m, r)	SQL_VARCHAR	Character string of variable length with maximum length m ($1 \le m \le 255$) and minimum amount of reserved space r ($0 \le r < m$)

(4 of 4)

GLS

Additional SQL Data Types for GLS

The following table lists the additional Informix SQL data types for GLS and their corresponding Informix ODBC Driver data types. IBM Informix ODBC Driver does not provide full GLS support. For more information about GLS, see the *IBM Informix GLS User's Guide*.

Informix SQL Data Type	Informix ODBC Driver SQL Data Type (fSqlType)	Description
NCHAR(n)	SQL_CHAR	Character string of fixed length n ($1 \le n \le 32{,}767$). Collation depends on locale.
NVARCHAR(m, r)	SQL_VARCHAR	Character string of variable length with maximum length m ($1 \le m \le 255$) and minimum amount of reserved space r ($0 \le r < m$). Collation depends on locale.

IDS

Additional SQL Data Types for Dynamic Server

The following table lists the additional Informix SQL data types for Dynamic Server and their corresponding Informix ODBC Driver data types. To use the Informix ODBC Driver SQL data types for Dynamic Server, include **infxcli.h.**

Informix SQL Data Type	Informix ODBC Driver SQL Data Type (fSqlType)	Description
Collection (LIST, MULTISET, SET)	Any Informix ODBC Driver SQL data type	Composite value that consists of one or more elements, where each element has the same data type.
DISTINCT	Any Informix ODBC Driver SQL data type	UDT that is stored the same way as its source data type but has different casts and functions
OPAQUE (fixed)	SQL_INFX_UDT_FIXED	Fixed-length UDT with an internal structure that has the same size for all possible values
OPAQUE (varying)	SQL_INFX_UDT_VARYING	Variable-length UDT with an internal structure that can have a different size for each different value
Row (Named row, unnamed row)	Any Informix ODBC Driver SQL data type	Composite value that consists of one or more elements, where each element can have a different data type. For more information, see Chapter 5.
Smart large object (BLOB or CLOB)	SQL_IFMX_UDT_BLOB SQL_IFMX_UDT_CLOB	Large object that is stored in an sbspace on disk and is recoverable. For more information, see Chapter 4.

Precision, Scale, Length, and Display Size

The functions that get and set precision, scale, length, and display size for SQL values have size limitations for their input arguments. Therefore, these values are limited to the size of an SDWORD that has a maximum value of 2,147,483,647. The following table describes these values.

Value	Description for a Numeric Data Type	Description for a Non-Numeric Data Type
Precision	Maximum number of digits.	Either the maximum length or the specified length.
Scale	Maximum number of digits to the right of the decimal point. For floating point values, the scale is undefined because the number of digits to the right of the decimal point is not fixed.	Not applicable.
Length	Maximum number of bytes that a function returns when a value is transferred to its default C data type.	Maximum number of bytes that a function returns when a value is transferred to its default C data type. The length does not include the NULL termination byte.
Display size	Maximum number of bytes needed to display data in character form.	Maximum number of bytes needed to display data in character form.

Standard SQL Data Types

The following table describes the precision, scale, length, and display size for the standard Informix ODBC Driver SQL data types.

Informix ODBC Driver SQL Data Type (fSqlType)	Description
SQL_BIGINT (IDS)	**Precision**: 19. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: 0. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 8 bytes. **Display size**: 20 digits. One digit is for the sign.
SQL_BIT (IDS)	**Precision**: 1. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: 0. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 1 byte. **Display size**: 1 digit.
SQL_CHAR	**Precision**: Same as the length. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: The specified length. For example, the length of CHAR(10) is 10 bytes. **Display size**: Same as the length.
SQL_DATE	**Precision**: 10. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 6 bytes. **Display size**: 10 digits in the format yyyy-mm-dd.

(1 of 4)

Informix ODBC Driver SQL Data Type (fSqlType)	Description
SQL_DECIMAL	**Precision**: The specified precision. For example, the precision of DECIMAL (12, 3) is 12. **Scale**: The specified scale. For example, the scale of DECIMAL(12, 3) is 3. **Length**: The specified precision plus 2. For example, the length of DECIMAL(12, 3) is 14 bytes. The two additional bytes are used for the sign and the decimal points because functions return this data type as a character string. **Display size**: Same as the length.
SQL_DOUBLE	**Precision**: 15. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 8 bytes. **Display size**: 22 digits. The digits are for a sign, 15 numeric characters, a decimal point, the letter E, another sign, and 2 more numeric characters.
SQL_INTEGER	**Precision**: 10. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: 0. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 4 bytes. **Display size**: 11 digits. One digit is for the sign.
SQL_LONGVARBINARY	**Precision**: Same as the length. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: The maximum length. If a function cannot determine the maximum length, it returns SQL_NO_TOTAL. **Display size**: The maximum length times 2. If a function cannot determine the maximum length, it returns SQL_NO_TOTAL.

(2 of 4)

Informix ODBC Driver SQL Data Type (fSqlType)	Description
SQL_LONGVARCHAR	**Precision**: Same as the length. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: The maximum length. If a function cannot determine the maximum length, it returns SQL_NO_TOTAL. **Display size**: Same as the length.
SQL_REAL	**Precision**: 7. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 4 bytes. **Display size**: 13 digits. The digits are for a sign, 7 numeric characters, a decimal point, the letter E, another sign, and 2 more numeric characters.
SQL_SMALLINT	**Precision**: 5. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: 0. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: 2 bytes. **Display size**: 6 digits. One digit is for the sign.

Informix ODBC Driver SQL Data Type (fSqlType)	Description
SQL_TIMESTAMP	**Precision**: 8. **SQLBindParameter** ignores the value of *cbColDef* for this data type. **Scale**: The number of digits in the FRACTION field. **Length**: 16 bytes. **Display size**: 19 or more digits: ■ If the scale of the time stamp is 0: 19 digits in the format yyyy-mm-dd hh:mm:ss. ■ If the scale of the time stamp exceeds 0: 20 digits plus digits for the FRACTION field in the format yyyy-mm-dd hh:mm:ss.f...
SQL_VARCHAR	**Precision**: Same as the length. **Scale**: Not applicable. **SQLBindParameter** ignores the value of *ibScale* for this data type. **Length**: The specified length. For example, the length of VARCHAR(10) is 10 bytes. **Display size**: Same as the length.

(4 of 4)

IDS

Additional SQL Data Types for Dynamic Server

The following table describes the precision, scale, length, and display size for the Informix ODBC Driver SQL data types for Dynamic Server.

Informix ODBC Driver SQL Data Type (fSqlType)	Description
SQL_IFMX_UDT_BLOB	**Precision**: Variable value. To determine this value, call a function that returns the precision for a column. **Scale**: Not applicable. A function that returns the scale for a column returns -1 for this data type. **Length**: Variable value. To determine this value, call a function that returns the length for a column. **Display Size**: Variable value. To determine this value, call a function that returns the display size for a column.
SQL_IFMX_UDT_CLOB	**Precision**: Variable value. To determine this value, call a function that returns the precision for a column. **Scale**: Not applicable. A function that returns the scale for a column returns -1 for this data type. **Length**: Variable value. To determine this value, call a function that returns the length for a column. **Display Size**: Variable value. To determine this value, call a function that returns the display size for a column.

(1 of 2)

Informix ODBC Driver SQL Data Type (fSqlType)	Description
SQL_INFX_UDT_FIXED	**Precision**: Variable value. To determine this value, call a function that returns the precision for a column. **Scale**: Not applicable. A function that returns the scale for a column returns -1 for this data type. **Length**: Variable value. To determine this value, call a function that returns the length for a column. **Display Size**: Variable value. To determine this value, call a function that returns the display size for a column.
SQL_INFX_UDT_VARYING	**Precision**: Variable value. To determine this value, call a function that returns the precision for a column. **Scale**: Not applicable. A function that returns the scale for a column returns -1 for this data type. **Length**: Variable value. To determine this value, call a function that returns the length for a column. **Display Size**: Variable value. To determine this value, call a function that returns the display size for a column.

(2 of 2)

C Data Types

An IBM Informix ODBC Driver application uses C data types to store values that the application processes. The following table describes the C data types that IBM Informix ODBC Driver provides.

Important: *String arguments in IBM Informix ODBC Driver functions are unsigned. Therefore, you need to cast a CString object as an unsigned string before you use it as an argument in an IBM Informix ODBC Driver function.*

Kind of Value	Informix ODBC Driver C Data Type (fCType)	Informix ODBC Driver Typedef	Standard C Data Type
Binary	**SQL_C_BINARY**	UCHAR FAR *	**unsigned char FAR ***
Boolean (IDS)	**SQL_C_BIT**	UCHAR	**unsigned char**
Character	**SQL_C_CHAR**	UCHAR FAR *	**unsigned char FAR ***
Wide Character	**SQL_C_WCHAR**	WCHAR FAR *	**wchar_t FAR ***

Date	**SQL_C_DATE**	DATE_STRUCT	`struct tagDATE_STRUCT {`
			`  SWORD year;`
			`  UWORD month;`
			`  UWORD day;`
			`}`

Interval	**SQL_C_INTERVAL_YEAR**	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_MONTH	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_DAY	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_HOUR	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_MINUTE	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_SECOND	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_INTERVAL_STRUCT	**C Interval Structure**

(1 of 3)

Kind of Value	Informix ODBC Driver C Data Type (fCType)	Informix ODBC Driver Typedef	Standard C Data Type
	SQL_C_INTERVAL_DAY_ TO_HOUR	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_DAY _TO_MINUTE	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_DAY _TO_SECOND	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_HOUR _TO_MINUTE	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_HOUR _TO_SECOND	SQL_INTERVAL_STRUCT	**C Interval Structure**
	SQL_C_INTERVAL_MINUTE _TO_SECOND	SQL_INTERVAL_STRUCT	**C Interval Structure**
Numeric	**SQL_C_DOUBLE**	SDOUBLE	**signed double**
	SQL_C_FLOAT	SFLOAT	**signed float**
	SQL_C_LONG	SDWORD	**signed long int**
	SQL_C_NUMERIC	SQL_NUMERIC_STRUCT	`struct tag SQL_NUMERIC_STRUCT { SQLCHAR precision; SQLSCHAR scale; SQLCHAR sign; SQLCHAR val [SQL_MAX_ NUMERIC_LEN]; }SQL_NUMERIC_STRUCT;`
	SQL_C_SHORT	SWORD	**signed short int**
	SQL_C_SLONG	SDWORD	**signed long int**
	SQL_C_SSHORT	SWORD	**signed short int**
	SQL_C_STINYINT	SCHAR	**signed char**

Kind of Value	Informix ODBC Driver C Data Type (fCType)	Informix ODBC Driver Typedef	Standard C Data Type
	SQL_C_TINYINT	SCHAR	**signed char**
	SQL_C_ULONG	UDWORD	**unsigned long int**
	SQL_C_USHORT	UWORD	**unsigned short int**
	SQL_C_UTINYINT	UCHAR	**unsigned char**

Time stamp	**SQL_C_TIMESTAMP**	TIMESTAMP_STRUCT	`struct tagTIMESTAMP_STRUCT {`
			`  SWORD  year;`
			`  UWORD  month;`
			`  UWORD  day;`
			`  UWORD  hour;`
			`  UWORD  minute;`
			`  UWORD  second;`
			`  UDWORD fraction;`
			`}`

(3 of 3)

C Interval Structure

The following structures specify the C data type for the SQL interval data type:

```
typedef struct tagSQL_INTERVAL_STRUCT
   {
   SQLINTERVAL interval_type;
   SQLSMALLINTinterval_sign;
   union
      {
      SQL_YEAR_MONTH_STRUCT year_month;
      SQL_DAY_SECOND_STRUCT day_second;
      } intval;
   }SQLINTERVAL_STRUCT;

typedef enum
   {
   SQL_IS_YEAR=1,
   SQL_IS_MONTH=2,
   SQL_IS_DAY=3,
   SQL_IS_HOUR=4,
   SQL_IS_MINUTE=5,
   SQL_IS_SECOND=6,
   SQL_IS_YEAR_TO_MONTH=7,
   SQL_IS_DAY_TO_HOUR=8,
   SQL_IS_DAY_TO_MINUTE=9,
   SQL_IS_DAY_TO_SECOND=10,
   SQL_IS_HOUR_TO_MINUTE=11,
   SQL_IS_HOUR_TO_SECOND=12,
   SQL_IS_MINUTE_TO_SECOND=13,
   }SQLINTERVAL;

typedef struct tagSQL_YEAR_MONTH
   {
   SQLUINTEGER year;
   SQLUINTEGER month;
   }SQL_YEAR_MOHTH_STRUCT;

typedef struct tagSQL_DAY_SECOND
   {
   SQLUINTEGER day;
   SQLUNINTEGER hour;
   SQLUINTEGER minute;
   SQLUINTEGER second;
   SQLUINTEGER fraction;
   }SQL_DAY_SECOND_STRUCT;
```

Transferring Data

Among data sources that use the same DBMS, you can safely transfer data in the internal form that a DBMS uses. For a particular piece of data, the SQL data types must be the same in the source and target data sources. The C data type is **SQL_C_BINARY**.

When you call **SQLFetch**, **SQLExtendedFetch**, or **SQLGetData** to retrieve this kind of data from a data source, IBM Informix ODBC Driver retrieves the data and transfers it, without conversion, to a storage location of type **SQL_C_BINARY**. When you call **SQLExecute**, **SQLExecDirect**, or **SQLPutData** to send this kind of data to a target data source, IBM Informix ODBC Driver retrieves the data from the storage location and transfers it, without conversion, to the target data source.

IDS

The binary representation of INT8 and SERIAL8 is an array of two unsigned long integers followed by a short integer that indicates the sign field. The sign field is `1` for a positive value, `-1` for a negative value, or `0` for a null value. ♦

Important: *Applications that transfer any data (except binary data) in this manner are not interoperable among DBMSs.*

Reporting Standard ODBC Types

IBM Informix ODBC Driver supports existing applications that support standard ODBC data types only. You should check the DSN option **Report Standard ODBC Types** to turn on this behavior. When an application sets this option, the driver sets the following behavior:

- Only Standard ODBC data types are reported for all the driver defined new data types.
- The data type access method for smart-large-object (LO) data can be accessed as SQL_LONGVARCHAR and SQL_LONGVARBINARY. In other words, SQL_LONGVARCHAR and SQL_LONGVARBINARY act like the simple large objects, byte and text.
- The defaultUDTfetchtype is set to **SQL_C_CHAR**.

However, you can control each of the preceding behaviors individually as a connection or a statement level option. Use the following connection and statement level attributes:

- SQL_INFX_ATTR_ODBC_TYPES_ONLY
- SQL_INFX_ATTR_LO_AUTOMATIC
- SQL_INFX_ATTR_DEFAULT_UDT_FETCH_TYPE

Applications can use SQLSetConnectAttr and SQLSetStmtAttr to set and unset these values. (ODBC 2.x applications can use SQLSetConnectOption and SQLSetStmtOption equivalently.)

SQL_INFX_ATTR_ODBC_TYPES_ONLY

Applications can set this attribute to value SQL_TRUE or SQL_FALSE. This attribute can be set and unset at connection and statement level. All the statements allocated under the same connection inherit this value. Alternatively each statement can change this attribute. By default this attribute is set to SQL_FALSE.

An application can change the value of this attribute using SQLSetConnectAttr and SQLSetStmtAttr (SQLSetConnectOption and SQLSetStmtOption in ODBC 2.x). Applications can retrieve the values set using SQLGetConnectAttr and SQLGetStmtAttr (SQLGetConnectOption and SQLGetStmtOption in ODBC 2.x).

This attribute cannot be set to SQL_TRUE when SQL_INFX_ATTR_LO_AUTOMATIC is set SQL_FALSE. An error message is returned that reports the following message: Attribute cannot be set. LoAutomatic should be ON to set this value.

The application should first set the SQL_INFX_ATTR_LO_AUTOMATIC attribute to SQL_TRUE and then set the attribute SQL_INFX_ATTR_ODBC_TYPES_ONLY to SQL_TRUE.

SQL_INFX_ATTR_LO_AUTOMATIC

Applications can set this attribute to value SQL_TRUE or SQL_FALSE. This attribute can be set and unset at connection and statement level. All the statements allocated under the same connection inherit this value. Alternatively each statement can change this attribute. By default this attribute is set to SQL_FALSE.

An application can change the value of this attribute using SQLSetConnectAttr and SQLSetStmtAttr (SQLSetConnectOption and SQLSetStmtOption in ODBC 2.x). Applications can retrieve the values set using SQLGetConnectAttr and SQLGetStmtAttr (SQLGetConnectOption and SQLGetStmtOption in ODBC 2.x).

The attribute SQL_INFX_ATTR_LO_AUTOMATIC cannot be set to SQL_FALSE when SQL_INFX_ATTR_ODBC_TYPES_ONLY is set to SQL_TRUE. An error message is returned that reports the following message: Attribute cannot be set. ODBC types only should be OFF to set this value.

Applications should first set the attribute SQL_INFX_ODBC_TYPES_ONLY to SQL_FALSE and then set the attribute SQL_INFX_ATTR_LO_AUTOMATIC to SQL_FALSE.

SQL_INFX_ATTR_DEFAULT_UDT_FETCH_TYPE

Applications can set this attribute to **SQL_C_CHAR** or **SQL_C_BINARY** to set the default fetch type for UDTs. The default value of this attribute is set depending on the following conditions:

- If the DSN setting for **Report Standard ODBC Types** is ON, the value of DefaultUDTFetchType is set to **SQL_C_CHAR**.
- If the DSN setting for **Report Standard ODBC Types** is OFF, the value of DefaultUDTFetchType is set to **SQL_C_BINARY**.
- If a user has set a registry key, the value of DefaultUDTFetchType is set to the value in the registry provided **Report Standard ODBC Types** is not set.

An application can change the value of this attribute using SQLSetConnectAttr and SQLSetStmtAttr (SQLSetConnectOption and SQLSetStmtOption in ODBC 2.x). Applications can retrieve the values set using SQLGetConnectAttr and SQLGetStmtAttr (SQLGetConnectOption and SQLGetStmtOption in ODBC 2.x).

Setting the **Report ODBC Types** to ON always overrides DefaultUDTFetchType to **SQL_C_CHAR**.

Reporting Wide Character Columns

Informix servers do not support wide character data types. When an application sets the **Report Char Columns as Wide Char Columns** option, the driver sets the following behavior:

- SQLDescribeCol reports char columns as wide char columns
- SQL_CHAR column is reported as SQL_WCHAR
- SQL_VARCHAR column is reported as SQL_WVARCHAR
- SQL_LONGVARCHAR column is reported as SQL_WLONGVARCHAR
- The default is 0: (disabled)

After setting the **Report Char Columns as Wide Char Columns** option, calls to SQLBindParameter with SQL data types have the following behavior:

- SQL_WCHAR is mapped to SQL_CHAR
- SQL_WVARCHAR is mapped to SQL_VARCHAR
- SQL_WLONGVARCHAR is mapped to SQL_LONGVARCHAR

DSN Settings for Report Standard ODBC Data Types

UNIX

Add a new DSN option "NeedODBCTypesOnly" under your DSN setting in your **.odbc.ini** file [default is 0]. For example:

```
[Informix9]
Driver=/informix/lib/cli/libthcli.so
Description=Informix ODBC 3.81 Driver
….
NeedODBCTypesOnly=1
```

♦

Windows

Check this option under the **Advanced** tab of the ODBC Administration for IBM Informix Driver DSN [default is 0]. ♦

The following table shows how the INFORMIX 9 data types map to the standard ODBC data types. These types are in addition to the Informix data types.

INFORMIX 9	ODBC
Blob	SQL_LONGVARBINARY
Boolean	SQL_BIT
Clob	SQL_LONGVARCHAR
Int8	SQL_BIGINT
Lvarchar	SQL_VARCHAR
Serial8	SQL_BIGINT
Multiset	**SQL_C_CHAR/SQL_C_BINARY**
Set	**SQL_C_CHAR/SQL_C_BINARY**
List	**SQL_C_CHAR/SQL_C_BINARY**
Row	**SQL_C_CHAR/SQL_C_BINARY**

Important: *For multiset, set, row, and list data types, the data type is mapped to the defaultUDTFetchType attribute set (**SQL_C_CHAR** or **SQL_C_BINARY**).*

Converting Data

The word *convert* is used in this section in a broad sense; it includes the transfer of data from one storage location to another without a conversion in data type.

Standard Conversions

The following table shows the supported conversions between the Informix SQL data types and the Informix ODBC Driver C data types. A solid circle (●) indicates a supported conversion.

IDS

Only Dynamic Server can convert data to **SQL_C_BIT**. ♦

	Informix ODBC Driver C Data Type (Target Type)																		
Informix SQL Data Type	**SQL_C_BINARY**	**SQL_C_BIT**	**SQL_C_CHAR**	**SQL_C_WCHAR**	**SQL_C_DATE**	**SQL_C_DOUBLE**	**SQL_C_FLOAT**	**SQL_C_LONG**	**SQL_C_NUMERIC**	**SQL_C_SHORT**	**SQL_C_SLONG**	**SQL_C_SSHORT**	**SQL_C_STINYINT**	**SQL_C_TIMESTAMP**	**SQL_C_TINYINT**	**SQL_C_ULONG**	**SQL_C_USHORT**	**SQL_C_UTINYINT**
BOOLEAN (IDS)	●	●	●	●														
BYTE	●		●	●														
CHAR, CHARACTER	●	●	●	●		●	●	●	●	●	●	●	●		●	●	●	●
CHARACTER VARYING	●	●	●	●		●	●	●	●	●	●	●	●		●	●	●	●
DATE	●		●	●	●									●				
DATETIME	●		●	●	●									●				
DEC, DECIMAL	●	●	●	●		●	●	●	●	●	●	●	●		●	●	●	●
DOUBLE PRECISION	●		●	●		●	●	●	●	●	●	●	●		●	●	●	●

(1 of 2)

Informix SQL Data Type	Informix ODBC Driver C Data Type (Target Type)																	
	SQL_C_BINARY	SQL_C_BIT	SQL_C_CHAR	SQL_C_WCHAR	SQL_C_DATE	SQL_C_DOUBLE	SQL_C_FLOAT	SQL_C_LONG	SQL_C_NUMERIC	SQL_C_SHORT	SQL_C_SLONG	SQL_C_SSHORT	SQL_C_STINYINT	SQL_C_TIMESTAMP	SQL_C_TINYINT	SQL_C_ULONG	SQL_C_USHORT	SQL_C_UTINYINT
FLOAT	●		●	●		●	●	●	●	●	●	●	●		●	●	●	●
INT, INTEGER	●	●	●	●		●	●	●	●	●	●	●	●		●	●	●	●
INT8 (IDS)	●		●	●				●	●		●					●		
LVARCHAR (IDS)	●	●	●	●	●		●	●		●	●	●	●	●	●	●	●	●
MONEY	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
NUMERIC	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
REAL	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
SERIAL	●		●	●	●	●	●	●		●	●	●	●	●	●	●	●	●
SERIAL8 (IDS)	●		●	●				●	●		●					●		
SMALLFLOAT	●	●	●	●		●	●	●	●	●	●	●	●		●	●	●	●
SMALLINT	●	●	●	●		●	●	●	●	●	●	●	●		●	●	●	●
TEXT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
VARCHAR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

(2 of 2)

GLS

Additional Conversions for GLS

The following table shows the supported conversions between the additional Informix SQL data types for GLS and the Informix ODBC Driver C data types. A solid circle (●) indicates a supported conversion.

IDS

Only Dynamic Server can convert data to **SQL_C_BIT**. ♦

	Informix ODBC Driver C Data Type (fCType)															
Informix SQL Data Type	SQL_C_BINARY	SQL_C_BIT	SQL_C_CHAR	SQL_C_DATE	SQL_C_DOUBLE	SQL_C_FLOAT	SQL_C_LONG	SQL_C_SHORT	SQL_C_SLONG	SQL_C_SSHORT	SQL_C_STINYINT	SQL_C_TIMESTAMP	SQL_C_TINYINT	SQL_C_ULONG	SQL_C_USHORT	SQL_C_UTINYINT
NCHAR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
NVARCHAR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

IDS

Additional Conversions for Dynamic Server

The following table shows the supported conversions between the additional Informix SQL data types for Dynamic Server and the Informix ODBC Driver C data types, including the additional Informix ODBC Driver C data type for Dynamic Server. A solid circle (●) indicates a supported conversion.

	Informix ODBC Driver C Data Type (fCType)															
Informix SQL Data Type	**SQL_C_BINARY**	**SQL_C_BIT**	**SQL_C_CHAR**	**SQL_C_DATE**	**SQL_C_DOUBLE**	**SQL_C_FLOAT**	**SQL_C_LONG**	**SQL_C_SHORT**	**SQL_C_SLONG**	**SQL_C_SSHORT**	**SQL_C_STINYINT**	**SQL_C_TIMESTAMP**	**SQL_C_TINYINT**	**SQL_C_ULONG**	**SQL_C_USHORT**	**SQL_C_UTINYINT**
Collection	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
DISTINCT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
OPAQUE [a]	●		●													
Row	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Smart large object	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

[a] Use **SQL_C_CHAR** to access an OPAQUE value in the external format as a string. Use **SQL_C_BINARY** to access an OPAQUE value in the internal binary format.

Converting Data from SQL to C

When you call **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData**, IBM Informix ODBC Driver retrieves data from a data source. If necessary, IBM Informix ODBC Driver converts the data from the source data type to the data type that the *TargetType* argument in **SQLBindCol** or the *fCType* argument in **SQLGetData** specifies. Finally, IBM Informix ODBC Driver stores the data in the location pointed to by the *rgbValue* argument in **SQLBindCol** or **SQLGetData**.

The tables in the following sections describe how IBM Informix ODBC Driver converts data that it retrieves from a data source. For a given Informix ODBC Driver SQL data type, the first column of the table lists the legal input values of the *TargetType* argument in **SQLBindCol** and the *fCType* argument in **SQLGetData**. The second column lists the outcomes of a test, often using the *cbValueMax* argument specified in **SQLBindCol** or **SQLGetData**, which IBM Informix ODBC Driver performs to determine whether it can convert the data. For each outcome, the third and fourth columns list the values of the *rgbValue* and *pcbValue* arguments specified in **SQLBindCol** or **SQLGetData** after IBM Informix ODBC Driver tries to convert the data.

The last column lists the SQLSTATE returned for each outcome by **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData**.

If the *TargetType* argument in **SQLBindCol** or the *fCType* argument in **SQLGetData** contains a value for an Informix ODBC Driver C data type that is not shown in the table for a given Informix ODBC Driver SQL data type, **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData** returns SQLSTATE 07006 (Restricted data type attribute violation). If the *fCType* argument or the *TargetType* argument contains a value that specifies a conversion from a driver-specific SQL data type to an Informix ODBC Driver C data type and IBM Informix ODBC Driver does not support this conversion, then **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData** returns SQLSTATE S1C00 (Driver not capable).

Although the tables in this chapter do not show it, the *pcbValue* argument contains SQL_NULL_DATA when the SQL data value is null. When IBM Informix ODBC Driver converts SQL data to character C data, the character count returned in *pcbValue* does not include the null-termination byte. If *rgbValue* is a null pointer, **SQLBindCol** or **SQLGetData** returns SQLSTATE S1009 (Invalid argument value).

The following terms and conventions are used in the tables:

- *Length of data* is the number of bytes of C data that are available to return in *rgbValue*, regardless of whether or not the data is truncated before it returns to the application. For string data, this does not include the null-termination byte.
- *Display size* is the total number of bytes that are needed to display the data in character format.
- Words in *italics* represent function arguments or elements of the Informix ODBC Driver SQL grammar.

Default C Data Types

If you specify **SQL_C_DEFAULT** for the *TargetType* argument in **SQLBindCol**, the *fCType* argument in **SQLGetData**, or the *ValueType* argument in **SQLBindParameter**, IBM Informix ODBC Driver uses the C data type of the output or input buffer for the SQL data type of the column or parameter to which the buffer is bound.

Standard Default C Data Types

For each Informix ODBC Driver SQL data type, the following table shows the default C data type.

Informix ODBC Driver SQL Data Type (fSqlType)	Default Informix ODBC Driver C Data Type (fCType)
SQL_BIGINT (IDS)	**SQL_C_CHAR**
SQL_BIT (IDS)	**SQL_C_BITS**
SQL_CHAR	**SQL_C_CHAR**
SQL_DATE	**SQL_C_DATE**
SQL_DECIMAL	**SQL_C_CHAR**
SQL_DOUBLE	**SQL_C_DOUBLE**
SQL_INTEGER	**SQL_C_SLONG**
SQL_LONGVARBINARY	**SQL_C_BINARY**

(1 of 2)

Informix ODBC Driver SQL Data Type (fSqlType)	Default Informix ODBC Driver C Data Type (fCType)
SQL_LONGVARCHAR	**SQL_C_CHAR**
SQL_NUMERIC	**SQL_C_NUMERIC**
SQL_REAL	**SQL_C_FLOAT**
SQL_SMALLINT	**SQL_C_SSHORT**
SQL_TIMESTAMP	**SQL_C_TIMESTAMP**
SQL_VARCHAR	**SQL_C_CHARS**

(2 of 2)

IDS

Additional Default C Data Types for Dynamic Server

For each additional Informix ODBC Driver SQL data type for Dynamic Server, the following table shows the default C data type.

Informix ODBC Driver SQL Data Type (fSqlType)	Default Informix ODBC Driver C Data Type (fCType)
SQL_IFMX_UDT_BLOB	**SQL_C_BINARY**
SQL_IFMX_UDT_CLOB	**SQL_C_BINARY**
SQL_INFX_UDT_FIXED	This Informix ODBC Driver SQL data type does not have a default Informix ODBC Driver C data type. Because this Informix ODBC Driver SQL data type can contain binary data or character data, you must bind a variable for this Informix ODBC Driver SQL data type before you fetch a corresponding value. The data type of the bound variable specifies the C data type for the value.

(1 of 2)

Informix ODBC Driver SQL Data Type (fSqlType)	Default Informix ODBC Driver C Data Type (fCType)
SQL_INFX_UDT_VARYING	This Informix ODBC Driver SQL data type does not have a default Informix ODBC Driver C data type. Because this Informix ODBC Driver SQL data type can contain binary data or character data, you must bind a variable for this Informix ODBC Driver SQL data type before you fetch a corresponding value. The data type of the bound variable specifies the C data type for the value.

(2 of 2)

SQL to C: Binary

The binary Informix ODBC Driver SQL data type is SQL_LONGVARBINARY. The following table shows the Informix ODBC Driver C data types to which binary SQL data can be converted.

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Truncated data	Length of data	01004
SQL_C_CHAR	(Length of data) * 2 < *cbValueMax*	Data	Length of data	N/A
	(Length of data) * 2 ≥ *cbValueMax*	Truncated data	Length of data	01004

When IBM Informix ODBC Driver converts binary SQL data to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, IBM Informix ODBC Driver converts binary 00000001 to "01" and binary 11111111 to "FF."

IBM Informix ODBC Driver converts individual bytes to pairs of hexadecimal digits and terminates the character string with a null byte. Because of this conversion, if *cbValueMax* is even and is less than the length of the converted data, the last byte of the *rgbValue* buffer is not used. (The converted data requires an even number of bytes, the next-to-last byte is a null byte, and the last byte cannot be used.)

IDS

SQL to C: Boolean

The boolean Informix ODBC Driver SQL data type is SQL_BIT. The following table shows the Informix ODBC Driver C data types to which boolean SQL data can be converted. When IBM Informix ODBC Driver converts boolean SQL data to character C data, the possible values are `0` and `1`.

fCType	Test	rgbValue	pcbValue	SQLSTATE5
SQL_C_BINARY	*cbValueMax* ≤ 1	Data	`1`	N/A
	cbValueMax < 1	Untouched	Untouched	22003
SQL_C_BIT	IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.	Data	`1` (This is the size of the corresponding C data type.)	N/A
SQL_C_CHAR	*cbValueMax* > 1	Data	`1`	N/A
	cbValueMax ≤ 1	Untouched	Untouched	22003

SQL to C: Character

The character Informix ODBC Driver SQL data types are:

- SQL_CHAR
- SQL_LONGVARCHAR
- SQL_VARCHAR

The following table shows the Informix ODBC Driver C data types to which character SQL data can be converted. When IBM Informix ODBC Driver converts character SQL data to numeric, date, or time stamp C data, it ignores leading and trailing spaces.

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data ≤ *cbValueMax*.	Data	Length of data	N/A
	Length of data > *cbValueMax*.	Truncated data	Length of data	01004
SQL_C_BIT (IDS)	Data is 0 or 1.	Data	`1`	N/A
	Data is greater than 0, less than 2, and not equal to 1.	Truncated data	`1`	01004
	Data is less than 0 or greater than or equal to 2.	Untouched	Untouched	22003
	Data is not a *numeric-literal*.	Untouched	Untouched (The size of the corresponding C data type is 1.)	22005
SQL_C_CHAR	Length of data < *cbValueMax*.	Data	Length of data	N/A
	Length of data ≥ *cbValueMax*.	Truncated data	Length of data	01004
SQL_C_DATE	Data value is a valid *date-value*.	Data	`6`	N/A
	Data value is a valid *timestamp-value*; time portion is zero.	Data	`6`	N/A
	Data value is a valid *timestamp-value*; time portion is non-zero. (IBM Informix ODBC Driver ignores the date portion of *timestamp-value*.)	Truncated data	`6`	01004
	Data value is not a valid *date-value* or *timestamp-value*. (For all these conversions, IBM Informix ODBC Driver ignores the value of *cbValueMax*. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)	Untouched	Untouched (The size of the corresponding C data type is 6.)	22008

(1 of 3)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_DOUBLE **SQL_C_FLOAT**	Data is within the range of the data type to which the number is being converted.	Data	Size of the C data type	N/A
	Data is outside the range of the data type to which the number is being converted.	Untouched	Untouched	22003
	Data is not a *numeric-literal*.	Untouched	Untouched	22005
	(For all these conversions, IBM Informix ODBC Driver ignores the value of *cbValueMax*. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)			
SQL_C_LONG **SQL_C_SHORT** **SQL_C_SLONG** **SQL_C_SSHORT** **SQL_C_STINYINT** **SQL_C_TINYINT** **SQL_C_ULONG** **SQL_C_USHORT** **SQL_C_UTINYINT**	Data converted without truncation.	Data	Size of the C data type	N/A
	Data converted with truncation of fractional digits.	Truncated data	Size of the C data type	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	Untouched	Untouched	22003
	Data is not a *numeric-literal*.	Untouched	Untouched	22005
	(For all these conversions, IBM Informix ODBC Driver ignores the value of *cbValueMax*. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)			

(2 of 3)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_TIMESTAMP	Data value is a valid *timestamp-value*; fractional seconds portion not truncated.	Data	16	N/A
	Data value is a valid *timestamp-value*; fractional seconds portion truncated.	Truncated data	16	N/A
	Data value is a valid *date-value*.	Data (IBM Informix ODBC Driver sets the time fields of the time stamp structure to zero.)	16	N/A
	Data value is a valid *time-value*.	Data (IBM Informix ODBC Driver sets the date fields of the time stamp structure to the current date.)	16	N/A
	Data value is not a valid *date-value*, *time-value*, or *timestamp-value*.	Untouched	Untouched	22008
	(For all these conversions, IBM Informix ODBC Driver ignores the value of *cbValueMax*. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)		(The size of the corresponding C data type is 16.)	

(3 of 3)

SQL to C: Date

The date IBM Informix ODBC Driver SQL data type is SQL_DATE. The following table shows the IBM Informix ODBC Driver C data types to which date SQL data can be converted. When IBM Informix ODBC Driver converts date SQL data to character C data, the resulting string is in the "yyyy-mm-dd" format.

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Untouched	Untouched	22003
SQL_C_CHAR	*cbValueMax* ≥ 11	Data	`10`	N/A
	cbValueMax < 11	Untouched	Untouched	22003
SQL_C_DATE	IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.	Data	`6` (This is the size of the corresponding C data type.)	N/A
SQL_C_TIMESTAMP	IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.	Data (IBM Informix ODBC Driver sets the time fields of the time stamp structure to zero.)	`16` (This is the size of the corresponding C data type.)	N/A

SQL to C: Numeric

The numeric Informix ODBC Driver SQL data types are:

- SQL_DECIMAL
- SQL_DOUBLE
- SQL_INTEGER
- SQL_REAL
- SQL_SMALLINT

The following table shows the Informix ODBC Driver C data types to which numeric SQL data can be converted.

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data ≤ *cbValueMax*.	Data	Length of data	N/A
	Length of data > *cbValueMax*.	Untouched	Untouched	22003
SQL_C_BIT (IDS)	Data is 0 or 1.	Data	`1`	N/A
	Data is greater than 0, less than 2, and not equal to 1.	Truncated data	`1`	01004
	Data is less than 0 or greater than or equal to 2.	Untouched	Untouched	22003
	Data is not a *numeric-literal*.	Untouched	Untouched (The size of the corresponding C data type is 1.)	22005
SQL_C_CHAR	Display size < *cbValueMax*.	Data	Length of data	N/A
	Number of whole (as opposed to fractional) digits < *cbValueMax*.	Truncated data	Length of data	01004
	Number of whole (as opposed to fractional) digits ≥ *cbValueMax*.	Untouched	Untouched	22003

(1 of 2)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_DOUBLE **SQL_C_FLOAT**	Data is within the range of the data type to which the number is being converted.	Data	Size of the C data type	N/A
	Data is outside the range of the data type to which the number is being converted. (IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)	Untouched	Untouched	22003
SQL_C_LONG **SQL_C_SHORT** **SQL_C_SLONG** **SQL_C_SSHORT** **SQL_C_STINYINT** **SQL_C_TINYINT** **SQL_C_ULONG** **SQL_C_USHORT** **SQL_C_UTINYINT**	Data converted without truncation.	Data	Size of the C data type	N/A
	Data converted with truncation of fractional digits.	Truncated data	Size of the C data type	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits. (IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)	Untouched	Untouched	22003

(2 of 2)

SQL to C: Time Stamp

The time-stamp Informix ODBC Driver SQL data type is SQL_TIMESTAMP. The following table shows the Informix ODBC Driver C data types to which time-stamp SQL data can be converted.

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data ≤ *cbValueMax*.	Data	Length of data	N/A
	Length of data > *cbValueMax*.	Untouched	Untouched	22003
SQL_C_CHAR	*cbValueMax* > Display size.	Data	Length of data	N/A
	20 ≤ *cbValueMax* ≤ Display size.	Truncated data (IBM Informix ODBC Driver truncates the fractional seconds portion of the time stamp.)	Length of data	01004
	cbValueMax < 20.	Untouched	Untouched	22003

(1 of 2)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_DATE	Time portion of time stamp is zero.	Data	6	N/A
	Time portion of time stamp is nonzero.	Truncated data	6	01004
	(IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)	(IBM Informix ODBC Driver truncates the time portion of the time stamp.)	(The size of the corresponding C data type is 6.)	
SQL_C_TIMESTAMP	Fractional seconds portion of time stamp is not truncated.	Data	16	N/A
	Fractional seconds portion of time stamp is truncated.	Truncated data	16	01004
	(IBM Informix ODBC Driver ignores the value of *cbValueMax* for this conversion. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the C data type.)	(IBM Informix ODBC Driver truncates the fractional seconds portion of the time stamp.)	(The size of the corresponding C data type is 16.)	

(2 of 2)

When IBM Informix ODBC Driver converts time stamp SQL data to character C data, the resulting string is in the "yyyy-mm-dd hh:mm:ss[.f...]" format, where up to nine digits can be used for fractional seconds. Except for the decimal point and fractional seconds, the entire format must be used, regardless of the precision of the time stamp SQL data type.

SQL-to-C Data Conversion Examples

The following table illustrates how IBM Informix ODBC Driver converts SQL data to C data. "\0" represents a null-termination byte ("\0" represents a wide null termination character when the C data type is SQL_C_WCHAR). IBM Informix ODBC Driver always null-terminates **SQL_C_CHAR** and **SQL_C_WCHAR** data. For the combination of SQL_DATE and **SQL_C_TIMESTAMP**, IBM Informix ODBC Driver stores the numbers that are in the *rgbValue* column in the fields of the TIMESTAMP_STRUCT structure.

SQL Data Type	SQL Data Value	C Data Type	cbValueMax	rgbValue	SQLSTATE
SQL_CHAR	tigers	**SQL_C_CHAR**	7	tigers\0	N/A
SQL_CHAR	tigers	**SQL_C_CHAR**	6	tiger\0	01004
SQL_CHAR	tigers	**SQL_C_WCHAR**	14	tigers\0	N/A
SQL_CHAR	tigers	**SQL_C_WCHAR**	12	tiger\0	01004
SQL_DECIMAL	1234.56	**SQL_C_CHAR**	8	1234.56\0	N/A
SQL_DECIMAL	1234.56	**SQL_C_CHAR**	5	1234\0	01004
SQL_DECIMAL	1234.56	**SQL_C_CHAR**	4	—	22003
SQL_DECIMAL	1234.56	**SQL_C_WCHAR**	16	1234.56\0	N/A
SQL_DECIMAL	1234.56	**SQL_C_WCHAR**	10	1234\0	01004
SQL_DECIMAL	1234.56	**SQL_C_WCHAR**	8	—	220023
SQL_DECIMAL	1234.56	**SQL_C_FLOAT**	Ignored	1234.56	N/A
SQL_DECIMAL	1234.56	**SQL_C_SSHORT**	Ignored	1234	01004
SQL_DECIMAL	1234.56	**SQL_C_STINYINT**	Ignored	—	22003
SQL_DOUBLE	1.2345678	**SQL_C_DOUBLE**	Ignored	1.2345678	N/A
SQL_DOUBLE	1.2345678	**SQL_C_FLOAT**	Ignored	1.234567	N/A
SQL_DOUBLE	1.2345678	**SQL_C_STINYINT**	Ignored	1	N/A
SQL_DATE	1992-12-31	**SQL_C_CHAR**	11	1992-12-31\0	N/A
SQL_DATE	1992-12-31	**SQL_C_CHAR**	10	—	22003

(1 of 2)

SQL Data Type	SQL Data Value	C Data Type	cbValueMax	rgbValue	SQLSTATE
SQL_DATE	1992-12-31	**SQL_C_WCHAR**	22	1992-12-31\0	N/A
SQL_DATE	1992-12-31	**SQL_C_WCHAR**	20	—	22003
SQL_DATE	1992-12-31	**SQL_C_TIMESTAMP**	Ignored	1992,12,31, 0,0,0,0	N/A
SQL_TIMESTAMP	1992-12-31 23:45:55.12	**SQL_C_CHAR**	23	1992-12-31 23:45:55.12\0	N/A
SQL_TIMESTAMP	1992-12-31 23:45:55.12	**SQL_C_CHAR**	22	1992-12-31 23:45:55.1\0	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	**SQL_C_CHAR**	18	—	22003
SQL_TIMESTAMP	1992-12-31 23:45:55.12	**SQL_C_WCHAR**	46	1992-12-31 23:45:55.12\0	N/A
SQL_TIMESTAMP	1992-12-31 23:45:55.12	**SQL_C_WCHAR**	44	1992-12-31 23:45:55.1\0	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	**SQL_C_WCHAR**	36	—	22003

(2 of 2)

Important: *The size of a wide character (wchar_t) is platform dependent. The above examples are applicable to Windows where the size of wide characters is two bytes. On most UNIX platforms, wide characters are four bytes. On IBM AIX versions lower than AIX5L, it is 2 bytes.*

Converting Data from C to SQL

When you call **SQLExecute** or **SQLExecDirect**, IBM Informix ODBC Driver retrieves the data for parameters that are bound with **SQLBindParameter** from storage locations in the application. For data-at-execution parameters, call **SQLPutData** to send the parameter data. If necessary, IBM Informix ODBC Driver converts the data from the data type that the *ValueType* argument specifies in **SQLBindParameter** to the data type that the *fSqlType* argument specifies in **SQLBindParameter**. Finally, IBM Informix ODBC Driver sends the data to the data source.

The tables in the following sections describe how IBM Informix ODBC Driver converts data that it sends to the data source. For a given Informix ODBC Driver C data type, the first column of the table lists the legal input values of the *fSqlType* argument in **SQLBindParameter**. The second column lists the outcomes of a test that IBM Informix ODBC Driver performs to determine whether it can convert the data. The third column lists the SQLSTATE that is returned for each outcome by **SQLExecDirect**, **SQLExecute**, or **SQLPutData**. Data is sent to the data source only if SQL_SUCCESS is returned.

If the *fSqlType* argument in **SQLBindParameter** contains a value for an Informix ODBC Driver SQL data type that is not shown in the table for a given C data type, then **SQLBindParameter** returns SQLSTATE 07006 (Restricted data type attribute violation).

If the *rgbValue* and *pcbValue* arguments specified in **SQLBindParameter** are both null pointers, then that function returns SQLSTATE S1009 (Invalid argument value). To specify a null SQL data value, set the value that the *pcbValue* argument of **SQLBindParameter** points to or the value of the *cbValue* argument to SQL_NULL_DATA. To specify that the value in *rgbValue* is a null-terminated string, set these values to SQL_NTS.

The following terms are used in the tables:

- *Length of data* is the number of bytes of SQL data that are available to send to the data source, regardless of whether the data is truncated before it goes to the data source. For string data, this does not include the null-termination byte.
- *Column length* and *display size* are defined for each SQL data type in "Precision, Scale, Length, and Display Size" on page 3-9.
- *Number of digits* is the number of characters that represent a number, including the minus sign, decimal point, and exponent (if needed).
- Words in *italics* represent elements of the Informix ODBC Driver SQL syntax.

C to SQL: Binary

The binary Informix ODBC Driver C data type is **SQL_C_BINARY**. The following table shows the Informix ODBC Driver SQL data types to which binary C data can be converted. In the Test column, the SQL data length is the number of bytes needed to store the data on the data source. This length might be different from the column length, as defined in "Precision, Scale, Length, and Display Size" on page 3-9.

fSqlType	Test	SQLSTATE
SQL_BIGINT (IDS)	Length of data = SQL data length.	N/A
	Length of data ≠ SQL data length.	22003
SQL_BIT (IDS)	Length of data = SQL data length.	N/A
	Length of data ≠ SQL data length.	22003
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	Length of data ≤ Column length.	N/A
	Length of data > Column length.	01004
SQL_DATE SQL_TIMESTAMP	Length of data = SQL data length.	N/A
	Length of data ≠ SQL data length.	22003
SQL_DECIMAL SQL_DOUBLE SQL_INTEGER SQL_REAL SQL_SMALLINT	Length of data = SQL data length.	N/A
	Length of data ≠ SQL data length.	22003
SQL_LONGVARBINARY	Length of data ≤ Column length.	N/A
	Length of data > Column length.	01004

IDS

C to SQL: Bit

The bit Informix ODBC Driver C data type is **SQL_C_BIT**. The following table shows the Informix ODBC Driver SQL data types to which bit C data can be converted.

fSqlType	Test	SQLSTATE
SQL_BIGINT SQL_DECIMAL SQL_DOUBLE SQL_INTEGER SQL_REAL SQL_SMALLINT	None	N/A
SQL_BIT	None	N/A
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	None	N/A

IBM Informix ODBC Driver ignores the value that the *pcbValue* argument of **SQLBindParameter** points to and the value of the *cbValue* argument of **SQLPutData** when it converts data from the boolean C data type.
IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the boolean C data type.

C to SQL: Character

The character Informix ODBC Driver C data type is **SQL_C_CHAR**. The following table shows the Informix ODBC Driver SQL data types to which C character data can be converted.

fSqlType	Test	SQLSTATE
SQL_BIGINT (IDS)	Data converted without truncation.	N/A
	Data converted with truncation of fractional digits.	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	22003
	Data value is not a *numeric-literal*.	22005
SQL_BIT (IDS)	Data is 0 or 1.	N/A
	Data is greater than 0, less than 2, and not equal to 1.	01004
	Data is less than 0 or greater than or equal to 2.	22003
	Data is not a *numeric-literal*.	22005
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	Length of data ≤ Column length.	N/A
	Length of data > Column length.	01004
SQL_DATE	Data value is a valid Informix ODBC Driver *date-literal*.	N/A
	Data value is a valid Informix ODBC Driver *timestamp-literal*; time portion is zero.	N/A 01004
	Data value is a valid Informix ODBC Driver *timestamp-literal*; time portion is non-zero. IBM Informix ODBC Driver truncates the time portion of the time stamp.	22008
	Data value is not a valid Informix ODBC Driver *date-literal* or Informix ODBC Driver *timestamp-literal*.	

(1 of 2)

fSqlType	Test	SQLSTATE
SQL_DECIMAL SQL_INTEGER SQL_SMALLINT	Data converted without truncation.	N/A
	Data converted with truncation of fractional digits.	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	22003
	Data value is not a *numeric-literal*.	22005
SQL_DOUBLE SQL_REAL	Data is within the range of the data type to which the number is being converted.	N/A
	Data is outside the range of the data type to which the number is being converted.	22003
	Data value is not a *numeric-literal*.	22005
SQL_LONGVARBINARY	(Length of data) / 2 ≤ Column length.	N/A
	(Length of data) / 2 > Column length.	01004
	Data value is not a hexadecimal value.	22005
SQL_TIMESTAMP	Data value is a valid Informix ODBC Driver *timestamp-literal*; fractional seconds portion not truncated.	N/A
	Data value is a valid Informix ODBC Driver timestamp-literal; fractional seconds portion truncated.	01004
	Data value is a valid Informix ODBC Driver *date-literal*. IBM Informix ODBC Driver sets the time portion of the time stamp to zero.	N/A
	Data value is a valid Informix ODBC Driver *time-literal*. IBM Informix ODBC Driver sets the date portion of the time stamp to the current date.	N/A
	Data value is not a valid Informix ODBC Driver *date-literal*, Informix ODBC Driver *time-literal*, or Informix ODBC Driver *timestamp-literal*.	22008

(2 of 2)

When IBM Informix ODBC Driver converts character C data to numeric, date, or time stamp SQL data, it ignores leading and trailing blanks. When IBM Informix ODBC Driver converts character C data to binary SQL data, it converts each two bytes of character data to one byte of binary data. Each two bytes of character data represent a number in hexadecimal form. For example, IBM Informix ODBC Driver converts "01" to binary 00000001 and "FF" to binary 11111111.

IBM Informix ODBC Driver always converts pairs of hexadecimal digits to individual bytes and ignores the null-termination byte. Because of this conversion, if the length of the character string is odd, the last byte of the string (excluding the null termination byte, if any) is not converted.

C to SQL: Date

The date Informix ODBC Driver C data type is **SQL_C_DATE**. The following table shows the Informix ODBC Driver SQL data types to which date C data can be converted.

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	Column length ≥ 10.	N/A
	Column length < 10.	22003
	Data value is not a valid date.	22008
SQL_DATE	Data value is a valid date.	N/A
	Data value is not a valid date.	22008
SQL_TIMESTAMP	Data value is a valid date. IBM Informix ODBC Driver sets the time portion of the time stamp to zero.	N/A
	Data value is not a valid date.	22008

When IBM Informix ODBC Driver converts date C data to character SQL data, the resulting character data is in the "yyyy-mm-dd" format.

IBM Informix ODBC Driver ignores the value that the *pcbValue* argument of **SQLBindParameter** points to and the value of the *cbValue* argument of **SQLPutData** when it converts data from the date C data type. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the date C data type.

C to SQL: Numeric

The numeric Informix ODBC Driver C data types are:

- **SQL_C_DOUBLE**
- **SQL_C_FLOAT**
- **SQL_C_LONG**
- **SQL_C_SHORT**
- **SQL_C_SLONG**
- **SQL_C_STINYINT**
- **SQL_C_TINYINT**
- **SQL_C_ULONG**
- **SQL_C_USHORT**
- **SQL_C_UTINYINT**

The following table shows the Informix ODBC Driver SQL data types to which numeric C data can be converted.

fSqlType	Test	SQLSTATE
SQL_BIGINT (IDS)	Data converted without truncation.	N/A
	Data converted with truncation of fractional digits.	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	22003
SQL_BIT (IDS)	Data is 0 or 1.	N/A
	Data is greater than 0, less than 2, and not equal to 1.	01004
	Data is less than 0 or greater than or equal to 2.	22003

(1 of 2)

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	Number of digits ≤ Column length.	N/A
	Number of whole (as opposed to fractional) digits ≤ Column length.	01004
	Number of whole (as opposed to fractional) digits > Column length.	22003
SQL_DECIMAL SQL_INTEGER SQL_SMALLINT	Data converted without truncation.	N/A
	Data converted with truncation of fractional digits.	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	22003
SQL_DOUBLE SQL_REAL	Data is within the range of the data type to which the number is being converted.	N/A
	Data is outside the range of the data type to which the number is being converted.	22003

(2 of 2)

IBM Informix ODBC Driver ignores the value that the *pcbValue* argument of **SQLBindParameter** points to and the value of the *cbValue* argument of **SQLPutData** when it converts data from the numeric C data types. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the numeric C data type.

C to SQL: Time Stamp

The time-stamp Informix ODBC Driver C data type is **SQL_C_TIMESTAMP**. The following table shows the Informix ODBC Driver SQL data types to which time-stamp C data can be converted.

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	Column length ≥ Display size.	N/A
	19 ≤ Column length < Display size. IBM Informix ODBC Driver truncates the fractional seconds of the time stamp.	01004
	Column length < 19.	22003
	Data value is not a valid date.	22008
SQL_DATE	Time fields are zero.	N/A
	Time fields are non-zero. IBM Informix ODBC Driver truncates the time fields of the time stamp structure.	01004
	Data value does not contain a valid date.	22008
SQL_TIMESTAMP	Fractional seconds fields are not truncated.	N/A
	Fractional seconds fields are truncated. IBM Informix ODBC Driver truncates the fractional seconds fields of the time stamp structure.	01004
	Data value is not a valid time stamp.	22008

When IBM Informix ODBC Driver converts time stamp C data to character SQL data, the resulting character data is in the "yyyy-mm-dd hh:mm:ss[.f...]" format.

IBM Informix ODBC Driver ignores the value that the *pcbValue* argument of **SQLBindParameter** points to and the value of the *cbValue* argument of **SQLPutData** when it converts data from the time stamp C data type. IBM Informix ODBC Driver uses the size of *rgbValue* for the size of the time stamp C data type.

C-to-SQL Data Conversion Examples

The following table illustrates how IBM Informix ODBC Driver converts C data to SQL data. "\0" represents a null-termination byte. The null-termination byte is required only if the length of the data is SQL_NTS. For **SQL_C_DATE**, the numbers that are in the C Data Value column are the numbers that are stored in the fields of the DATE_STRUCT structure. For **SQL_C_TIMESTAMP**, the numbers that are in the C Data Value column are the numbers that are stored in the fields of the TIMESTAMP_STRUCT structure.

C Data Type	C Data Value	SQL Data Type	Column Length	SQL Data Value	SQLSTATE
SQL_C_CHAR	tigers\0	SQL_CHAR	6	tigers	N/A
SQL_C_CHAR	tigers\0	SQL_CHAR	5	tiger	01004
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	8 (In addition to bytes for numbers, one byte is required for a sign and another for the decimal point.)	1234.56	N/A
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	7 (In addition to bytes for numbers, one byte is required for a sign and another for the decimal point.)	1234.5	01004
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	4	—	22003
SQL_C_FLOAT	1234.56	SQL_FLOAT	not applicable	1234.56	N/A
SQL_C_FLOAT	1234.56	SQL_INTEGER	not applicable	1234	01004
SQL_C_FLOAT	1234.56	SQL_TINYINT	not applicable	—	22003
SQL_C_DATE	1992,12,31	SQL_CHAR	10	1992-12-31	N/A
SQL_C_DATE	1992,12,31	SQL_CHAR	9	—	22003

(1 of 2)

C Data Type	C Data Value	SQL Data Type	Column Length	SQL Data Value	SQLSTATE
SQL_C_DATE	1992,12,31	SQL_TIMESTAMP	not applicable	1992-12-31 00:00:00.0	N/A
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000	SQL_CHAR	22	1992-12-31 23:45:55.12	N/A
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000	SQL_CHAR	21	1992-12-31 23:45:55.1	01004
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000	SQL_CHAR	18	—	22003

(2 of 2)

Chapter

4

Working with Smart Large Objects

IDS

In This Chapter

The information in this chapter applies only if your database server is IBM Informix Dynamic Server.

This chapter describes how to store, create, and access a smart large object; how to transfer smart-large-object data; how to retrieve the status of a smart large object; and how to read or write a smart large object to or from a file.

A smart large object is a recoverable large object that is stored in an sbspace on disk. You can access a smart large object with read, write, and seek operations similar to an operating-system file. The two data types for smart large objects are *character large object* (CLOB) and *binary large object* (BLOB). A CLOB consists of text data and a BLOB consists of binary data in an undifferentiated byte stream.

For more information about smart-large-object data types, see Chapter 3, "Data Types," and the *IBM Informix Guide to SQL: Reference*. For information about the client functions that you use to access smart large objects, see Chapter 6, "Client Functions."

Working with Data Structures for Smart Large Objects

A smart large object can be huge. Therefore, instead of storing the content of a smart large object in a database table, Dynamic Server does the following:

- Stores the content of the smart large object in an sbspace
- Stores a pointer to the smart large object in the database table

Because a smart large object can be huge, an IBM Informix ODBC Driver application cannot receive a smart large object in a variable. Instead, the application sends or receives information about the smart large object in a data structure. The following table describes the data structures that IBM Informix ODBC Driver uses for smart large objects.

Data Structure	Name	Description
lofd	Smart-large-object file descriptor	Provides access to a smart large object. Uses a file descriptor to access smart-large-object data as if it were in an operating-system file.
loptr	Smart-large-object pointer structure	Provides security information and a pointer to a smart large object. This structure is the data that the database server stores in a database table for a smart large object. Therefore, SQL statements such as INSERT and SELECT accept a smart-large-object pointer structure as a value for a column or a parameter that has a data type of smart large object.
lospec	Smart-large-object specification structure	Specifies the storage characteristics for a smart large object.
lostat	Smart-large-object status structure	Stores status information for a smart large object. Normally you can fetch a user-defined data type (UDT) in either binary or character representation. However, it is not possible to convert a smart-large-object status structure to character representation. Therefore, you need to use **SQL_C_BINARY** as the Informix ODBC Driver C data type for **lostat**.

Important: *These data structures are opaque to IBM Informix ODBC Driver applications and their internal structures might change. Therefore, do not access the internal structures directly. Use the smart-large-object client functions to manipulate the data structures.*

The application is responsible for allocating space for these smart-large-object data structures.

To work with a smart-large-object data structure

1. Determine the size of the smart-large-object structure.
2. Use either a fixed size array or a dynamically allocated buffer that is at least the size of the data structure.
3. Free the array or buffer space when you are done using it.

The following code example illustrates these steps:

```
rc = SQLGetInfo(hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
   sizeof(lospec_size), NULL);
lospec_buffer = malloc(lospec_size);
.
.
.
free(lospec_buffer);
```

Handling the Storage of Smart Large Objects

The smart-large-object specification structure stores the following storage characteristics for a smart large object:

- Disk-storage information
- Create-time flags

Disk-Storage Information

Disk-storage information helps Dynamic Server determine how to store the smart large object most efficiently on disk. The following table describes the types of disk-storage information and the corresponding client functions. For most applications, it is recommended that you use the values for the disk-storage information that the database server determines.

Disk-Storage Information	Description	Client Functions
Estimated size	An estimate of the final size, in bytes, of the smart large object. The database server uses this value to determine the extents in which to store the smart large object. This value provides optimization information. If the value is grossly incorrect, it does not cause incorrect behavior. However, it does mean that the database server might not necessarily choose optimal extent sizes for the smart large object.	**ifx_lo_specget_estbytes()** **ifx_lo_specset_estbytes()**
Maximum size	The maximum size, in bytes, for the smart large object. The database server does not allow the smart large object to grow beyond this size.	**ifx_lo_specget_maxbytes()** **ifx_lo_specset_maxbytes()**

(1 of 2)

Disk-Storage Information	Description	Client Functions
Allocation extent size	The allocation extent size is specified in kilobytes. Optimally, the allocation extent is the single extent in a chunk that holds all the data for the smart large object. The database server performs storage allocations for smart large objects in increments of the allocation extent size. It tries to allocate an allocation extent as a single extent in a chunk. However, if no single extent is large enough, the database server must use multiple extents as necessary to satisfy the request.	**ifx_lo_specget_extsz()** **ifx_lo_specset_extsz()**
Name of the sbspace	The name of the sbspace that contains the smart large object. On this database server, an sbspace name can be up to 128 characters long and must be null terminated.	**ifx_lo_specget_sbspace()** **ifx_lo_specset_sbspace()**

(2 of 2)

Create-Time Flags

Create-time flags tell Dynamic Server what options to assign to the smart large object. The following table describes the create-time flags.

Type of Indicator	Create-Time Flag	Description
Logging	LO_LOG	Tells the database server to log changes to the smart large object in the system log file. Consider carefully whether to use the LO_LOG flag value. The database server incurs considerable overhead to log smart large objects. You must also make sure that the system log file is large enough to hold the value of the smart large object. For more information, see your *Administrator's Guide*.
	LO_NOLOG	Tells the database server to turn off logging for all operations that involve the associated smart large object.
Last access-time	LO_KEEP_LASTACCESS_TIME	Tells the database server to save the last access time for the smart large object. This access time is the time of the last read or write operation. Consider carefully whether to use the LO_KEEP_LASTACCESS_TIME flag value. The database server incurs considerable overhead to maintain last access times for smart large objects.
	LO_NOKEEP_LASTACCESS_TIME	Tells the database server not to maintain the last access time for the smart large object.

The **ifx_lo_specset_flags()** function sets the create-time flags to a new value. The **ifx_lo_specget_flags()** function retrieves the current value of the create-time flag.

Logging indicators and the last access-time indicators are stored in the smart-large-object specification structure as a single flag value. To set a flag from each group, use the C-language OR operator to mask the two flag values together. However, masking mutually exclusive flags causes an error. If you do not specify a value for one of the flag groups, the database server uses the inheritance hierarchy to determine this information.

Inheritance Hierarchy

Dynamic Server uses an inheritance hierarchy to obtain storage characteristics. Figure 4-1 shows the inheritance hierarchy for smart-large-object storage characteristics.

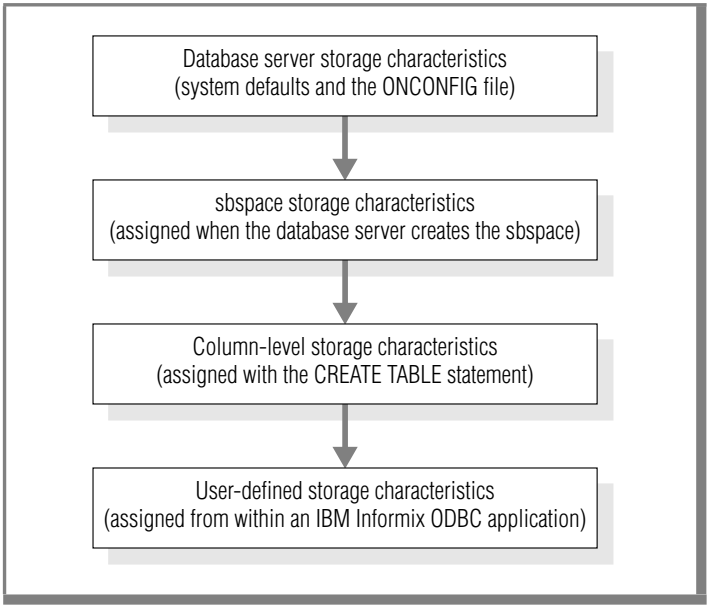


Figure 4-1 *Inheritance Hierarchy for Storage Characteristics*

System-Specified Storage Characteristics

Dynamic Server uses one of the following sets of storage characteristics as the system-specified storage characteristics:

- If the sbspace in which the smart large object is stored specifies a value for a particular storage characteristic, the database server uses the sbspace value as the system-specified storage characteristic.

  The database administrator can use the **onspaces** utility to define storage characteristics for an sbspace.

- If the sbspace in which the smart large object is stored does not specify a value for a particular storage characteristic, the database server uses the system default as the system-specified storage characteristic.

  The database server defines the system defaults for storage characteristics internally. However, you can specify a default sbspace name with the SBSPACENAME configuration parameter in the ONCONFIG file. Also, an application call to **ifx_lo_col_info()** or **ifx_lo_specset_sbspace()** can supply the target sbspace in the smart-large-object specification structure.

Warning: *An error will occur if the SBSPACENAME configuration parameter is not specified and the smart-large-object specification structure does not contain the name of the target sbspace.*

It is recommended that you use the system-specified storage characteristics for the disk-storage information. For more information about sbspaces and the description of the **onspaces** utility, see your *Administrator's Guide*.

To use system-specified storage characteristics for a new smart large object

1. Call **ifx_lo_def_create_spec()** to allocate a smart-large-object specification structure and to initialize the structure to null values.
2. Call **ifx_lo_create()** to create an instance of the smart large object.

Column-Level Storage Characteristics

The CREATE TABLE statement assigns storage characteristics to a database column. The PUT clause of the CREATE TABLE statement specifies storage characteristics for a smart-large-object column. The **syscolattribs** system catalog table stores the column-level storage characteristics.

To use column-level storage characteristics for a new smart-large-object instance

1. Call **ifx_lo_def_create_spec()** to allocate a smart-large-object specification structure and initialize this structure to null values.
2. Call **ifx_lo_col_info()** to retrieve the column-level storage characteristics and store them in the specified smart-large-object specification structure.
3. Call **ifx_lo_create()** to create an instance of the smart large object.

User-Defined Storage Characteristics

You can define a unique set of storage characteristics for a new smart large object, as follows:

- For a smart large object that will be stored in a column, you can override some storage characteristics for the column when you create an instance of a smart large object.

  If you do not override some or all of these characteristics, the smart large object uses the column-level storage characteristics.
- You can specify a wider set of characteristics for a smart large object because a smart large object is not constrained by table column properties.

  If you do not override some or all of these characteristics, the smart large object inherits the system-specified storage characteristics.

To specify user-defined storage characteristics, call an **ifx_lo_specset_*** function.

Creating a Smart Large Object

The following code example, **locreate.c**, shows how to create a smart large object. You can find the **locreate.c** file in the **%INFORMIXDIR%/demo/clidemo** directory on UNIX platforms and in the **%INFORMIXDIR%\demo\odbcdemo** directory in Windows environments. You can also find instructions on how to build the **odbc_demo** database in the same location.

```
/*
**      locreate.c
**
**  To create a smart large object
**
**    OBDC Functions:
**      SQLAllocHandle
**      SQLBindParameter
**      SQLConnect
**      SQLFreeStmt
**      SQLGetInfo
**      SQLDisconnect
**      SQLExecDirect
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN      12
#define ERRMSG_LEN      200

UCHAR   defDsn[] = "odbc_demo";


int checkError (SQLRETURN         rc,
                SQLSMALLINT     handleType,
                SQLHANDLE       handle,
                char            *errmsg)
{
   SQLRETURN         retcode = SQL_SUCCESS;

   SQLSMALLINT     errNum = 1;
   SQLCHAR         sqlState[6];
   SQLINTEGER      nativeError;
   SQLCHAR         errMsg[ERRMSG_LEN];
   SQLSMALLINT     textLengthPtr;
```

```
   if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
   {
      while (retcode != SQL_NO_DATA)
      {
         retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
            &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

         if (retcode == SQL_INVALID_HANDLE)
         {
            fprintf (stderr, "checkError function was called with an
               invalid handle!!\n");
            return 1;
         }

         if ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO))
            fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
               sqlState, errMsg);

         errNum++;
      }

      fprintf (stderr, "%s\n", errmsg);
      return 1;   /* all errors on this handle have been reported */
   }
   else
      return 0;   /* no errors to report */
}

int main (long         argc,
          char        *argv[])
{
   /* Declare variables
   */

   /* Handles */
   SQLHDBC     hdbc;
   SQLHENV     henv;
   SQLHSTMT    hstmt;

   /* Smart large object file descriptor */
   long        lofd;
   long        lofd_valsize = 0;

   /* Smart large object pointer structure */
   char*       loptr_buffer;
   short       loptr_size;
   long        loptr_valsize = 0;

   /* Smart large object specification structure */
   char*       lospec_buffer;
   short       lospec_size;
   long        lospec_valsize = 0;

   /* Write buffer */
   char*       write_buffer;
   short       write_size;
   long        write_valsize = 0;
```

```
   /* Miscellaneous variables */
   UCHAR       dsn[20];    /*name of the DSN used for connecting to the
                           database*/
   SQLRETURN   rc = 0;
   int         in;

   FILE*       hfile;
   char*       lo_file_name = "advert.txt";

   char        colname[BUFFER_LEN] = "item.advert";
   long        colname_size = SQL_NTS;

   long        mode = LO_RDWR;
   long        cbMode = 0;

   char*       insertStmt = "INSERT INTO item VALUES (1005, 'Helmet', 235,
                  'Each', ?, '39.95')";


/* STEP 1.  Get data source name from command line (or use default).
   **       Allocate environment handle and set ODBC version.
   **       Allocate connection handle.
   **       Establish the database connection.
   **       Allocate the statement handle.
   */

   /* If(dsn is not explicitly passed in as arg) */
   if (argc != 2)
   {
      /* Use default dsn - odbc_demo */
      fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
      strcpy ((char *)dsn, (char *)defDsn);
   }
   else
   {
      /* Use specified dsn */
      strcpy ((char *)dsn, (char *)argv[1]);
      fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
   }

   /* Allocate the Environment handle */
   rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   if (rc != SQL_SUCCESS)
   {
      fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
      return (1);
   }

   /* Set the ODBC version to 3.5 */
   rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
         (SQLPOINTER)SQL_OV_ODBC3, 0);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
         SQLSetEnvAttr failed\nExiting!!"))
      return (1);

   /* Allocate the connection handle */
   rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
         Handle Allocation failed\nExiting!!"))
      return (1);
```

```
   /* Establish the database connection */
   rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
         failed\n"))
      return (1);

   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
         Handle Allocation failed\nExiting!!"))
      return (1);

   fprintf (stdout, "STEP 1 done...connected to database\n");


/* STEP 2.  Get the size of the smart large object specification
   **       structure.
   **       Allocate a buffer to hold the structure.
   **       Create a default smart large object specification structure.
   **       Reset the statement parameters.
   */

   /* Get the size of a smart large object specification structure */
   rc = SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
         sizeof(lospec_size), NULL);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 2 -- SQLGetInfo
         failed\n"))
      goto Exit;

   /* Allocate a buffer to hold the smart large object specification
         structure*/
   lospec_buffer = malloc (lospec_size);

   /* Create a default smart large object specification structure */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
         lospec_size, &lospec_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter failed\n"))
      goto Exit;
   rc = SQLExecDirect (hstmt, "{call ifx_lo_def_create_spec(?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 2 done...default smart large object specification
         structure created\n");


/* STEP 3.  Initialise the smart large object specification structure
   **       with values for the database column where the smart large
   **       object is being inserted.
   **       Reset the statement parameters.
   */
```

```
   /* Initialise the smart large object specification structure */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         BUFFER_LEN, 0, colname, BUFFER_LEN, &colname_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   lospec_valsize = lospec_size;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
         lospec_size, &lospec_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_col_info(?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLFreeStm failed\n"))
      goto Exit;

   fprintf(stdout, "STEP 3 done...smart large object specification
         structure initialised\n");


/* STEP 4.  Get the size of the smart large object pointer structure.
   **       Allocate a buffer to hold the structure.
   */

   /* Get the size of the smart large object pointer structure */
   rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
         sizeof(loptr_size), NULL);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 4 --
         SQLGetInfo failed\n"))
      goto Exit;

   /* Allocate a buffer to hold the smart large object pointer structure */
   loptr_buffer = malloc (loptr_size);

   fprintf (stdout, "STEP 4 done...smart large object pointer structure
         allocated\n");


/* STEP 5.  Create a new smart large object.
   **       Reset the statement parameters.
   */

   /* Create a new smart large object */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
         lospec_size, &lospec_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;
```

```
   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
         SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   loptr_valsize = loptr_size;

   rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
         loptr_size, &loptr_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLBindParameter failed (param 3)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_SLONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLBindParameter failed (param 4)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_create(?, ?, ?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 5 done...smart large object created\n");


/* STEP 6.  Open the file containing data for the new smart large object.
   **       Allocate a buffer to hold the smart large object data.
   **       Read data from the input file into the smart large object.
   **       data buffer
   **       Write data from the data buffer into the new smart large.
   **       object.
   **       Reset the statement parameters.
   */

   /* Open the file containg data for the new smart large object */
   hfile = open (lo_file_name, "rt");
   /* sneaky way to get the size of the file */
   write_size = lseek (open (lo_file_name, "rt"), 0L, SEEK_END);

   /* Allocate a buffer to hold the smart large object data */
   write_buffer = malloc (write_size + 1);

   /* Read smart large object data from file */
   read (hfile, write_buffer, write_size);

   write_buffer[write_size] = '\0';
   write_valsize = write_size;
```

```
    /* Write data from the data buffer into the new smart large object */
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
          SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLBindParameter failed (param 1)\n"))
       goto Exit;

    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
          (UDWORD)write_size, 0, write_buffer, write_size, &write_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLBindParameter failed (param 2)\n"))
       goto Exit;

    rc = SQLExecDirect (hstmt, "{call ifx_lo_write(?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLExecDirect failed\n"))
       goto Exit;

    /* Reset the statement parameters */
    rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLFreeStmt failed\n"))
       goto Exit;

    fprintf (stdout, "STEP 6 done...data written to new smart large
          object\n");


/* STEP 7.  Insert the new smart large object into the database.
   **       Reset the statement parameters.
   */

    /* Insert the new smart large object into the database */
    loptr_valsize = loptr_size;

    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
          SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
          loptr_size, &loptr_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
          SQLBindParameter failed\n"))
       goto Exit;

    rc = SQLExecDirect (hstmt, insertStmt, SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
          SQLExecDirect failed\n"))
       goto Exit;

    /* Reset the statement parameters */
    rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
          SQLFreeStmt failed\n"))
       goto Exit;

    fprintf (stdout, "STEP 7 done...smart large object inserted into the
          database\n");
```

```
/* STEP 8.  Close the smart large object.
   */

   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLBindParameter failed\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 8 done...smart large object closed\n");


/* STEP 9.  Free the allocated buffers.
   */

   free (lospec_buffer);
   free (loptr_buffer);
   free (write_buffer);

   fprintf (stdout, "STEP 9 done...smart large object buffers freed\n");

   Exit:

   /* CLEANUP: Close the statement handle
   **          Free the statement handle
   **          Disconnect from the datasource
   **          Free the connection and environment handles
   **          Exit
   */

   /* Close the statement handle */
   SQLFreeStmt (hstmt, SQL_CLOSE);

   /* Free the statement handle */
   SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

   /* Disconnect from the data source */
   SQLDisconnect (hdbc);

   /* Free the environment handle and the database connection handle */
   SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
   SQLFreeHandle (SQL_HANDLE_ENV, henv);

   fprintf (stdout,"\n\nHit <Enter> to terminate the program...\n\n");
   in = getchar ();
   return (rc);
}
```

Transferring Smart-Large-Object Data

An INSERT or UPDATE statement does not perform the actual input of the smart-large-object data. It does, however, provide a means for the application to identify which smart-large-object data to associate with the column. A BLOB or CLOB column in a database table stores the smart-large-object pointer structure for a smart large object. Therefore, when you store a BLOB or CLOB column, you provide a smart-large-object pointer structure for the column in a **loptr** variable to the INSERT or UPDATE statement.

Figure 4-2 shows how an application transfers the data of a smart large object to the database server.

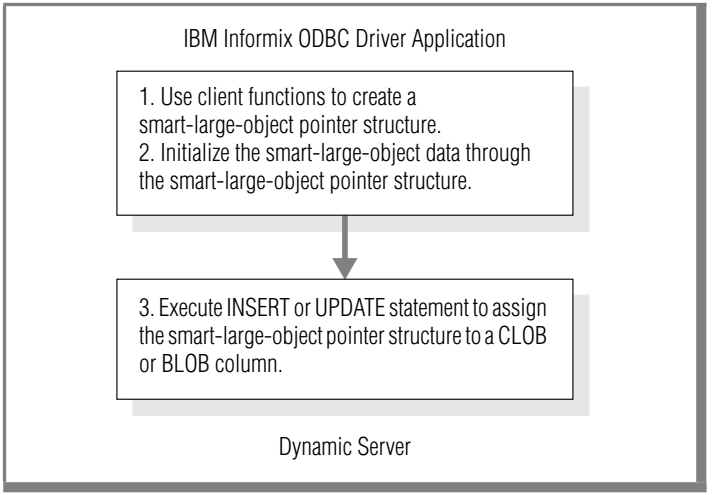


Figure 4-2 *Transferring Smart-Large-Object Data from Client Application to Database Server*

The smart large object that a smart-large-object pointer structure identifies exists as long as the smart-large-object pointer structure exists. When you store a smart-large-object pointer structure in a database, the database server deallocates the smart large object when appropriate.

If your application does not store the smart-large-object pointer structure for a new smart large object in the database, the smart-large-object pointer structure is only valid to access the version of the smart large object that was current when the pointer was passed to the application. If the smart large object is subsequently updated, the pointer is invalid. The smart-large-object pointer structures that you store in a row do not expire when the object version changes.

When you retrieve a row and then update a smart large object that is contained in that row, the database server exclusively locks the row for the time that it updates the smart large object. Moreover, long updates for smart large objects (whether or not logging is enabled and whether or not they are associated with a table row) create the potential for a long transaction condition if the smart large object takes a long time to update or create.

The smart-large-object pointer structure, not the CLOB or BLOB data itself, is stored in a CLOB or BLOB column in the database. Therefore, SQL statements such as INSERT and SELECT accept and return a smart-large-object pointer structure as the column value for a smart-large-object column.

Accessing a Smart Large Object

This section describes how to select, open, delete, modify, and close a smart large object using either the standard ODBC API or using **ifx_lo** functions.

Smart-Large-Object Automation

Instead of accessing smart large objects using the **ifx_lo** functions, you can access smart large objects using the standard ODBC API.

Operations supported when accessing smart large objects using the standard ODBC API include select, insert, update, and delete for CLOB and BLOB data types. You cannot access BYTE and TEXT smart large objects in this way.

Setting the Access Method Using INFX_LO_AUTOMATIC_ATTR

You can use the INFX_LO_AUTOMATIC_ATTR attribute to tell the database server whether you will access smart large objects using the ODBC API or using **ifx_lo** functions. If the application enables the INFX_LO_AUTOMATIC_ATTR attribute as a connection attribute, all statements for that connection inherit the attribute value. To change this attribute value per statement, you have to set and reset it as a statement attribute. If you enable this attribute for the statement, the application can access the smart large object using the standard ODBC way, as previously described. If you do not enable this attribute for the statement, the application accesses smart large objects using **ifx_lo** functions. The application cannot use the **ifx_lo** functions if this attribute is enabled for the statement.

You can also enable the INFX_LO_AUTOMATIC_ATTR attribute by turning on the **Report Standard ODBC Types** option under the **Advanced** tab of the ODBC Administration for Informix Driver DSN.

SQLDescribeCol for a CLOB data type column returns SQL_LONGVARCHAR for the DataPtrType. SQLDescribeCol for a BLOB data type column returns SQL_LONGVARBINARY, if the INFX_LO_AUTOMATIC_ATTR attribute is enabled for that statement.

SQLCOLATTRIBUTES for a CLOB data type column returns SQL_LONGVARCHAR for the Field Identifier of SQL_DESC_TYPE, whereas for the BLOB data type column it returns SQL_LONGVARBINARY only if the INFX_LO_AUTOMATIC_ATTR attribute is enabled for that statement.

Inserting, Updating, and Deleting Smart Large Objects Using the ODBC API

When you insert, update, or delete a CLOB data type, the application binds the CLOB data type using SQLBindParameter with C type as **SQL_C_CHAR** and SQL type as SQL_LONGVARCHAR.

When you insert, update, or delete a BLOB data type, the application binds BLOB data type using SQLBindParameter with C type as **SQL_C_BINARY** and SQL type as SQL_LONGVARBINARY.

IBM Informix ODBC Driver performs insertion of smart large objects in the following way:

- The driver sends a request to the database server to create a smart large object on the server side in the form of a new file.
- The driver gets back the file descriptor (for example, lofd) of this file from the database server.
- The driver sends the preceding lofd file and the smart-large-object data that was bound by the application using SQLBindParameter to the database server.
- The database server writes the data onto the file.

Selecting Smart Large Objects Using the ODBC API

When you select a CLOB data type, the application binds the column's C type as **SQL_C_CHAR**. When you select a BLOB data type, the C type is bound as **SQL_C_BINARY**.

IBM Informix ODBC Driver selects smart large objects in the following way:

- The driver sends a request to the database server to open the smart large object as a file on the server side.
- The driver gets back the file descriptor (for example, lofd) of this file from the database server.
- The driver sends the preceding lofd and a read request to the database server to read the smart-large-object data from the file.
- The database server reads the data from the corresponding file using the preceding lofd and sends it to the driver.
- The driver writes the data to the buffer that was bound by the application using SQLBindParameter.

Using ifx_lo Functions

This section describes how to select, open, delete, modify, and close a smart large object using **ifx_lo** functions. The section includes a code example at the end.

Selecting a Smart Large Object Using ifx_lo functions

A SELECT statement does not perform the actual output for the smart-large-object data. It does, however, establish a means for the application to identify a smart large object so that the application can then perform operations on the smart large object. Figure 4-3 on page 4-24 shows how the database server transfers the data of a smart large object to the application.

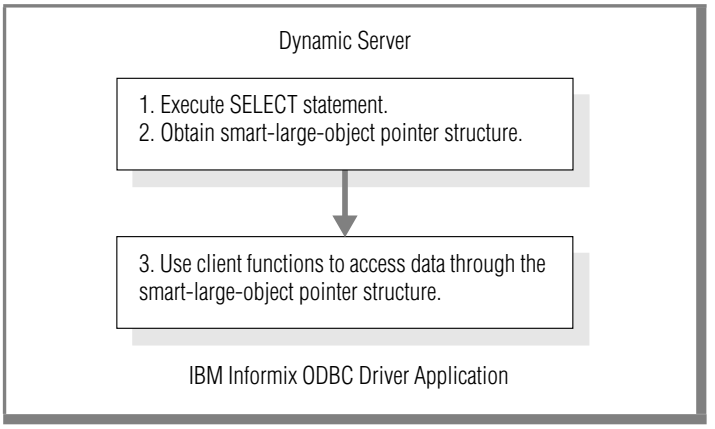


Figure 4-3 *Transferring Smart-Large-Object Data from Database Server to Client Application*

Opening a Smart Large Object Using ifx_lo functions

When you open a smart large object, you obtain a smart-large-object file descriptor for the smart large object. Through the smart-large-object file descriptor you can access the data of a smart large object as if it were in an operating-system file.

Access Modes

When you open a smart large object, you specify the access mode for the data. The access mode determines which read and write operations are valid on the open smart large object. The following table describes the access modes that **ifx_lo_open()** and **ifx_lo_create()** support.

Access Mode	Purpose	Constant
Read only	Only read operations are valid on the data.	LO_RDONLY
Dirty read	Lets you read uncommitted data pages for the smart large object. You cannot write to a smart large object after you set the mode to LO_DIRTY_READ. When you set this flag, you reset the current transaction isolation mode to dirty read for this smart large object. Do not base updates on data that you obtain from a smart large object in dirty-read mode.	LO_DIRTY_READ
Write only	Only write operations are valid on the data.	LO_WRONLY
Append	Intended for use in conjunction with LO_WRONLY or LO_RDWR. Sets the location pointer to the end of the object immediately prior to each write. Appends any data you write to the end of the smart large object. If LO_APPEND is used alone, the object is opened for reading only.	LO_APPEND

(1 of 2)

Access Mode	Purpose	Constant
Read/write	Both read and write operations are valid on the data.	LO_RDWR
Buffered access	Uses standard database server buffer pool.	LO_BUFFER
Lightweight I/O	Uses private buffers from the session pool of the database server.	LO_NOBUFFER

(2 of 2)

When you open a smart large object with LO_APPEND only, the database server opens the smart large object as read-only. Seek operations and read operations move the file pointer. Write operations fail and do not move the file pointer.

You can mask the LO_APPEND flag with another access mode. In any of these OR combinations, the seek operation remains unaffected. The following table shows the effect on the read and write operations that each of the OR combinations has.

OR Operation	Read Operations	Write Operations
LO_RDONLY | LO_APPEND	Occur at the file position and then move the file position to the end of the data that has been read.	Fail and do not move the file position.
LO_WRONLY | LO_APPEND	Fail and do not move the file position.	Move the file position to the end of the smart large object and then write the data; file position is at the end of the data after the write.
LO_RDWR | LO_APPEND	Occur at the file position and then move the file position to the end of the data that has been read.	Move the file position to the end of the smart large object and then write the data; file position is at the end of the data after the write.

Lightweight I/O

When the database server accesses smart large objects, it uses buffers from the buffer pool for buffered access. Unbuffered access is called *lightweight I/O*. Lightweight I/O uses private buffers instead of the buffer pool to hold smart large objects. These private buffers are allocated out of the database server session pool.

Lightweight I/O allows you to bypass the overhead of the least-recently-used (LRU) queues that the database server uses to manage the buffer pool. For more information about LRU queues, see your *Performance Guide*.

You can specify lightweight I/O by setting the flags parameter to LO_NOBUFFER when you create or open a smart large object. To specify buffered access, which is the default, use the LO_BUFFER flag.

Important: *Keep the following issues in mind when you use lightweight I/O:*

- *Close smart large objects with **ifx_lo_close()** when you finish with them to free memory allocated to the private buffers.*
- *All open operations that use lightweight I/O for a particular smart large object share the same private buffers. Consequently, one operation can cause the pages in the buffer to be flushed while other operations expect the object to be present in the buffer.*

The database server imposes the following restrictions on switching from lightweight I/O to buffered I/O:

- You can use the **ifx_lo_alter()** function to switch a smart large object from lightweight I/O (LO_NOBUFFER) to buffered I/O (LO_BUFFER) if the smart large object is *not* open. However, **ifx_lo_alter()** generates an error if you try to change a smart large object that uses buffered I/O to one that uses lightweight I/O.
- Unless you first use **ifx_lo_alter()** to change the access mode to buffered access (LO_BUFFER), you can only open a smart large object that was created with lightweight I/O with the LO_NOBUFFER access-mode flag. If an open operation specifies LO_BUFFER, the database server ignores the flag.

- You can open a smart large object that has been created with buffered access (LO_BUFFER) with the LO_NOBUFFER flag only if you open the object in read-only mode. If you attempt to write to the object, the database server returns an error. To write to the smart large object, you must close it and then reopen it with the LO_BUFFER flag and an access flag that allows write operations.

You can use the database server utility **onspaces** to specify lightweight I/O for all smart large objects in an sbspace. For more information on the **onspaces** utility, see your *Administrator's Guide*.

Smart-Large-Object Locks

To prevent simultaneous access to smart-large-object data, the database server locks a smart large object when you open it. Locks on smart large objects are different than row locks. If you retrieve a smart large object from a row, the database server might hold a row lock as well as a smart-large-object lock. The database server locks smart large objects because many columns can contain the same smart-large-object data.

To specify the lock mode of a smart large object, pass the access-mode flags, LO_RDONLY, LO_DIRTY_READ, LO_APPEND, LO_WRONLY, LO_RDWR, and LO_TRUNC, to the **ifx_lo_open()** and **ifx_lo_create()** functions. When you specify LO_RDONLY, the database server places a lock on the smart-large-object data. When you specify LO_DIRTY_READ, the database server does not place a lock on the smart-large-object data. If you specify any other access-mode flag, the database server obtains an update lock, which it promotes to an exclusive lock on first write or other update operation.

Share and update locks (read-only mode or write mode before an update operation occurs) are held until your application takes one of the following actions:

- Closes the smart large object
- Commits the transaction or rolls it back

Exclusive locks are held until the end of a transaction even if you close the smart large object.

Important: *You lose the lock at the end of a transaction even if the smart large object remains open. When the database server detects that a smart large object does not have an active lock, it places a new lock the next time that you access the smart large object. The lock that it places is based on the original open mode of the smart large object.*

Duration of an Open Operation on a Smart Large Object

After you open a smart large object with the **ifx_lo_create()** function or the **ifx_lo_open()** function, it remains open until one of the following events occurs:

- The **ifx_lo_close()** function closes the smart large object.
- The session ends.

Warning: *The end of the current transaction does not close a smart large object. It does, however, release any lock on a smart large object.*

Close smart large objects as soon as you finish using them. Leaving smart large objects open unnecessarily consumes system memory. Leaving many smart large objects open can eventually produce an out-of-memory condition.

Deleting a Smart Large Object

A smart large object is not deleted until *both* of the following conditions are met:

- The current transaction commits.
- The smart large object is closed, if the application opened the smart large object.

Modifying a Smart Large Object

To modify the data of a smart large object, perform the following steps:

1. Read and write the data in the open smart large object.
2. Use an UPDATE or INSERT statement to store the smart-large-object pointer in the database.

Closing a Smart Large Object

After you finish modifying a smart large object, call **ifx_lo_close()** to deallocate the resources that are assigned to it. When the resources are freed, you can reallocate them to other structures that your application needs. You can also reallocate the smart-large-object file descriptor to other smart large objects.

Example of Retrieving a Smart Large Object from the Database Using ifx_lo Functions

The following code example, **loselect.c**, shows how to retrieve a smart large object from the database. You can find the **loselect.c** file in the **%INFORMIXDIR%/demo/clidemo** directory on UNIX platforms and in the **%INFORMIXDIR%\demo\odbcdemo** directory on Windows platforms. You can also find instructions on how to build the **odbc_demo** database in the same location.

```
/*
**    loselect.c
**
** To access a smart large object
**    SQLBindCol
**    SQLBindParameter
**    SQLConnect
**    SQLFetch
**    SQLFreeStmt
**    SQLGetInfo
**    SQLDisconnect
**    SQLExecDirect
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define ERRMSG_LEN  200

UCHAR   defDsn[] = "odbc_demo";

int checkError (SQLRETURN     rc,
                SQLSMALLINT    handleType,
                SQLHANDLE      handle,
                char           *errmsg)
```

```
{
   SQLRETURN   retcode = SQL_SUCCESS;

   SQLSMALLINT errNum = 1;
   SQLCHAR     sqlState[6];
   SQLINTEGER  nativeError;
   SQLCHAR     errMsg[ERRMSG_LEN];
   SQLSMALLINT textLengthPtr;


   if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
   {
      while (retcode != SQL_NO_DATA)
      {
         retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
               &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

         if (retcode == SQL_INVALID_HANDLE)
         {
            fprintf (stderr, "checkError function was called with an
               invalid handle!!\n");
            return 1;
         }

         if ((retcode == SQL_SUCCESS) || (retcode ==
               SQL_SUCCESS_WITH_INFO))
            fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
                  sqlState, errMsg);

         errNum++;
      }

      fprintf (stderr, "%s\n", errmsg);
      return 1;   /* all errors on this handle have been reported */
   }
   else
      return 0;   /* no errors to report */
}

int main (long    argc,
          char    *argv[])
{
   /* Declare variables
   */

   /* Handles */
   SQLHDBC     hdbc;
   SQLHENV     henv;
   SQLHSTMT    hstmt;

   /* Smart large object file descriptor */
   long        lofd;
   long        lofd_valsize = 0;

   /* Smart large object pointer structure */
   char*       loptr_buffer;
   short       loptr_size;
   long        loptr_valsize = 0;
```

```
    /* Smart large object status structure */
    char*       lostat_buffer;
    short       lostat_size;
    long        lostat_valsize = 0;

    /* Smart large object data */
    char*       lo_data;
    long        lo_data_valsize = 0;

    /* Miscellaneous variables */
    UCHAR       dsn[20]; /*name of the DSN used for connecting to the
                   database*/
    SQLRETURN   rc = 0;
    int         in;

    char*       selectStmt = "SELECT advert FROM item WHERE item_num =
                   1004";
    long        mode = LO_RDONLY;
    long        lo_size;
    long        cbMode = 0, cbLoSize = 0;


/* STEP 1.  Get data source name from command line (or use default)
   **       Allocate the environment handle and set ODBC version
   **       Allocate the connection handle
   **       Establish the database connection
   **       Allocate the statement handle
   */

   /* If(dsn is not explicitly passed in as arg) */
   if (argc != 2)
   {
      /* Use default dsn - odbc_demo */
      fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
      strcpy ((char *)dsn, (char *)defDsn);
   }
   else
   {
      /* Use specified dsn */
      strcpy ((char *)dsn, (char *)argv[1]);
      fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
   }

   /* Allocate the Environment handle */
   rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   if (rc != SQL_SUCCESS)
   {
      fprintf (stdout, "Environment Handle Allocation
            failed\nExiting!!\n");
      return (1);
   }

   /* Set the ODBC version to 3.5 */
   rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
         (SQLPOINTER)SQL_OV_ODBC3, 0);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
         SQLSetEnvAttr failed\nExiting!!\n"))
      return (1);
```

```
   /* Allocate the connection handle */
   rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
         Handle Allocation failed\nExiting!!\n"))
      return (1);

   /* Establish the database connection */
   rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
         failed\nExiting!!"))
      return (1);
   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
         Handle Allocation failed\nExiting!!"))
      return (1);

   fprintf (stdout, "STEP 1 done...connected to database\n");


/* STEP 2.  Select a smart-large object from the database
   **       -- the select  statement executed is -
   **       "SELECT advert FROM item WHERE item_num = 1004"
   */

   /* Execute the select statement */
   rc = SQLExecDirect (hstmt, selectStmt, SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 2 done...select statement executed...smart large
         object retrieved from the databse\n");


/* STEP 3.  Get the size of the smart large object pointer structure.
   **       Allocate a buffer to hold the structure.
   **       Get the smart large object pointer structure from the
   **       database.
   **       Close the result set cursor.
   */

   /* Get the size of the smart large object pointer structure */
   rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
         sizeof(loptr_size),
         NULL);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 3 -- SQLGetInfo
         failed\n"))
      goto Exit;

   /* Allocate a buffer to hold the smart large object pointer structure */
   loptr_buffer = malloc (loptr_size);

   /* Bind the smart large object pointer structure buffer allocated to the
         column in the result set & fetch it from the database */
   rc = SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size,
         &loptr_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindCol failed\n"))
      goto Exit;
```

```
    rc = SQLFetch (hstmt);
    if (rc == SQL_NO_DATA_FOUND)
    {
       fprintf (stdout, "No Data Found\nExiting!!\n");
       goto Exit;
    }
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 -- SQLFetch
          failed\n"))
       goto Exit;

    /* Close the result set cursor */
    rc = SQLCloseCursor (hstmt);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
          SQLCloseCursor failed\n"))
       goto Exit;

    fprintf (stdout, "STEP 3 done...smart large object pointer structure
       fetched from the database\n");


  /* STEP 4.  Use the smart large object's pointer structure to open it
     **       and obtain the smart large object file descriptor.
     **       Reset the statement parameters.
     */

    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
          SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
          SQLBindParameter failed (param 1)\n"))
       goto Exit;

    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
          SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
          loptr_size, &loptr_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
          SQLBindParameter failed (param 2)\n"))
       goto Exit;

    rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
          SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
          SQLBindParameter failed (param 3)\n"))
       goto Exit;

    rc = SQLExecDirect (hstmt, "{? = call  ifx_lo_open(?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
          SQLExecDirect failed\n"))
       goto Exit;

    /* Reset the statement parameters */
    rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
          SQLFreeStmt failed\n"))
       goto Exit;

    fprintf (stdout, "STEP 4 done...smart large object opened... file
          descriptor obtained\n");
```

```
/* STEP 5.  Get the size of the smart large object status structure.
   **       Allocate a buffer to hold the structure.
   **       Get the smart large object status structure from the
   **       database.
   **       Reset the statement parameters.
   */

   /* Get the size of the smart large object status structure */
   rc = SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size,
         sizeof(lostat_size), NULL);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 5 -- SQLGetInfo
         failed\n"))
      goto Exit;

   /* Allocate a buffer to hold the smart large object status structure. */
   lostat_buffer = malloc(lostat_size);

   /* Get the smart large object status structure from the database. */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
         lostat_size, &lostat_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_stat(?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLExecDiret failed\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 5 done...smart large object status structure
         fetched from the database\n");


/* STEP 6.  Use the smart large object's status structure to get the
   **       size of the smart large object.
   **       Reset the statement parameters.
   */

   /* Use the smart large object status structure to get the size of the
         smart large object */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
         lostat_size, &lostat_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;
```

```
   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG,
         SQL_BIGINT, (UDWORD)0, 0, &lo_size, sizeof(lo_size), &cbLoSize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 6 done...smart large object size = %ld bytes\n",
         lo_size);


/* STEP 7.  Allocate a buffer to hold the smart large object's data.
   **       Read the smart large object's data using its file descriptor.
   **       Null-terminate the last byte of the smart large-object's data.
   **       Print out the contents of the smart large object.
   **       Reset the statement parameters.
   */

   /* Allocate a buffer to hold the smart large object's data chunks */
   lo_data = malloc (lo_size + 1);

   /* Read the smart large object's data */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR,
         lo_size, 0, lo_data, lo_size, &lo_data_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_read(?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Null-terminate the last byte of the smart large objects data */
   lo_data[lo_size] = '\0';

   /* Print the contents of the smart large object */
   fprintf (stdout, "Smart large object contents are.....\n\n\n%s\n\n\n",
         lo_data);

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
         SQLFreeStmt failed\n"))
      goto Exit;
```

```
   fprintf (stdout, "STEP 7 done...smart large object read completely\n");


/* STEP 8.  Close the smart large object.
   */

   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLBindParameter failed\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 8 done...smart large object closed\n");


/* STEP 9.  Free the allocated buffers.
   */

   free (loptr_buffer);
   free (lostat_buffer);
   free (lo_data);

   fprintf (stdout, "STEP 9 done...smart large object buffers freed\n");

   Exit:

   /* CLEANUP: Close the statement handle
   **          Free the statement handle
   **          Disconnect from the datasource
   **          Free the connection and environment handles
   **          Exit
   */

   /* Close the statement handle */
   SQLFreeStmt (hstmt, SQL_CLOSE);

   /* Free the statement handle */
   SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

   /* Disconnect from the data source */
   SQLDisconnect (hdbc);

   /* Free the environment handle and the database connection handle */
   SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
   SQLFreeHandle (SQL_HANDLE_ENV, henv);

   fprintf (stdout,"\n\nHit <Enter> to terminate the program...\n\n");
   in = getchar ();
   return (rc);
}
```

Retrieving the Status of a Smart Large Object

The following table describes the status information and the corresponding client functions.

Disk-Storage Information	Description	Client Functions
Last access time	The time, in seconds, that a smart large object was last accessed. This value is available only if the LO_KEEP_LASTACCESS_TIME flag is set for the smart large object.	**ifx_lo_stat_atime()**
Last change in status	The time, in seconds, of the last status change for a smart large object. A change in status includes updates, changes in ownership, and changes to the number of references.	**ifx_lo_stat_ctime()**
Last modification time (seconds)	The time, in seconds, that a smart large object was last modified.	**ifx_lo_stat_mtime_sec()**
Last modification time (microseconds)	The microsecond component of the time of last modification. This value is only supported on platforms that provide system time to microsecond granularity.	**ifx_lo_stat_mtime_usec()**
Reference count	A count of the number of references to a smart large object.	**ifx_lo_stat_refcnt()**
Size	The size, in bytes, of a smart large object.	**ifx_lo_stat_size()**

The time values (such as last access time and last change time) might differ slightly from the system time. This difference is due to the algorithm that the database server uses to obtain the time from the operating system.

Example of Retrieving Information About a Smart Large Object

The following code example, **loinfo.c**, shows how to retrieve information about a smart large object. You can find the **loinfo.c** file in the **%INFORMIXDIR%/demo/clidemo** directory on UNIX platforms and in the **%INFORMIXDIR%\demo\odbcdemo** directory in Windows environments. You can also find instructions on how to build the **odbc_demo** database in the same location.

```
/*
**    loinfo.c
**
** To check the status of a smart large object
**
**    OBDC Functions:
**       SQLBindCol
**       SQLBindParameter
**       SQLConnect
**       SQLFetch
**       SQLFreeStmt
**       SQLDisconnect
**       SQLExecDirect
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN  20
#define ERRMSG_LEN  200

UCHAR   defDsn[] = "odbc_demo";

int checkError (SQLRETURN     rc,
                SQLSMALLINT    handleType,
                SQLHANDLE      handle,
                char           *errmsg)
{
   SQLRETURN      retcode = SQL_SUCCESS;

   SQLSMALLINT    errNum = 1;
   SQLCHAR        sqlState[6];
   SQLINTEGER     nativeError;
   SQLCHAR        errMsg[ERRMSG_LEN];
   SQLSMALLINT    textLengthPtr;
```

```
   if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
   {
      while (retcode != SQL_NO_DATA)
      {
         retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
               &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

         if (retcode == SQL_INVALID_HANDLE)
         {
            fprintf (stderr, "checkError function was called with an
                  invalid handle!!\n");
            return 1;
         }

         if ((retcode == SQL_SUCCESS) || (retcode ==
               SQL_SUCCESS_WITH_INFO))
            fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
                  sqlState, errMsg);

         errNum++;
      }

      fprintf (stderr, "%s\n", errmsg);
      return 1;   /* all errors on this handle have been reported */
   }
   else
      return 0;   /* no errors to report */
}

int main (long    argc,
          char    *argv[])
{
   /* Declare variables
   */

   /* Handles */
   SQLHDBC     hdbc;
   SQLHENV     henv;
   SQLHSTMT    hstmt;

   /* Smart large object file descriptor */
   long        lofd;
   long        lofd_valsize = 0;

   /* Smart large object specification structure */
   char*       lospec_buffer;
   short       lospec_size;
   long        lospec_valsize = 0;

   /* Smart large object status structure */
   char*       lostat_buffer;
   short       lostat_size;
   long        lostat_valsize = 0;

   /* Smart large object pointer structure */
   char*       loptr_buffer;
   short       loptr_size;
   long        loptr_valsize = 0;
```

```
   /* Miscellaneous variables */
   UCHAR       dsn[20]; /*name of the DSN used for connecting to the
                  database*/
   SQLRETURN   rc = 0;
   int         in;

   char*       selectStmt = "SELECT advert FROM item WHERE item_num =
                  1004";
   long        lo_size;
   long        mode = LO_RDONLY;

   char        sbspace_name[BUFFER_LEN];
   long        sbspace_name_size = SQL_NTS;

   long        cbMode = 0, cbLoSize = 0;


/* STEP 1.  Get data source name from command line (or use default).
   **       Allocate the environment handle and set ODBC version.
   **       Allocate the connection handle.
   **       Establish the database connection.
   **       Allocate the statement handle.
   */

   /* If(dsn is not explicitly passed in as arg) */
   if (argc != 2)
   {
      /* Use default dsn - odbc_demo */
      fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
      strcpy ((char *)dsn, (char *)defDsn);
   }
   else
   {
      /* Use specified dsn */
      strcpy ((char *)dsn, (char *)argv[1]);
      fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
   }

   /* Allocate the Environment handle */
   rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   if (rc != SQL_SUCCESS)
   {
      fprintf (stdout, "Environment Handle Allocation
         failed\nExiting!!\n");
      return (1);
   }

   /* Set the ODBC version to 3.5 */
   rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
         (SQLPOINTER)SQL_OV_ODBC3, 0);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
      SQLSetEnvAttr failed\nExiting!!\n"))
      return (1);

   /* Allocate the connection handle */
   rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
      Handle Allocation failed\nExiting!!\n"))
      return (1);
```

```
   /* Establish the database connection */
   rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
         failed\nExiting!!"))
      return (1);

   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
         Handle Allocation failed\nExiting!!"))
      return (1);

   fprintf (stdout, "STEP 1 done...connected to database\n");


/* STEP 2.  Select a smart-large object from the database.
   **       -- the select  statement executed is -
   **       "SELECT advert FROM item WHERE item_num = 1004"
   */

   /* Execute the select statement */
   rc = SQLExecDirect (hstmt, selectStmt, SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 2 done...select statement executed...smart large
      object retrieved from the databse\n");


/* STEP 3.  Get the size of the smart large object pointer structure.
   **       Allocate a buffer to hold the structure.
   **       Get the smart large object pointer structure from the database.
   **       Close the result set cursor.
   */

   /* Get the size of the smart large object pointer structure */
   rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
         sizeof(loptr_size), NULL);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 3 -- SQLGetInfo
         failed\n"))
      goto Exit;

   /* Allocate a buffer to hold the smart large object pointer structure */
   loptr_buffer = malloc (loptr_size);

   /* Bind the smart large object pointer structure buffer allocated to the
         column in the result set & fetch it from the database */
   rc = SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size,
         &loptr_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindCol failed\n"))
      goto Exit;

   rc = SQLFetch (hstmt);
   if (rc == SQL_NO_DATA_FOUND)
   {
      fprintf (stdout, "No Data Found\nExiting!!\n");
      goto Exit;
   }
```

```
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 -- SQLFetch
         failed\n"))
      goto Exit;

   /* Close the result set cursor */
   rc = SQLCloseCursor (hstmt);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLCloseCursor failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 3 done...smart large object pointer structure
      fetched from the database\n");


/* STEP 4.  Use the smart large object's pointer structure to open it
   **       and obtain the smart large object file descriptor.
   **       Reset the statement parameters.
   */

   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
         loptr_size, &loptr_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLBindParameter failed (param 3)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{? = call  ifx_lo_open(?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 4 done...smart large object opened... file
      descriptor obtained\n");


/* STEP 5.  Get the size of the smart large object status structure.
   **       Allocate a buffer to hold the structure.
   **       Get the smart large object status structure from the database.
   **       Reset the statement parameters.
   */
```

```
    /* Get the size of the smart large object status structure */
    rc = SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size,
          sizeof(lostat_size), NULL);
    if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 5 -- SQLGetInfo
          failed\n"))
       goto Exit;

    /* Allocate a buffer to hold the smart large object status structure. */
    lostat_buffer = malloc(lostat_size);

    /* Get the smart large object status structure from the database. */
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
          SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
          SQLBindParameter failed (param 1)\n"))
       goto Exit;

    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
          SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
          lostat_size, &lostat_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
          SQLBindParameter failed (param 2)\n"))
       goto Exit;

    rc = SQLExecDirect (hstmt, "{call ifx_lo_stat(?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
          SQLExecDirect failed\n"))
       goto Exit;

    /* Reset the statement parameters */
    rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
          SQLFreeStmt failed\n"))
       goto Exit;

    fprintf (stdout, "STEP 5 done...smart large object status structure
       fetched from the database\n");


 /* STEP 6.  Use the smart large object's status structure to get the size
    **       of the smart large object.
    **       Reset the statement parameters.
    **       You can use additional ifx_lo_stat_*() functions to get more
    **       status information about the samrt large object.
    **       You can also use it to retrieve the smart large object
    **       specification structure and get further information about the
    **       smart large objectusing it's specification structure.
    */

    /* Use the smart large object status structure to get the size of the
          smart large object. */
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
          SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
          lostat_size, &lostat_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLBindParameter failed (param 1)\n"))
       goto Exit;
```

```
    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG,
          SQL_BIGINT, (UDWORD)0, 0, &lo_size, sizeof(lo_size), &cbLoSize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLBindParameter failed (param 1)\n"))
       goto Exit;

    rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLExecDirect failed\n"))
       goto Exit;

    /* Reset the statement parameters */
    rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
          SQLFreeStmt failed\n"))
       goto Exit;

    fprintf (stdout, "LARGE OBJECT SIZE = %ld\n", lo_size);
    fprintf (stdout, "STEP 6 done...smart large object size retrieved\n");


 /* STEP 7.  Get the size of the smart large object specification structure.
    **       Allocate a buffer to hold the structure.
    **       Get the smart large object specification structure from the
    **       database.
    **       Reset the statement parameters.
    */

    /* Get the size of the smart large object specification structure */
    rc = SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
          sizeof(lospec_size), NULL);
    if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 7 -- SQLGetInfo
          failed\n"))
       goto Exit;

    /* Allocate a buffer to hold the smart large object specification
          structure */
    lospec_buffer = malloc (lospec_size);

    /* Get the smart large object specification structure from the
          database */
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
          SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
          lostat_size, &lostat_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
          SQLBindParameter failed (param 1)\n"))
       goto Exit;

    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_BINARY,
          SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
          lospec_size, &lospec_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
          SQLBindParameter failed (param 2)\n"))
       goto Exit;

    rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_cspec(?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
          SQLExecDirect failed\n"))
       goto Exit;
```

```
   fprintf (stdout, "STEP 7 done...smart large object status structure
      fetched from the database\n");


/* STEP 8.  Use the smart large object's specification structure to get
   **       the sbspace name where the smart large object is stored.
   **       Reset the statement parameters.
   */

   /*  Use the smart large object's specification structure to get the
         sbspace name of the smart large object. */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
         lospec_size, &lospec_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR,
         BUFFER_LEN, 0, sbspace_name, BUFFER_LEN, &sbspace_name_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_specget_sbspace(?, ?)}",
         SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "LARGE OBJECT SBSPACE NAME = %s\n", sbspace_name);
   fprintf (stdout, "STEP 8 done...large object sbspace name retrieved\n");


/* STEP 9.  Close the smart large object.
   */

   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
         SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 9 --
         SQLBindParameter failed\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 9 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 9 done...smart large object closed\n");


/* STEP 10. Free the allocated buffers.
   */

   free (loptr_buffer);
   free (lostat_buffer);
   free (lospec_buffer);

   fprintf (stdout, "STEP 10 done...smart large object buffers freed\n");
```

```
    Exit:

    /* CLEANUP: Close the statement handle.
    **          Free the statement handle.
    **          Disconnect from the datasource.
    **          Free the connection and environment handles.
    **          Exit.
    */

    /* Close the statement handle */
    SQLFreeStmt (hstmt, SQL_CLOSE);

    /* Free the statement handle */
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

    /* Disconnect from the data source */
    SQLDisconnect (hdbc);

    /* Free the environment handle and the database connection handle */
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    fprintf (stdout,"\n\nHit <Enter> to terminate the program...\n\n");
    in = getchar ();
    return (rc);
}
```

Reading or Writing a Smart Large Object to or from a File

You can use the SQL functions **FILETOBLOB()** and **FILETOCLOB()** to transfer data from a file to a smart large object. The file can be on a client computer or on a server computer.

You can use the SQL function **LOTOFILE()** to transfer data from a smart large object to a file. The file might be on a client computer or on a server computer. **LOTOFILE()** accepts a smart-large-object pointer as a parameter. You can use the smart-large-object pointer structure for this parameter.

For more information about these SQL functions, see the *IBM Informix Guide to SQL: Syntax*.

Chapter

5

Working with Rows and Collections

IDS

In This Chapter

The information in this chapter applies only if your database server is IBM Informix Dynamic Server.

Rows and collections are composite values that consist of one or more elements. You can use the SELECT, UPDATE, INSERT, and DELETE statements to access an entire row or collection. However, these SQL statements do not let you access an element that is in a row or collection. To access an element, you need to retrieve the row or collection and then access the element from the local copy of the row or collection.

For more information about rows and collections, see Chapter 3, "Data Types," the *IBM Informix Guide to SQL: Reference*, and *IBM Informix User-Defined Routines and Data Types Developer's Guide*. For information about the client functions that you use to access rows and collections, see Chapter 6, "Client Functions."

Transferring Row and Collection Data

When you retrieve a row or collection, the database server puts the row or collection into a buffer that is local to your IBM Informix ODBC Driver application.

To allocate and bind a row or collection buffer

1. Call **ifx_rc_create()** to allocate the buffer.
2. Call **SQLBindCol()** to bind the buffer handle to the database column.
3. Execute a SELECT statement to transfer the row or collection data to the local buffer.
4. Use the row or collection buffer.
5. Call **ifx_rc_free()** to deallocate the buffer.

Fixed-Type Buffers and Unfixed-Type Buffers

The following table describes the differences between fixed-type buffers and unfixed-type buffers.

Kind of Buffer	Description
Fixed type	When you call **ifx_rc_create()** to create a row or collection buffer, you specify the following data types for the buffer: ■ The buffer data type (a row or one of the collection types) ■ The data types of the elements that are in the row or collection When you retrieve the row or collection, the database server compares the source and target data types and converts data from one Informix SQL data type to another as necessary. You can modify the row or collection buffer before you retrieve data into the buffer.
Unfixed type	When you call **ifx_rc_create()** to create a row or collection buffer, you specify only the buffer data type (a row or a collection) and *not* the element types. When you retrieve the row or collection, the database server does not compare data types because you did not specify the target data types. Instead, the row or collection buffer adopts the data types of the source data. You must initialize the row or collection buffer before you modify it. To initialize the buffer, retrieve a row or collection into it. The buffer type remains unfixed even when it contains data.

Buffers and Memory Allocation

When you retrieve data into a buffer that already contains a row or collection, IBM Informix ODBC Driver does not reuse the same buffer. Instead, IBM Informix ODBC Driver performs the following steps:

1. Creates a new row or collection buffer.
2. Associates the new buffer with the given buffer handle.
3. Deallocates the original buffer.

SQL Data

If the data types for a row or collection that are on a database server differ from the data types for a row or collection buffer into which the data is retrieved, the database server calls cast functions to convert the data from the source Informix SQL data types to the target Informix SQL data types. The following table lists the provider of the cast functions for each combination of source data type and target data type. Cast functions that a data type provides are located on the database server.

Source Data Type	Target Data Type	Provider of Cast Functions
Built-in	Built-in	Database server
Built-in	Extended	Data type
Extended	Built-in	Data type
Extended	Extended	Data type

Local Fetch

IBM Informix ODBC Driver performs a local fetch when you retrieve a row or collection from one location on the client computer to another location on the client computer.

To perform a local fetch

1. Call **ifx_rc_create()** to allocate a row or collection buffer.
2. Call **SQLBindCol()** to bind the buffer handle to the local row or collection.
3. Execute a SELECT statement to transfer the row or collection data to the local buffer.
4. For each element in the row or collection, call **ifx_rc_fetch()** to copy the value to the buffer.
5. Use the row or collection buffer.
6. Call **ifx_rc_free()** to deallocate the buffer.

A local fetch has the following limits on SQL data conversion:

- IBM Informix ODBC Driver cannot convert extended data types for which the cast functions are on a database server.
- IBM Informix ODBC Driver cannot convert data from one named row type to another. Only the database server can perform this kind of conversion.
- IBM Informix ODBC Driver cannot convert SQL data types when retrieving an entire row or collection. Thus, IBM Informix ODBC Driver can perform a local fetch of an entire row or collection only if the internal structures for the source and destination are the same or if the destination is an unfixed-type buffer.

  For example, if you define a local collection as **list (char(1) not null)**, the database server can put a **list (int not null)** value from the database server into the local collection. During this operation, the database server converts each integer into a string and constructs a new list to return to the client computer. You cannot perform this operation on the client computer where you retrieve a local list of integers into a list characters.

Example of Retrieving Row and Collection Data from the Database

The following sample program, **rcselect.c**, retrieves row and collection data from the database and displays it. This example also illustrates the fact that the same client functions can use row and collection handles interchangeably.

You can find the **rcselect.c** file in the **%INFORMIXDIR%/demo/clidemo** directory on UNIX and in the **%INFORMIXDIR%\demo\odbcdemo** directory in Windows. You can also find instructions on how to build the **odbc_demo** database in the same location.

```
/*
**      rcselect.c
**
** To access rows and collections
**    OBDC Functions:
**       SQLBindParameter
**       SQLConnect
**       SQLDisconnect
**       SQLExecDirect
**       SQLFetch
**       SQLFreeStmt
```

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN     25
#define ERRMSG_LEN     200

UCHAR   defDsn[] = "odbc_demo";

int checkError (SQLRETURN     rc,
                SQLSMALLINT    handleType,
                SQLHANDLE      handle,
                char           *errmsg)
{
   SQLRETURN      retcode = SQL_SUCCESS;

   SQLSMALLINT    errNum = 1;
   SQLCHAR        sqlState[6];
   SQLINTEGER     nativeError;
   SQLCHAR        errMsg[ERRMSG_LEN];
   SQLSMALLINT    textLengthPtr;

   if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
   {
      while (retcode != SQL_NO_DATA)
      {
         retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
               &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

         if (retcode == SQL_INVALID_HANDLE)
         {
            fprintf (stderr, "checkError function was called with an
                  invalid handle!!\n");
            return 1;
         }

         if ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO))
            fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
                  sqlState, errMsg);

         errNum++;
      }

      fprintf (stderr, "%s\n", errmsg);
      return 1;   /* all errors on this handle have been reported */
   }
   else
      return 0;   /* no errors to report */
}
```

```
/*
**  Executes the given select statement and assumes the results will be
**  either rows or collections. The 'hrc' parameter may reference either
**  a row or a collection. Rows and collection handles may often be used
**  interchangeably.
**
**  Each row of the select statement will be fetched into the given row or
**  collection handle.  Then each field of the row or collection will be
**  individually converted into a character buffer and displayed.
**
**  This function returns 0 if an error occurs, else returns 1
**
*/

int do_select  (SQLHDBC hdbc,
                char*    select_str,
                HINFX_RC hrc)
{
   SQLHSTMT    hRCStmt;
   SQLHSTMT    hSelectStmt;
   SQLRETURN   rc = 0;

   short       index, rownum;
   short       position = SQL_INFX_RC_ABSOLUTE;
   short       jump;

   char        fname[BUFFER_LEN];
   char        lname[BUFFER_LEN];
   char        rc_data[BUFFER_LEN];

   SQLINTEGER  cbFname = 0, cbLname = 0, cbHrc = 0;
   SQLINTEGER  cbPosition = 0, cbJump = 0, cbRCData = 0;

/* STEP A.  Allocate the statement handles for the select statement and
   **       the statement used to retrieve the row/collection data.
   */

   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hRCStmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step A -- Statement
         Handle Allocation failed for row/collection
         statement\nExiting!!"))
      return 0;

   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hSelectStmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step A -- Statement
         Handle Allocation failed for select statement\nExiting!!"))
      return 0;

   fprintf (stdout, "STEP A done...statement handles allocated\n");


/* STEP B.  Execute the select statement.
   **       Bind the result set columns -
   **       --  col1 = fname
   **       col2 = lname
   **       col3 = row/collection data
   */
```

```
   /* Execute the select statement */
   rc = SQLExecDirect (hSelectStmt, select_str, SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
         SQLExecDirect failed\n"))
      return 0;

   /* Bind the result set columns */
   rc = SQLBindCol (hSelectStmt, 1, SQL_C_CHAR, (SQLPOINTER)fname,
         BUFFER_LEN, &cbFname);
   if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
         SQLBindCol failed for column 'fname'\n"))
      return 0;

   rc = SQLBindCol (hSelectStmt, 2, SQL_C_CHAR, (SQLPOINTER)lname,
         BUFFER_LEN, &cbLname);
   if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
         SQLBindCol failed for column 'lname'\n"))
      return 0;

   rc = SQLBindCol (hSelectStmt, 3, SQL_C_BINARY, (SQLPOINTER)hrc,
         sizeof(HINFX_RC), &cbHrc);
   if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
         SQLBindCol failed for row/collection column\n"))
      return 0;

   fprintf (stdout, "STEP B done...select statement executed and result set
      columns bound\n");


/* STEP C. Retrieve the results.
   */

   for (rownum = 1;; rownum++)
   {
      rc = SQLFetch (hSelectStmt);
      if (rc == SQL_NO_DATA_FOUND)
      {
         fprintf (stdout, "No data found...\n");
         break;
      }
      else if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in
            Step C -- SQLFetch failed\n"))
         return 0;


      fprintf(stdout, "Retrieving row number %d:\n\tfname -- %s\n\tlname --
         %s\n\tRow/Collection Data --\n", rownum, fname, lname);

      /*  For each row in the result set, display each field of the
            retrieved row/collection */
      for (index = 1;; index++)
      {
         strcpy(rc_data, "<null>");

         /* Each value in the local row/collection will be fetched into a
          * character buffer and displayed using fprintf().
          */
```

```
            rc = SQLBindParameter (hRCStmt, 1, SQL_PARAM_OUTPUT, SQL_C_CHAR,
                  SQL_CHAR, 0, 0, rc_data, BUFFER_LEN, &cbRCData);
            if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
                  SQLBindParameter failed (param 1)\n"))
               return 0;

            rc = SQLBindParameter (hRCStmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
                  SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrc,
                  sizeof(HINFX_RC), &cbHrc);
            if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
                  SQLBindParameter failed (param 2)\n"))
               return 0;

            rc = SQLBindParameter (hRCStmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
                  SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
            if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
                  SQLBindParameter failed (param 3)\n"))
               return 0;

            jump = index;
            rc = SQLBindParameter (hRCStmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
                  SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
            if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
                  SQLBindParameter failed (param 4)\n"))
               return 0;

            rc = SQLExecDirect(hRCStmt, "{ ? = call ifx_rc_fetch( ?, ?, ? ) }",
                  SQL_NTS);
            if (rc == SQL_NO_DATA_FOUND)
            {
               break;
            }
            else if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in
                  Step C -- SQLExecDirect failed\n"))
               return 0;

            /* Display retrieved row */
            fprintf(stdout, "\t\t%d: %s\n", index, rc_data);
         }
      }

      fprintf (stdout, "STEP C done...results retrieved\n");

      /* Free the statement handles */
      SQLFreeHandle (SQL_HANDLE_STMT, hSelectStmt);
      SQLFreeHandle (SQL_HANDLE_STMT, hRCStmt);

      return 1; /* no error */
   }

   /*
    * This function allocates the row and collection buffers, passes
    * them to the do_select() function, along with an appropriate select
    * statement and then frees all allocated handles.
    */
   int main (long argc,
            char *argv[])
   {
      /* Declare variables
      */
```

```
   /* Handles */
   SQLHDBC     hdbc;
   SQLHENV     henv;
   SQLHSTMT    hstmt;
   HINFX_RC    hrow, hlist;

   /* Miscellaneous variables */

   UCHAR       dsn[20]; /*name of the DSN used for connecting to the
                        database*/
   SQLRETURN   rc = 0;
   int         in;

   int         data_size = SQL_NTS;
   char*       listSelectStmt = "SELECT fname, lname, contact_dates FROM
               customer";
   char*       rowSelectStmt = "SELECT fname, lname, address FROM
               customer";

   SQLINTEGER  cbHlist = 0, cbHrow = 0;


/* STEP 1.  Get data source name from command line (or use default).
   **       Allocate environment handle and set ODBC version.
   **       Allocate connection handle.
   **       Establish the database connection.
   **       Allocate the statement handle.
   */

   /* If(dsn is not explicitly passed in as arg) */
   if (argc != 2)
   {
      /* Use default dsn - odbc_demo */
      fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
      strcpy ((char *)dsn, (char *)defDsn);
   }
   else
   {
      /* Use specified dsn */
      strcpy ((char *)dsn, (char *)argv[1]);
      fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
   }

   /* Allocate the Environment handle */
   rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   if (rc != SQL_SUCCESS)
   {
      fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
      return (1);
   }

   /* Set the ODBC version to 3.5 */
   rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
          (SQLPOINTER)SQL_OV_ODBC3, 0);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
         SQLSetEnvAttr failed\nExiting!!"))
      return (1);
```

```
   /* Allocate the connection handle */
   rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
         Handle Allocation failed\nExiting!!"))
      return (1);

   /* Establish the database connection */
   rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
         failed\n"))
      return (1);

   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
         Handle Allocation failed\nExiting!!"))
      return (1);

   fprintf (stdout, "STEP 1 done...connected to database\n");


/* STEP 2.  Allocate an unfixed collection handle.
   **       Retrieve database rows containing a list.
   **       Reset the statement parameters.
   */

   /* Allocate an unfixed list handle */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
         SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC),
         &cbHlist);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter (param 1) failed\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         0, 0, (UCHAR *) "list", 0, &data_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter (param 2) failed\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{? = call ifx_rc_create(?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Retrieve databse rows containing a list */
    if (!do_select (hdbc, listSelectStmt, hlist))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 2 done...list data retrieved\n");
   fprintf (stdout,"\nHit <Enter> to continue...");
   in = getchar ();
```

```
/* STEP 3.  Allocate an unfixed row handle.
   **       Retrieve database rows containing a row.
   **       Reset the statement parameters.
   */

   /* Allocate an unfixed row handle */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
         SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC),
         &cbHrow);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 1) failed\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         0, 0, (UCHAR *) "row", 0, &data_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 2) failed\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, "{? = call ifx_rc_create(?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLExecDirect failed\n"))
      goto Exit;

   /* Retrieve databse rows containing a row */
   if (!do_select (hdbc, rowSelectStmt, hrow))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 3 done...row data retrieved\n");


/* STEP 4.  Free the row and list handles.
   */

   /* Free the row handle */
   rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
         &cbHrow);

   rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

   /* Free the list handle */
   rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
         &cbHlist);

   rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

   fprintf (stdout, "STEP 4 done...row and list handles freed\n");

   Exit:
```

```
/* CLEANUP: Close the statement handle.
   **       Free the statement handle.
   **       Disconnect from the datasource.
   **       Free the connection and environment handles.
   **       Exit.
   */

   /* Close the statement handle */
   SQLFreeStmt (hstmt, SQL_CLOSE);

   /* Free the statement handle */
   SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

   /* Disconnect from the data source */
   SQLDisconnect (hdbc);

   /* Free the environment handle and the database connection handle */
   SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
   SQLFreeHandle (SQL_HANDLE_ENV, henv);

   fprintf (stdout,"\n\nHit <Enter> to terminate the program...\n\n");
   in = getchar ();
   return (rc);
```

Creating a Row or Collection

The following code example, **rccreate.c**, creates a row and a list on the client, adds items to them, and inserts them into the database. You can find the **rccreate.c** file in the **%INFORMIXDIR%/demo/clidemo** directory on UNIX and in the **%INFORMIXDIR%\demo\odbcdemo** directory in Windows. You can also find instructions on how to build the **odbc_demo** database in the same location.

Example of Creating a Row and a List on the Client

```
/*
**    rccreate.c
**
** To create a collection & insert it into the database table
**
**
**    OBDC Functions:
**       SQLBindParameter
**       SQLConnect
**       SQLDisconnect
**       SQLExecDirect
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN   25
#define ERRMSG_LEN   200

UCHAR    defDsn[] = "odbc_demo";

int checkError (SQLRETURN     rc,
                SQLSMALLINT    handleType,
                SQLHANDLE      handle,
                char           *errmsg)
{
   SQLRETURN      retcode = SQL_SUCCESS;

   SQLSMALLINT    errNum = 1;
   SQLCHAR        sqlState[6];
   SQLINTEGER     nativeError;
   SQLCHAR        errMsg[ERRMSG_LEN];
   SQLSMALLINT    textLengthPtr;
   if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
   {
      while (retcode != SQL_NO_DATA)
      {
         retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
               &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

         if (retcode == SQL_INVALID_HANDLE)
         {
            fprintf (stderr, "checkError function was called with an
                  invalid handle!!\n");
            return 1;
         }

         if ((retcode == SQL_SUCCESS) || (retcode ==
                  SQL_SUCCESS_WITH_INFO)) fprintf (stderr, "ERROR: %d:  %s
                  : %s \n", nativeError, sqlState, errMsg);

         errNum++;
      }

      fprintf (stderr, "%s\n", errmsg);
      return 1;   /* all errors on this handle have been reported */
   }
   else
      return 0;   /* no errors to report */
}

int main (long    argc,
          char     *argv[] )
{
```

```
   /* Declare variables
   */

   /* Handles */
   SQLHDB      hdbc;
   SQLHENV     henv;
   SQLHSTMT    hstmt;

   HINFX_RC    hrow;
   HINFX_RC    hlist;

   /* Miscellaneous variables */
   UCHAR       dsn[20];    /*name of the DSN used for connecting to the
                           database*/
   SQLRETURN   rc = 0;
   int         i, in;
   int         data_size = SQL_NTS;
   short       position = SQL_INFX_RC_ABSOLUTE;
   short       jump;

   UCHAR       row_data[4][BUFFER_LEN] = {"520 Topaz Way", "Redwood City",
                     "CA", "94062"};
   int         row_data_size = SQL_NTS;

   UCHAR        list_data[2][BUFFER_LEN] = {"1991-06-20", "1993-07-17"};
   int         list_data_size = SQL_NTS;

   char*       insertStmt = "INSERT INTO customer VALUES (110, 'Roy',
               'Jaeger', ?, ?)";
   SQLINTEGER  cbHrow = 0, cbHlist = 0, cbPosition = 0, cbJump = 0;
/* STEP 1.  Get data source name from command line (or use default).
   **       Allocate environment handle and set ODBC version.
   **       Allocate connection handle.
   **       Establish the database connection.
   **       Allocate the statement handle.
   */

   /* If(dsn is not explicitly passed in as arg) */
   if (argc != 2)
   {
      /* Use default dsn - odbc_demo */
      fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
      strcpy ((char *)dsn, (char *)defDsn);
   }
   else
   {
      /* Use specified dsn */
      strcpy ((char *)dsn, (char *)argv[1]);
      fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
   }
   /* Allocate the Environment handle */
    rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    if (rc != SQL_SUCCESS)
   {
      fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
      return (1);
   }
```

```
   /* Set the ODBC version to 3.5 */
   rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
         (SQLPOINTER)SQL_OV_ODBC3, 0);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
         SQLSetEnvAttr failed\nExiting!!"))
      return (1);

   /* Allocate the connection handle */
   rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
   if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
         Handle Allocation failed\nExiting!!"))
      return (1);

   /* Establish the database connection */
   rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
         failed\n"))
      return (1);

   /* Allocate the statement handle */
   rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
   if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
         Handle Allocation failed\nExiting!!"))
      return (1);

   fprintf (stdout, "STEP 1 done...connected to database\n");


/* STEP 2.  Allocate fixed-type row handle -- this creates a non-null row
   **       buffer, each of whose values is null, and can be updated.
   **       Allocate a fixed-type list handle -- this creates a non-null
   **       but empty list buffer into which values can be inserted.
   **       Reset the statement parameters.
   */

   /* Allocate a fixed-type row handle -- this creates a row with each
      value empty */

   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
         SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC),
         &cbHrow);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter (param 1) failed for row handle\n"))
       goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         0, 0, (UCHAR *) "ROW(address1 VARCHAR(25), city VARCHAR(15), state
         VARCHAR(15), zip  VARCHAR(5))", 0, &data_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter (param 2) failed for row handle\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, (UCHAR *) "{? = call ifx_rc_create(?)}",
         SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLExecDirect failed for row handle\n"))
      goto Exit;
```

```
   /* Allocate a fixed-type list handle */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
         SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC),
         &cbHlist);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter (param 1) failed for list handle\n"))
      goto Exit;

   data_size = SQL_NTS;
   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         0, 0, (UCHAR *) "LIST (DATETIME YEAR TO DAY NOT NULL)",0,
         &data_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLBindParameter (param 2) failed for list handle\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, (UCHAR *) "{? = call ifx_rc_create(?)}",
      SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLExecDirect failed for list handle\n"))
      goto Exit;

   /* Reset the statement parameters */
   rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
         SQLFreeStmt failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 2 done...fixed-type row and collection handles
      allocated\n");

/* STEP 3.  Update the elements of the fixed-type row buffer allocated.
   **       Insert elements into the fixed-type list buffer allocated.
   **       Reset the statement parameters.
   */

   /* Update elements of the row buffer */
   for (i=0; i<4; i++)
   {
      rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
            SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
            &cbHrow);
      if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
            SQLBindParameter (param 1) failed for row handle\n"))
         goto Exit;

      rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
            SQL_CHAR, BUFFER_LEN, 0, row_data[i], 0, &row_data_size);
      if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
            SQLBindParameter (param 2) failed for row handle\n"))
         goto Exit;

      rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
            SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
      if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
            SQLBindParameter (param 3) failed for row handle\n"))
         goto Exit;
```

```
   jump = i + 1;
   rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
         SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 4) failed for row handle\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt,
         (UCHAR *)"{call ifx_rc_update(?, ?, ?, ?)}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLExecDirect failed for row handle\n"))
      goto Exit;
}

/* Insert elements into the list buffer */
for (i=0; i<2; i++)
{
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
         &cbHlist);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 1) failed for list handle\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
         SQL_DATE, 25, 0, list_data[i], 0, &list_data_size);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 2) failed for list handle\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
         SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 3) failed for list handle\n"))
      goto Exit;

   jump = i + 1;
   rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
         SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLBindParameter (param 4) failed for list handle\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt,
         (UCHAR *)"{call ifx_rc_insert( ?, ?, ?, ? )}", SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
         SQLExecDirect failed for list handle\n"))
      goto Exit;
}

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
      SQLFreeStmt failed\n"))
   goto Exit;

fprintf (stdout, "STEP 3 done...row and list buffers populated\n");
```

```
/* STEP 4.  Bind paramters for the row and list handles.
   **       Execute the insert statement to insert the new row into table
   **       'customer'.
   */

   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrow,
         sizeof(HINFX_RC), &cbHrow);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLBindParameter failed (param 1)\n"))
      goto Exit;

   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hlist,
         sizeof(HINFX_RC), &cbHlist);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLBindParameter failed (param 2)\n"))
      goto Exit;

   rc = SQLExecDirect (hstmt, (UCHAR *)insertStmt, SQL_NTS);
   if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
         SQLExecDirect failed\n"))
      goto Exit;

   fprintf (stdout, "STEP 4 done...new row inserted into table
      'customer'\n");


/* STEP 5.  Free the row and list handles.
   */

   /* Free the row handle */
   rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
         &cbHrow);

   rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

   /* Free the list handle */
   rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
         SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
         &cbHlist);

   rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

   fprintf (stdout, "STEP 5 done...row and list handles freed\n");

   Exit:

/* CLEANUP: Close the statement handle.
   **       Free the statement handle.
   **       Disconnect from the datasource.
   **       Free the connection and environment handles.
   **       Exit.
   */

   /* Close the statement handle */
   SQLFreeStmt (hstmt, SQL_CLOSE);

   /* Free the statement handle */
   SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
```

```
/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout,"\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
```

Modifying a Row or Collection

The following table provides an overview of the functions that IBM Informix ODBC Driver provides for modifying rows and collections.

Function	Modification	Row	Collection
ifx_rc_delete()	Delete an element	No	Yes
ifx_rc_insert()	Insert an element	No	Yes (See the following table.)
ifx_rc_setnull()	Set the row or collection to null	Yes	Yes
ifx_rc_update()	Update the value of an element	Yes	Yes

The following table describes the collection locations into which you can insert an element. You can insert an element only at the end of a SET or MULTISET because elements in these types of collections do not have ordered positions.

	Beginning	Middle	End
List	Yes	Yes	Yes
Multiset	No	No	Yes
Set	No	No	Yes

Tip: *If you only need to insert or update a row or collection column with literal values, you do not need to use a row or collection buffer. Instead, you can explicitly list the literal value in either the INTO clause of the INSERT statement or the SET clause of the UPDATE statement.*

Each row and collection maintains a seek position that points to the current element in the row or collection. When the row or collection is created, the seek position points to the first element that is in the row or collection. All calls to client functions share the same seek position for a row or collection buffer. Therefore, one client function can affect the seek position for another client function that uses the same buffer handle. The following table describes how client functions use and modify the seek position.

Client Function	Acts On	Changes
ifx_rc_delete()	At the specified position	Sets the seek position to the value after the one that was deleted
ifx_rc_fetch()	At the specified position	Sets the seek position to the specified position
ifx_rc_insert()	Before the specified position	Sets the seek position to the specified position
ifx_rc_update()	At the specified position	Sets the seek position to the specified position

Retrieving Information About a Row or Collection

The following table provides an overview of the functions that IBM Informix ODBC Driver provides for retrieving information about rows and collections. The **ifx_rc_describe()** function returns the data types of elements in a row or collection. Chapter 6, "Client Functions," describes these functions.

Function	Information	Page
ifx_rc_count()	Number of columns	6-40
ifx_rc_describe()	Data type information	6-44
ifx_rc_isnull()	Value that indicates whether or not it is null	6-51
ifx_rc_typespec()	Type specification	6-53

Chapter

6

Client Functions

IDS

In This Chapter

The information in this chapter applies only if your database server is IBM Informix Dynamic Server.

This chapter describes the IBM Informix ODBC Driver client functions. Use these functions to access and manipulate smart large objects and rows and collections.

Calling a Client Function

This section describes the syntax of client functions, their input/output arguments, return values, and SQL_BIGINT.

SQL Syntax

The SQL syntax for a client function is:

```
{? = call client_function(?, ?,...)}
```

Use the first parameter marker ("?") only when the first parameter is an output parameter.

The following code example invokes a client function when the first parameter is an output parameter:

```
{? = call ifx_lo_open(?, ?, ?)}
```

The following code example invokes a client function when the first parameter is not an output parameter:

```
{call ifx_lo_create(?, ?, ?, ?)}
```

Function Syntax

The database server and the application both partially implement each client function. You can execute a client function with either **SQLPrepare()** and **SQLExecute()** or with **SQLExecDirect()**. You need to call **SQLBindParameter()** or **SQLBindCol()** to bind each parameter before you call **SQLExecute()** or **SQLExecDirect()**.

To execute a client function with SQLPrepare() and SQLExecute()

1. Prepare the SQL statement for the client function.
2. Bind the parameters.
3. Execute the SQL statement.

The following code examples illustrates these steps for **ifx_lo_open()**:

```
rc = SQLPrepare(hstmt, "{? = call ifx_lo_open(?, ?, ?)}",
   SQL_NTS);
rc = SQLBindParameter(...);
rc = SQLExecute(hstmt);
```

To execute a client function with SQLExecDirect()

1. Bind the parameters.
2. Execute the SQL statement.

The following code example illustrates these steps for **ifx_lo_open()**:

```
rc = SQLBindParameter(...);
rc = SQLExecDirect(hstmt, "{? = call ifx_lo_open(?, ?, ?)}",
   SQL_NTS);
```

Input and Output Parameters

Most of the input and output parameters for client functions are output parameters from the perspective of the client application. However, a client function that accepts an input/output parameter initializes the parameter internally and sends it to the database server with the request to execute the client function. Therefore, you need to pass these parameters as input/output parameters to the driver.

SQL_BIGINT

Dynamic Server supports the INT8 Informix SQL data type. By default, the driver maps INT8 to the SQL_BIGINT Informix ODBC Driver SQL data type and to the **SQL_C_CHAR** default Informix ODBC Driver C data type. However, client functions cannot access all the data type conversion functions. Therefore, you must use a data type other than **SQL_C_CHAR** when you use a value of type SQL_BIGINT.

For example, before you call **ifx_lo_specset_estbytes()**, you need to bind a variable for the *estbytes* input argument. Because *estbytes* is an SQL_BIGINT, you would normally bind *estbytes* to an **SQL_C_CHAR**. However, **SQL_C_CHAR** does not work for SQL_BIGINT for a client function. The following code example illustrates how to bind *estbytes* to an **SQL_C_LONG** instead of an **SQL_C_CHAR** for **ifx_lo_specset_estbytes()**:

```
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
   SQL_BIGINT, (UDWORD)0, 0, &estbytes, sizeof(estbytes), NULL);
rc = SQLExecDirect(hstmt, "{call ifx_lo_specset_estbytes(?, ?)}",
   SQL_NTS);
```

Return Codes

The client functions do not provide return codes. For success or failure information, see the return codes for the IBM Informix ODBC Driver function with which you call the client function (**SQLExecDirect()** or **SQLExecute()**).

Functions for Smart Large Objects

This section describes each client function that the driver provides for smart large objects. The functions are listed alphabetically. For more information, see Chapter 4, "Working with Smart Large Objects."

ifx_lo_alter()

The **ifx_lo_alter()** function alters the storage characteristics of a smart large object.

Syntax

```
ifx_lo_alter(loptr, lospec)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
loptr	SQL_INFX_UDT_FIXED	Input	Smart-large-object pointer structure
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure

Usage

The **ifx_lo_alter()** function performs the following steps to update the storage characteristics of a smart large object:

1. Gets an exclusive lock for the smart large object.
2. Uses the characteristics that are in the *lospec* smart-large-object specification structure to update the storage characteristics of the smart large object. The **ifx_lo_alter()** function lets you change the following storage characteristics:
   - Logging characteristics
   - Last-access time characteristics
   - Extent size
3. Unlocks the smart large object.

As an alternative to calling this function, you can call one of the following functions if you want to change only one of these characteristics:

- **ifx_lo_specset_flags()**
- **ifx_lo_specset_extsz()**

ifx_lo_close()

The **ifx_lo_close()** function closes a smart large object.

Syntax

```
ifx_lo_close(lofd)
```

Arguments

The function accepts the following argument.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor

Usage

The **ifx_lo_close()** function closes a smart large object. During this function, the database server tries to unlock the smart large object. If the isolation mode is repeatable read or if the lock is an exclusive lock, the database server does not release the lock until the end of the transaction.

Tip: *If you do not update a smart large object inside a BEGIN WORK transaction block, each update is a separate transaction.*

ifx_lo_col_info()

The **ifx_lo_col_info()** function updates a smart-large-object specification structure with column-level storage characteristics.

Syntax

```
ifx_lo_col_info(colname, lospec)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description

colname	SQL_CHAR	Input	Pointer to a buffer that contains the name of a database column
			This value must be in the following format:
			`database@server_name:table.column`
			If the column is in a database that is ANSI compliant, you can include the owner name. In this case, use the following format:
			`database@server_name:owner.table.column`

lospec	SQL_INFX_UDT_FIXED	I/O	Smart-large-object specification structure

Usage

The **ifx_lo_col_info()** function sets the fields for a smart-large-object specification structure to the storage characteristics for the *colname* database column. If the specified column does not have column-level storage characteristics defined, the database server uses the storage characteristics that are inherited.

Important: *You must call* ***ifx_lo_def_create_spec()*** *before you call this function.*

ifx_lo_create()

The **ifx_lo_create()** function creates and opens a new smart large object.

Syntax

```
ifx_lo_create(lospec, flags, loptr, lofd)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure that contains storage characteristics for the new smart large object
flags	SQL_INTEGER	Input	Mode in which to open the new smart large object. For more information, see "Access Modes" on page 4-25.
loptr	SQL_INFX_UDT_FIXED	I/O	Smart-large-object pointer structure
lofd	SQL_INTEGER	Output	Smart-large-object file descriptor. This file descriptor is only valid within the current database connection.

Usage

The **ifx_lo_create()** function performs the following steps to create and open a new smart large object:

1. Creates a smart-large-object pointer structure.
2. Assigns a pointer to this structure and returns this pointer in *loptr.*

3. Assigns storage characteristics for the smart large object from the smart-large-object specification structure that lospec indicates.

   If *lospec* is null, **ifx_lo_create()** uses the system-specified storage characteristics. If the smart-large-object specification structure exists but does not contain storage characteristics, **ifx_lo_create()** uses the storage characteristics from the inheritance hierarchy.

4. Opens the smart large object in the access mode that *flags* specifies.

5. Associates the smart large object with the current connection.

   When you close this connection, the database server deallocates any associated smart large objects that have a reference count of zero. The reference count indicates the number of database columns that refer to the smart large object.

6. Returns a file descriptor that identifies the smart large object.

The database server uses the default parameters that the call to **ifx_lo_create()** establishes to determine whether or not to lock or log subsequent operations on the smart large object.

ifx_lo_def_create_spec()

The **ifx_lo_def_create_spec()** function creates a smart-large-object specification structure.

Syntax

```
ifx_lo_def_create_spec(lospec)
```

Arguments

The function accepts the following argument.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	I/O	Smart-large-object specification structure

Usage

The **ifx_lo_def_create_spec()** function creates a smart-large-object specification structure and initializes the fields to null values. If you do not change these values, the null values tell the database server to use the system-specified defaults for the storage characteristics of the smart large object.

ifx_lo_open()

The **ifx_lo_open()** function opens a smart large object.

Syntax

```
ifx_lo_open(lofd, loptr, flags)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Output	Smart-large-object file descriptor. This file descriptor is only valid within the current database connection.
loptr	SQL_INFX_UDT_FIXED	Input	Smart-large-object pointer structure
flags	SQL_INTEGER	Input	Mode in which to open the smart large object. For more information, see "Access Modes" on page 4-25.

Usage

The **ifx_lo_open()** function performs the following steps to open a smart large object:

1. Opens the *loptr* smart large object in the access mode that *flags* specifies.
2. Sets the seek position to byte zero.
3. Locks the smart large object.

Important: *The database server does not check access permissions on the smart large object. Your application must make sure that the end user or application is trusted.*

As the following table describes, the access mode determines the type of lock.

Access Mode	Type of Lock
Dirty read	No lock
Read only	Shared lock
Write only, write/append, or read/write	Update lock. When you call **ifx_lo_write()** or **ifx_lo_writewithseek()** for the smart large object, the database server promotes the lock to an exclusive lock.

The database server loses this lock when the current transaction terminates. The database server obtains the lock again the next time you call a function that needs a lock.

As an alternative, you can use a BEGIN WORK transaction block and place a COMMIT WORK or ROLLBACK WORK statement after the last statement that needs to use the lock.

4. Associates the smart large object with the current connection.

   When you close this connection, the database server deallocates any associated smart large objects that have a reference count of zero. The reference count indicates the number of database columns that refer to the smart large object.

5. Returns a file descriptor that identifies the smart large object.

The database server uses the default parameters that the call to **ifx_lo_open()** establishes to determine whether or not to lock or log subsequent operations on the smart large object.

ifx_lo_read()

The **ifx_lo_read()** function reads data from an open smart large object.

Syntax

```
ifx_lo_read(lofd, buf)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
buf	SQL_CHAR	Output	Pointer to a character buffer into which the function will read the data

Usage

The **ifx_lo_read()** function reads data from an open smart large object. The read begins at the current seek position for *lofd*. You can call **ifx_lo_tell()** to obtain the current seek position.

The **ifx_lo_read()** function reads *cbValueMax* bytes of data. *cbValueMax* is an input argument for **SQLBindParameter()** and **SQLBindCol()**. Neither the size of *buf* nor *cbValueMax* can exceed 2 gigabytes. To read a smart large object that is larger than 2 gigabytes, read it in 2-gigabyte chunks. The **ifx_lo_read()** function reads this data into the user-defined buffer to which *buf* points.

If **SQLBindParameter()** or **SQLBindCol()** returns SQL_SUCCESS, then *pcbValue*, which is an argument for each of these functions, contains the number of bytes that the function read from the smart large object. If **SQLBindParameter()** or **SQLBindCol()** returns SQL_SUCCESS_WITH_INFO, then *pcbValue* contains the number of bytes that are available to read from the smart large object.

ifx_lo_readwithseek()

The **ifx_lo_readwithseek()** function performs a seek operation and then reads data from an open smart large object.

Syntax

```
ifx_lo_readwithseek(lofd, buf, offset, whence)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
buf	SQL_CHAR	Output	Pointer to a character buffer into which the function will read the data
offset	SQL_BIGINT	Input	Offset from the starting seek position, in bytes. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *offset*, use **SQL_C_LONG** or **SQL_C_SHORT**. For more information, see "SQL_BIGINT" on page 6-5.
whence	SQL_INTEGER	Input	Starting seek position. The possible values are: ■ LO_SEEK_CUR: The current seek position in the smart large object ■ LO_SEEK_END: The end of the smart large object ■ LO_SEEK_SET: The start of the smart large object

Usage

The **ifx_lo_readwithseek()** function performs a seek operation and then reads data from an open smart large object. The read begins at the seek position of *lofd* that the *offset* and *whence* arguments indicate.

The **ifx_lo_readwithseek()** function reads *cbValueMax* bytes of data. *cbValueMax* is an input argument for **SQLBindParameter()** and **SQLBindCol()**. Neither the size of *buf* nor *cbValueMax* can exceed 2 gigabytes. To read a smart large object that is larger than 2 gigabytes, read it in 2-gigabyte chunks. The **ifx_lo_readwithseek()** function reads this data into the user-defined buffer to which *buf* points.

If **SQLBindParameter()** or **SQLBindCol()** returns SQL_SUCCESS, then *pcbValue*, which is an argument for each of these functions, contains the number of bytes that the function read from the smart large object. If **SQLBindParameter()** or **SQLBindCol()** returns SQL_SUCCESS_WITH_INFO, then *pcbValue* contains the number of bytes that are available to read from the smart large object.

ifx_lo_seek()

The **ifx_lo_seek()** function sets the file position for the next read or write operation on an open smart large object.

Syntax

```
ifx_lo_seek(lofd, offset, whence, seek_pos)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
offset	SQL_BIGINT	Input	Offset from the starting seek position, in bytes. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *offset*, use **SQL_C_LONG** or **SQL_C_SHORT**. For more information, see "SQL_BIGINT" on page 6-5.
whence	SQL_INTEGER	Input	Starting seek position. The possible values are: ■ LO_SEEK_CUR: The current seek position in the smart large object ■ LO_SEEK_END: The end of the smart large object ■ LO_SEEK_SET: The start of the smart large object
seek_pos	SQL_BIGINT	I/O	New seek position. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *seek_pos*, use **SQL_C_LONG**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_seek()** function sets the seek position of *lofd* to the position that the *offset* and *whence* arguments indicate.

ifx_lo_specget_estbytes()

The **ifx_lo_specget_estbytes()** function gets the estimated number of bytes from a smart-large-object specification structure.

Syntax

```
ifx_lo_specget_estbytes(lospec, estbytes)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
estbytes	SQL_BIGINT	Output	Estimated final size of the smart large object, in bytes. This estimate is an optimization hint for the smart-large-object optimizer. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *estbytes*, use **SQL_C_LONG**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_specget_estbytes()** function gets the estimated number of bytes from a smart-large-object specification structure.

ifx_lo_specget_extsz()

The **ifx_lo_specget_extsz()** function gets the allocation extent from a smart-large-object specification structure.

Syntax

```
ifx_lo_specget_extsz(lospec, extsz)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
extsz	SQL_INTEGER	Output	Extent size of the smart large object, in bytes. This value is the size of the allocation extents to be allocated for the smart large object when the database server writes beyond the end of the current extent. This value overrides the estimate that the database server generates for how large an extent should be.

Usage

The **ifx_lo_specget_extsz()** function gets the allocation extent from a smart-large-object specification structure.

ifx_lo_specget_flags()

The **ifx_lo_specget_flags()** function gets the create-time flags from a smart-large-object specification structure.

Syntax

```
ifx_lo_specget_flags(lospec, flags)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
flags	SQL_INTEGER	Output	Create-time flags. For more information, see "Access Modes" on page 4-25.

Usage

The **ifx_lo_specget_flags()** function gets the create-time flags from a smart-large-object specification structure.

ifx_lo_specget_maxbytes()

The **ifx_lo_specget_maxbytes()** function gets the maximum number of bytes from a smart-large-object specification structure.

Syntax

```
ifx_lo_specget_maxbytes(lospec, maxbytes)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
maxbytes	SQL_BIGINT	Input	Maximum size, in bytes, of the smart large object. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *maxbytes*, use **SQL_C_LONG**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_specget_maxbytes()** function gets the maximum number of bytes from a smart-large-object specification structure.

ifx_lo_specget_sbspace()

The **ifx_lo_specget_sbspace()** function gets the sbspace name from a smart-large-object specification structure.

Syntax

```
ifx_lo_specget_sbspace(lospec, sbspace)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
sbspace	SQL_CHAR	Output	Name of the sbspace for the smart large object. An sbspace name can be up to 18 characters long and must be null terminated.

Usage

The **ifx_lo_specget_sbspace()** function returns the name of the sbspace in which to store the smart large object. The function copies up to (*pcbValue*-1) bytes into the *sbspace* buffer and makes sure that it is null terminated. *pcbValue* is an argument for **SQLBindParameter()** and **SQLBindCol()**.

ifx_lo_specset_estbytes()

The **ifx_lo_specset_estbytes()** function sets the estimated number of bytes in a smart-large-object specification structure.

Syntax

```
ifx_lo_specset_estbytes(lospec, estbytes)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
estbytes	SQL_BIGINT	Input	Estimated final size of the smart large object, in bytes. This estimate is an optimization hint for the smart large object optimizer. This value cannot exceed 2 gigabytes. If you do not specify an *estbytes* value when you create a new smart large object, the database server gets the value from the inheritance hierarchy of storage characteristics. Do not change this system value unless you know the estimated size for the smart large object. If you do set the estimated size for a smart large object, do not specify a value much higher than the final size of the smart large object. Otherwise, the database server might allocate unused storage. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *estbytes*, use **SQL_C_LONG** or **SQL_C_SHORT**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_specset_estbytes()** function sets the estimated number of bytes in a smart-large-object specification structure.

ifx_lo_specset_extsz()

The **ifx_lo_specset_extsz()** function sets the allocation extent size in a smart-large-object specification structure.

Syntax

```
ifx_lo_specset_extsz(lospec, extsz)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
extsz	SQL_INTEGER	Input	Extent size of the smart large object, in bytes. This value specifies the size of the allocation extents to be allocated for the smart large object when the database server writes beyond the end of the current extent. This value overrides the estimate that the database server generates for how large an extent should be. If you do not specify an *extsz* value when you create a new smart large object, the database server attempts to optimize the extent size based on past operations on the smart large object and other storage characteristics (such as maximum bytes) that it obtains from the inheritance hierarchy of storage characteristics. Do not change this system value unless you know the allocation extent size for the smart large object. Only applications that encounter severe storage fragmentation should ever set the allocation extent size. For such applications, make sure you know exactly the number of bytes by which to extend the smart large object.

Usage

The **ifx_lo_specset_extsz()** function sets the allocation extent size in a smart-large-object specification structure.

ifx_lo_specset_flags()

The **ifx_lo_specset_flags()** function sets the create-time flags in a smart-large-object specification structure.

Syntax

```
ifx_lo_specset_flags(lospec, flags)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
flags	SQL_INTEGER	Input	Create-time flags. For more information, see "Create-Time Flags" on page 4-8.

Usage

The **ifx_lo_specset_flags()** function sets the create-time flags in a smart-large-object specification structure.

ifx_lo_specset_maxbytes()

The **ifx_lo_specset_maxbytes()** function sets the maximum number of bytes in a smart-large-object specification structure.

Syntax

```
ifx_lo_specset_maxbytes(lospec, maxbytes)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
maxbytes	SQL_BIGINT	Input	Maximum size of the smart large object, in bytes. This value cannot exceed 2 gigabytes. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *maxbytes*, use **SQL_C_LONG** or **SQL_C_SHORT**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_specset_maxbytes()** function sets the maximum number of bytes in a smart-large-object specification structure.

ifx_lo_specset_sbspace()

The **ifx_lo_specset_sbspace()** function sets the sbspace name in a smart-large-object specification structure.

Syntax

```
ifx_lo_specset_sbspace(lospec, sbspace)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lospec	SQL_INFX_UDT_FIXED	Input	Smart-large-object specification structure
sbspace	SQL_CHAR	Input	Name of the sbspace for the smart large object. An sbspace name can be up to 18 characters long and must be null terminated. If you do not specify an *sbspace* when you create a new smart large object, the database server obtains the sbspace name from either the column information or from the SBSPACENAME parameter of the ONCONFIG file.

Usage

The **ifx_lo_specset_sbspace()** function uses *pcbValue* to determine the length of the sbspace name. *pcbValue* is an argument for **SQLBindParameter()** and **SQLBindCol()**.

ifx_lo_stat()

The **ifx_lo_stat()** function initializes a smart-large-object status structure.

Syntax

```
ifx_lo_stat(lofd, lostat)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
lostat	SQL_INFX_UDT_FIXED	I/O	Smart-large-object status structure

Usage

Before you call **ifx_lo_stat()**, call **SQLGetInfo()** to get the size of the smart-large-object status structure. Use this size to allocate memory for the structure.

The **ifx_lo_stat()** function allocates a smart-large-object status structure and initializes it with the status information for the smart large object.

ifx_lo_stat_atime()

The **ifx_lo_stat_atime()** function retrieves the last access time for a smart large object.

Syntax

```
ifx_lo_stat_atime(lostat, atime)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lostat	SQL_INFX_UDT_FIXED	Input	Smart-large-object status structure
atime	SQL_INTEGER	Output	Time of the last access for a smart large object, in seconds. The database server maintains the time of last access only if the LO_KEEP_LASTACCESS _TIME flag is set for the smart large object.

Usage

The **ifx_lo_stat_atime()** function retrieves the last access time for a smart large object.

ifx_lo_stat_cspec()

The **ifx_lo_stat_cspec()** function retrieves a smart-large-object specification structure.

Syntax

```
ifx_lo_stat_cspec(lostat, lospec)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lostat	SQL_INFX_UDT_FIXED	Input	Smart-large-object status structure
lospec	SQL_INFX_UDT_FIXED	Output	Smart-large-object specification structure

Usage

The **ifx_lo_stat_cspec()** function retrieves a smart-large-object specification structure and returns a pointer to the structure.

ifx_lo_stat_ctime()

The **ifx_lo_stat_ctime()** function retrieves the time of the last change of a smart large object.

Syntax

```
ifx_lo_stat_ctime(lostat, ctime)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lostat	SQL_INFX_UDT_FIXED	Input	Smart-large-object status structure
ctime	SQL_INTEGER	Output	Time of the last change of the smart large object, in seconds. The last change in status includes modification of storage characteristics, including a change in the number of references and writes to the smart large object.

Usage

The **ifx_lo_stat_ctime()** function retrieves the time of the last change of a smart large object.

ifx_lo_stat_refcnt()

The **ifx_lo_stat_refcnt()** function retrieves the number of references to a smart large object.

Syntax

```
ifx_lo_stat_refcnt(lostat, refcount)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lostat	SQL_INFX_UDT_FIXED	Input	Smart-large-object status structure
refcount	SQL_INTEGER	Output	Number of references to a smart large object. This value is the number of database columns that refer to the smart large object.

Usage

The **ifx_lo_stat_refcnt()** function retrieves the number of references to a smart large object.

A database server can remove a smart large object and reuse any resources that are allocated to it when the reference count for the smart large object is zero and one of the following events occurs:

- The transaction in which the reference count is decremented to zero commits.
- The connection during which the smart large object was created terminates, but the reference count is not incremented.

  The database server increments a reference counter when it stores the smart-large-object pointer structure for a smart large object in a row.

ifx_lo_stat_size()

The **ifx_lo_stat_size()** function retrieves the size of a smart large object.

Syntax

```
ifx_lo_stat_size(lostat, size)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lostat	SQL_INFX_UDT_FIXED	Input	Smart-large-object status structure
size	SQL_BIGINT	Output	Size of a smart large object, in bytes. This value cannot exceed 2 gigabytes. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *size*, use **SQL_C_LONG**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_stat_size()** function retrieves the size of a smart large object.

ifx_lo_tell()

The **ifx_lo_tell()** function retrieves the current file or seek position for an open smart large object.

Syntax

```
ifx_lo_tell(lofd, seek_pos)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
seek_pos	SQL_BIGINT	I/O	New seek position, which is the offset for the next read or write operation on the smart large object. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *seek_pos*, use **SQL_C_LONG**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_tell()** function retrieves the current file or seek position for an open smart large object.

This function works correctly for smart large objects up to 2 gigabytes in size.

ifx_lo_truncate()

The **ifx_lo_truncate()** function truncates a smart large object at the specified position.

Syntax

```
ifx_lo_truncate(lofd, offset)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
offset	SQL_BIGINT	Input	End of the smart large object. If this value exceeds the end of the smart large object, the function extends the smart large object. If this value is less than the end of the smart large object, the database server reclaims all storage from the offset position to the end of the smart large object. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *offset*, use **SQL_C_LONG** or **SQL_C_SHORT**. For more information, see "SQL_BIGINT" on page 6-5.

Usage

The **ifx_lo_truncate()** function sets the end of a smart large object to the location that the *offset* argument specifies.

ifx_lo_write()

The **ifx_lo_write()** function writes data to an open smart large object.

Syntax

```
ifx_lo_write(lofd, buf)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor
buf	SQL_CHAR	Input	Buffer that contains the data that the function writes to the smart large object. The size of the buffer cannot exceed 2 gigabytes.

Usage

The **ifx_lo_write()** function writes data to an open smart large object. The write begins at the current seek position for *lofd*. You can call **ifx_lo_tell()** to obtain the current seek position.

The **ifx_lo_write()** function writes *cbValueMax* bytes of data. *cbValueMax* is an input argument for **SQLBindParameter()** and **SQLBindCol()**. Neither the size of *buf* nor *cbValueMax* can exceed 2 gigabytes. To write to a smart large object that is larger than 2 gigabytes, write to it in 2-gigabyte chunks. The **ifx_lo_write()** function gets the data from the user-defined buffer to which *buf* points.

If **SQLExecDirect()** or **SQLExecute()** returns SQL_SUCCESS_WITH_INFO, then the database server wrote less than *cbValueMax* bytes of data to the smart large object and *pcbValue*, which is an argument for each of these functions, contains the number of bytes that the function wrote. This condition can occur when the sbspace runs out of space.

ifx_lo_writewithseek()

The **ifx_lo_writewithseek()** function performs a seek operation and then writes data to an open smart large object.

Syntax

```
ifx_lo_writewithseek(lofd, buf, offset, whence)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
lofd	SQL_INTEGER	Input	Smart-large-object file descriptor.
buf	SQL_CHAR	Input	Buffer that contains the data that the function writes to the smart large object. The size of the buffer must not exceed 2 gigabytes.
offset	SQL_BIGINT	Input	Offset from the starting seek position, in bytes. Instead of using the default Informix ODBC Driver C data type of **SQL_C_CHAR** for *offset*, use **SQL_C_LONG** or **SQL_C_SHORT**. For more information, see "SQL_BIGINT" on page 6-5.
whence	SQL_INTEGER	Input	Starting seek position. The possible values are: ■ LO_SEEK_CUR: The current seek position in the smart large object ■ LO_SEEK_END: The end of the smart large object ■ LO_SEEK_SET: The start of the smart large object

Usage

The **ifx_lo_writewithseek()** function performs a seek operation and then writes data to an open smart large object. The write begins at the seek position of *lofd* that the *offset* and *whence* arguments indicate.

The **ifx_lo_writewithseek()** function writes *cbValueMax* bytes of data. *cbValueMax* is an input argument for **SQLBindParameter()** and **SQLBindCol()**. Neither the size of *buf* nor *cbValueMax* can exceed 2 gigabytes. To write to a smart large object that is larger than 2 gigabytes, write to it in 2-gigabyte chunks. The **ifx_lo_writewithseek()** function gets the data from the user-defined buffer to which *buf* points.

If **SQLExecDirect()** or **SQLExecute()** returns SQL_SUCCESS_WITH_INFO, then the database server wrote less than *cbValueMax* bytes of data to the smart large object and *pcbValue*, which is an argument for each of these functions, contains the number of bytes that the function wrote. This condition can occur when the sbspace runs out of space.

Functions for Rows and Collections

This section describes each client function that Informix ODBC Driver provides for rows and collections. The functions are listed alphabetically. For more information about rows and collections, see Chapter 5, "Working with Rows and Collections."

ifx_rc_count()

The **ifx_rc_count()** function returns the number of elements or fields that are in a row or collection.

Syntax

```
ifx_rc_count(rowcount, rchandle)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
rowcount	SQL_SMALLINT	Output	Number of elements or fields that are in the row or collection
rchandle	HINFX_RC	Input	Handle for a row or collection buffer

Usage

The **ifx_rc_count()** function returns the number of elements or fields that are in the row or collection.

ifx_rc_create()

The **ifx_rc_create()** function creates a buffer for a row or collection.

Syntax

```
ifx_rc_create(rchandle, typespec)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
rchandle	HINFX_RC	Output	Handle for a row or collection buffer
typespec	SQL_CHAR	Input	Type specification for the buffer. See the following table.

The following table describes the syntax for the *typespec* argument.

Type of Buffer	Syntax	Example
Unfixed-type collection	`COLLECTION`	`COLLECTION`
Fixed-type collection	`COLLECTION {SET | MULTISET | LIST} (`*type* `not null)`	`COLLECTION SET (int not null)`
	or	or
	`{SET | MULTISET | LIST} (`*type* `not null)`	`SET (int not null)`
	where *type* is the Informix SQL data type for the elements in the collection	
Unfixed-type row	`ROW`	`ROW`

(1 of 2)

Type of Buffer	Syntax	Example
Fixed-type row	`ROW [`*"name"*`] (`*field_id type* `[, `*field_id type*`, ...])` where: ■ *name* is an optional name for the entire row ■ *field_id* is the name for a field ■ *type* is the Informix SQL data type for the field	`ROW "employee_t" (name char(255), id_num int, dept int)`

(2 of 2)

Usage

The **ifx_rc_create()** function allocates memory for a row or collection buffer and returns a handle to the buffer. The following table describes how the function initializes the buffer.

Type of Buffer	Initial Value for the Row or Collection	Initial Value for the Contents of the Row or Collection
Fixed-type collection	Non-null	Empty
Fixed-type row	Non-null	Each value is null
Unfixed-type collection	Null	Empty
Unfixed-type row	Null	Empty

For a row, the function sets the seek position to element number one. An empty collection buffer does not have a seek position.

ifx_rc_delete()

The **ifx_rc_delete()** function deletes an element from a collection.

Syntax

```
ifx_rc_delete(rchandle, action, jump)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
rchandle	HINFX_RC	Input	Handle for a collection buffer
action	SQL_SMALLINT	Input	Location of the element relative to the seek position. The possible values are: ■ SQL_INFX_RC_ABSOLUTE: Element number *jump* where the first element in the buffer is element number one ■ SQL_INFX_RC_CURRENT: Current element ■ SQL_INFX_RC_FIRST: First element ■ SQL_INFX_RC_LAST: Last element ■ SQL_INFX_RC_NEXT: Next element ■ SQL_INFX_RC_PRIOR: Previous element ■ SQL_INFX_RC_RELATIVE: Element that is *jump* elements past the current element s
jump	SQL_SMALLINT	Input	Offset when *action* is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE

Usage

The **ifx_rc_delete()** function deletes an element from a collection from the location that is specified by *action* and *jump*. The function sets the seek position to the value that was just deleted. It is not possible to delete an element from a row.

ifx_rc_describe()

The **ifx_rc_describe()** function returns descriptive information about the data type for a row or collection or for an element that is in a row or collection.

Syntax

```
ifx_rc_describe(rchandle, fieldnum, fieldname, typecode,
  columnsize, decdigits, nullable, typename, typeowner)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
rchandle	HINFX_RC	Input	Handle for a row or collection buffer
fieldnum	SQL_SMALLINT	Input	Field number. If this value is 0, the function returns information for the entire row or collection. For a collection, any value other than 0 causes the function to return information for the elements that are in the collection. For a row, this value specifies the element for which the function returns information.
fieldname	SQL_CHAR	Output	Field name. The function returns this value only for an element that is in a row.
typecode	SQL_SMALLINT	Output	Informix ODBC Driver SQL data type of the element
columnsize	SQL_INTEGER	Output	Column size. For a character element, this value is the size of the column, in bytes. For a numeric element, this value is the precision. For other data types, the function does not return this value.
decdigits	SQL_SMALLINT	Output	Decimal digits. For a numeric element, this value is the number of digits to the right of the decimal point. For other data types, the function does not return this value.

(1 of 2)

Argument	Type	Use	Description
nullable	SQL_SMALLINT	Output	Null indicator. The possible values are: ■ SQL_NO_NULLS ■ SQL_NULLABLE
typename	SQL_CHAR	Output	Type name. For a named row, this value is the name of the row. For collections and unnamed rows, the function does not return this value.
typeowner	SQL_CHAR	Output	Type owner. This value is the name of the owner of the data type. This name can be up to 18 characters long.

(2 of 2)

Usage

The **ifx_rc_describe()** function returns information about the data type for a row or collection or for an element that is in a row or collection. For elements that are in a collection, this information is the same for all elements that are in the collection. This function does not change the seek position.

ifx_rc_fetch()

The **ifx_rc_fetch()** function retrieves the value of an element that is in a row or collection.

Syntax

```
ifx_rc_fetch(result, rchandle, action, jump)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
result	Data type of the element	Output	Retrieved value
rchandle	HINFX_RC	Input	Handle for a row or collection buffer
action	SQL_SMALLINT	Input	Location of the element relative to the seek position. The possible values are: ■ SQL_INFX_RC_ABSOLUTE: Element number *jump* where the first element in the buffer is element number one ■ SQL_INFX_RC_CURRENT: Current element ■ SQL_INFX_RC_FIRST: First element ■ SQL_INFX_RC_LAST: Last element ■ SQL_INFX_RC_NEXT: Next element ■ SQL_INFX_RC_PRIOR: Previous element ■ SQL_INFX_RC_RELATIVE: Element that is *jump* elements past the current element
jump	SQL_SMALLINT	Input	Offset when *action* is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE

Usage

The **ifx_rc_fetch()** function retrieves the value of the element that is specified by *action* and *jump* and returns the value in *result*. The function sets the seek position to the value that was just fetched.

ifx_rc_free()

The **ifx_rc_free()** function frees a row or collection handle.

Syntax

```
ifx_rc_free(rchandle)
```

Arguments

The function accepts the following argument.

Argument	Type	Use	Description
rchandle	HINFX_RC	Input	Handle for a row or collection buffer

Usage

The **ifx_rc_free()** function frees all the resources that are associated with a row or collection handle and frees the handle.

ifx_rc_insert()

The **ifx_rc_insert()** function inserts a new element into a collection.

Syntax

```
ifx_rc_insert(rchandle, value, action, jump)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
rchandle	HINFX_RC	Input	Handle for a collection buffer
value	Data type of the element	Input	Value to insert
action	SQL_SMALLINT	Input	Location of the element relative to the seek position. The possible values are: ■ SQL_INFX_RC_ABSOLUTE: Element number *jump* where the first element in the buffer is element number one ■ SQL_INFX_RC_CURRENT: Current element ■ SQL_INFX_RC_FIRST: First element ■ SQL_INFX_RC_LAST: Last element ■ SQL_INFX_RC_NEXT: Next element ■ SQL_INFX_RC_PRIOR: Previous element ■ SQL_INFX_RC_RELATIVE: Element that is *jump* elements past the current element
jump	SQL_SMALLINT	Input	Offset when *action* is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE

Usage

The **ifx_rc_insert()** function inserts a new element into a collection before the location that is specified by *action* and *jump*. The function sets the seek position to the value that was just inserted. It is not possible to insert a new element into a row.

The following table describes the allowable insertion locations for each type of collection.

Type of Collection	Allowable Insertion Locations
List	Anywhere in the buffer
Set or multiset	At the end of the buffer

If the seek position specified by *action* and *jump* exceeds the end of the buffer, **ifx_rc_insert()** appends the new element at the end of the buffer. Likewise, if the seek position specified by *action* and *jump* precedes the beginning of the buffer, **ifx_rc_insert()** inserts the new element at the beginning of the buffer. If *action* specifies an insertion point other than the end for a set or multiset, **ifx_rc_insert()** fails.

For example, if *action* is SQL_INFX_RC_LAST, the function inserts the new element before the last element. To append a new element, take one of the following actions:

- Set the seek position to the end of the buffer and set *action* to SQL_INFX_RC_NEXT.
- Set *action* to SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE and set *jump* to a value that exceeds the end of the buffer.

To insert a new element at the beginning of a buffer, set *action* to SQL_INFX_RC_FIRST.

ifx_rc_isnull()

The **ifx_rc_isnull()** function returns a value that indicates whether or not a row or collection is null.

Syntax

```
ifx_rc_isnull(nullflag, rchandle)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
nullflag	SQL_SMALLINT	Output	Flag that indicates whether or not a row or collection is null. The possible values are: ■ `TRUE` ■ `FALSE`
rchandle	HINFX_RC	Input	Handle for a row or collection buffer

Usage

The **ifx_rc_isnull()** function returns a value that indicates whether or not a row or collection is null.

ifx_rc_setnull()

The **ifx_rc_setnull()** function sets a row or collection to null.

Syntax

```
ifx_rc_setnull(rchandle)
```

Arguments

The function accepts the following argument.

Argument	Type	Use	Description
rchandle	HINFX_RC	Input	Handle for a row or collection buffer

Usage

The **ifx_rc_setnull()** function sets a row or collection to null. The **ifx_rc_setnull()** function does not set each element within the row or collection to null.

ifx_rc_typespec()

The **ifx_rc_typespec()** function returns the type specification for a row or collection.

Syntax

```
ifx_rc_typespec(typespec, rchandle, flag)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
typespec	SQL_CHAR	Output	Type specification. The format for this value is the same as the type specification syntax for **ifx_rc_create()**. For more information on **ifx_rc_create()** arguments, see "Arguments" on page 6-41.
rchandle	HINFX_RC	Input	Handle for a row or collection buffer
flag	SQL_SMALLINT	Input	Flag that specifies whether to return the current or original type specification. If this value is TRUE, the function returns the original type specification. Otherwise, the function returns the current type specification.

Usage

The **ifx_rc_typespec()** function returns the type specification for a row or collection.

ifx_rc_update()

The **ifx_rc_update()** function updates the value for an element that is in a row or collection.

Syntax

```
ifx_rc_update(rchandle, value, action, jump)
```

Arguments

The function accepts the following arguments.

Argument	Type	Use	Description
rchandle	HINFX_RC	Input	Handle for a row or collection buffer
value	Data type of the element	Input	Value with which to update the element
action	SQL_SMALLINT	Input	Location of the element relative to the seek position. The possible values are: ■ SQL_INFX_RC_ABSOLUTE: Element number *jump* where the first element in the buffer is element number one ■ SQL_INFX_RC_CURRENT: Current element ■ SQL_INFX_RC_FIRST: First element ■ SQL_INFX_RC_LAST: Last element ■ SQL_INFX_RC_NEXT: Next element ■ SQL_INFX_RC_PRIOR: Previous element ■ SQL_INFX_RC_RELATIVE: Element that is *jump* elements past the current element
jump	SQL_SMALLINT	Input	Offset when *action* is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE

Usage

The **ifx_rc_update()** function updates the value for an element that immediately precedes the location that is specified by *action* and *jump*. The function sets the seek position to the value that was just updated.

Chapter

7

Improving Application Performance

In This Chapter

This chapter suggests ways to improve performance of IBM Informix ODBC Driver applications.

Case-Sensitive Catalog Functions

By default, the database server creates tables and columns in lowercase parameters and always passes catalog functions as lowercase, regardless of what case the SQL statement uses.

The following example creates a table called **TestTable** and fetches the characteristics of the table:

```
SQLExecDirect( hstmt, "create table \"TestTable\" ( a int )",
   SQL_NTS )
SQLColumns( hstmt, NULL, 0, NULL, 0, "TESTTABLE", SQL_NTS,
   NULL, 0 )
```

UNIX

To change the default database server behavior on UNIX, set the **DELIMIDENT** environment variable. For information on setting environment variables, see the *IBM Informix Guide to SQL: Reference*.

The following example creates a table called **TestTable** and fetches the characteristics of the TestTable table:

```
SQLExecDirect( hstmt, "create table 'TestTable' ( a char(5) )",
   SQL_NTS )
SQLColumns( hstmt, NULL, 0, NULL, 0, 'TestTable', SQL_NTS,
   NULL, 0 )
```

♦

Windows

To change the default database server behavior on Windows, set the **DELIMIDENT** environment variable to "y" in the registry with **Setnet32** and use double quotes around identifiers. For information on setting environment variables, see the *IBM Informix Guide to SQL: Reference*. ♦

If you set the **DELIMIDENT** environment variable before you run the statements, the database server differentiates between the two table names (**testtable** and **TestTable**) and returns the characteristics of the correct table.

Important: *If you set the **DELIMIDENT** environment variable, use single quotes to quote string constants. If you use double quotes, you get an error.*

For example, if you set the **DELIMIDENT** environment variable, you get an error when you run the first statement in the following example. The second statement in the example inserts the string value correctly.

```
SQLExecDirect( hstmt, "insert into \"TestTable\" values
   ( ”abcde” )", SQL_NTS ) inserts the string value properly.
SQLExecDirect( hstmt, "insert into \"TestTable\" values
   ( 'abcde' )", SQL_NTS )
```

Both cases send a query to the database server, but in Case 1, **SQLColumns** must evaluate the query and form a result set that it must send to the client. In Case 2, **SQLDescribeCol** gets the result-column information and the order of the columns within the table without returning the information to the client, thus improving performance.

Avoid using **SQLColumns** to determine characteristics of a table. Instead, use a dummy query with **SQLDescribeCol**.

Connection Level Optimizations

Establishing a connection to a database is an expensive process. Optimally, an application should perform as many tasks as possible while a connection is open. This can by achieved by:

- Pooling connections when using Windows Driver Manager
- Using multiple statement handles on the same connection handle

Also, you can fine tune application performance by setting the following connection level attributes:

- AutoCommit optimization
- Message transfer optimization (OPTMSG)
- Open-Fetch-Close optimization (OPTOFC)

For more information on connection level attributes, see Chapter 2, "Configuring Data Sources."

Optimizing Query Execution

Consider the following when using prepared SQL queries:

- SQLExecDirect is optimized for a single execution of an SQL statement. Thus it should be used for SQL queries that are not executed repeatedly.
- In cases where SQL queries are executed multiple times, using SQLPrepare and SQLExecute improves performance. Typically, you can do this with input and output parameters.
- SPL routines can be called from an ODBC application to perform certain SQL tasks and to expand what you can accomplish with SQL alone. Because SPL is native to the database and SPL routines are parsed and optimized at creation, rather than at runtime, SPL routines can improve performance for some tasks. SPL routines can also reduce traffic between a client application and the database server and reduce program complexity.

Inserting Multiple Rows

Use an insert cursor to efficiently insert rows into a table in bulk. To create an insert cursor, set the SQL_ENABLE_INSERT_CURSOR attribute using **SQLSetStmtOption**, then call **SQLParamOptions** with the number of rows as a parameter. You can create an insert cursor for data types TEXT, BYTE, VARCHAR, LVARCHAR, and opaque.

When you open an insert cursor, a buffer is created in memory to hold a block of rows. The buffer receives rows of data as the program produces them; then they are passed to the database server in a block when the buffer is full. The buffer reduces the amount of communication between the program and the database server. As a result, the insertions go faster.

Automatically Freeing a Cursor

When an application uses a cursor, it usually sends a FREE statement to the database server to deallocate memory assigned to a cursor once it no longer needs that cursor. Execution of this statement involves of message requests between the application and the database server. When the AUTOFREE is enabled, IBM Informix ODBC Driver saves message requests because it does not need to execute the FREE statement. When the database server closes an insert cursor, it automatically frees the memory that it has allocated for it.

Enabling the AUTOFREE Feature

You can enable the AUTOFREE feature for an ODBC application in either of the following ways:

- Set the SQL_INFX_ATTR_AUTO_FREE attribute using **SQLSetConnectAttr**.

  When you use **SQLSetConnectAttr** to enable this attribute, all new allocated statements for that connection inherit the attribute value. The only way to change this attribute value per statement is to set and reset it again as a statement attribute. The default is DISABLED for the connection attribute.

- Set the SQL_INFX_ATTR_AUTO_FREE attribute using **SQLSetStmtAttr**.

The SQL_INFX_ATTR_AUTO_FREE attribute can be set in any connection state between C2 and C5 (both included) when setting it using **SQLSetConnectAttr**, whereas it can be set using **SQLSetStmtAttr** only when the statement is in S1 (allocated) state. The value of the SQL_INFX_ATTR_AUTO_FREE attribute can be retrieved using **SQLGetConnectAttr** or **SQLSetStmtAttr**.

Using the AUTOFREE Feature

The AUTOFREE feature only works with result generating statements executed using **SQLExecDirect**, as it opens the cursor which is then closed and released by the corresponding **SQLCloseCursor** or **SQLFreeStmt**. The AUTOFREE feature does not work when the application has to prepare a statement once and then execute it several times (for example, using **SQLPrepare** to prepare and then executing it by calling **SQLExecute** several times). When you close the cursor using **SQLCloseCursor** after **SQLExecute**, it only closes the cursor but does not release the cursor memory on the database server side. But if you close the cursor using **SQLFreeStmt** with SQL_CLOSE or SQL_DROP, it not only closes and releases the cursor, but it also unprepares the statement. In the latter case there is savings of a network roundtrip, but the application is unable to execute the statement again until it reprepares it.

When AUTOFREE is enabled, the application sees an improvement in the network performance when the application closes the cursor using **SQLCloseCursor** or **SQLFreeStmt** with SQL_DROP.

Delaying Execution of the SQLPREPARE Statement

You can defer execution of the **SQLPrepare** statement by enabling the deferred-PREPARE feature. This feature works primarily with dynamic SQL statements where the application does a series of **SQLPrepare** and **SQLExecute** statements. It optimizes the number of round-trip messages to the database server by not sending **SQLPrepare** statements to the database server until the application calls **SQLExecute** on that statement.

When deferred-PREPARE is enabled, the following behavior is expected of the application:

- Execution of **SQLPrepare** does not put the statement in a prepared state.
- Syntax errors in an **SQLPrepare** statement are not known until the statement is executed because the SQL statement is never sent to the database server until it is executed. If open-fetch-close optimization is turned on, errors are not returned to the client until the first fetch, because open-fetch-close optimizes the OPEN/FETCH so that OPEN is sent on the first fetch.
- **SQLColAttribute**, **SQLDescribeCol**, **SQLNumResultCols**, and **SQLNumParams** always return HY010 (function sequence error) if called after **SQLPrepare** but before **SQLExecute** by the application.
- **SQLCopyDesc** returns HY010 if the source descriptor handle is an IRD if called after **SQLPrepare** but before **SQLExecute** by the application.
- **SQLGetDescField** and **SQLGetDescRec** return HY010 if the descriptor handle is an IRD if called after **SQLPrepare** but before **SQLExecute** by the application.

You can enable the deferred-PREPARE feature for an ODBC application in either of the following ways:

- Set the SQL_INFX_ATTR_DEFERRED_PREPARE attribute using **SQLSetConnectAttr**.

  When you use **SQLSetConnectAttr** to enable this attribute, all new allocated statements for that connection inherit the attribute value. The only way to change this attribute value per statement, is to set/reset it again as a statement attribute. The default is DISABLED for the connection attribute.

- Set the SQL_INFX_ATTR_DEFERRED_PREPARE attribute using **SQLSetStmtAttr**.

The SQL_INFX_ATTR_DEFERRED_PREPARE attribute can be set in any connection state between C2 and C5 (both included) when setting it using **SQLSetConnectAttr**, whereas it can be set using **SQLSetStmtAttr** only when the statement is in S1 (allocated) state. The value of the SQL_INFX_ATTR_DEFERRED_PREPARE attribute can be retrieved using **SQLGetConnectAttr** or **SQLSetStmtAttr**.

Setting the Fetch Array Size for Simple-Large-Object Data

To reduce the network overhead for fetches involving multiple rows of simple-large-object data, you can set the array size so when the driver receives a multiple-row fetch request, it optimizes the fetch buffer size and the internal fetch array size, and eliminates a round trip to the database server for every simple large object. Setting the array size greater than `1` can result in a performance improvement even for other types of data because it has the side effect of automatically increasing the fetch buffer size if necessary. (If the number of rows specified will fit in the current fetch buffer, setting it will have little effect.)

An application can request that multiple rows be returned to it by setting the statement attribute SQL_ATTR_ROW_ARRAY_SIZE or setting the ARD header field SQL_DESC_ARRAY_SIZE to a value greater than one, and then calling either **SQLFetch** or **SQLFetchScroll**. (The default value of SQL_ATTR_ROW_ARRAY_SIZE is one.) The driver then recognizes when it receives a multiple-row fetch request and optimizes the settings for the fetch buffer size and the internal fetch array size. Settings for these are based on the internal tuple size, the user setting of row array size, and the current setting of fetch array size.

You cannot use the internal fetch array feature under the following conditions:

- When OPTOFC and deferred-PREPARE are both enabled

  To use the fetch array feature, the driver is dependent upon knowing how large a row is going to be, as received from the database server, prior to sending the fetch request to the database server. When both of these are enabled, this information is not available until after a fetch is performed.

- When using scroll cursors

  There are separate internal client-to-server protocols used for scroll cursors that are distinct from those used for fetching arrays. The database server does not support simple large object columns in a scroll cursor. An error will be returned.

- When using **SQLGetData**

  In order for the driver to utilize the fetch array feature, it has to be able to tell the database server how much data it is prepared to receive at the time of the fetch request. Calls to **SQLGetData** take place after **SQLFetch**.

  According to the ODBC standard, when using block cursors, the application must call **SQLSetPos** to position the cursor on a particular row prior to calling **SQLGetData**. **SQLSetPos** is only usable with scroll cursors and, as above, simple-large-object columns are not allowed in scroll cursors. Also according to the standard, **SQLGetData** must not be used in conjunction with a forward-only cursor with a rowset size greater than `1`.

  The alternative to using **SQLGetData** is to use **SQLBindCol**, which would come before the call to **SQLFetch**.

You might want to optimize use of SQL_ATTR_ROW_ARRAY_SIZE so the application sets the value of it according to the maximum number of rows that can be transported in a single buffer. After a statement is prepared, the application might call **SQLGetStmtAttr** to get the value of SQL_INFX_ATTR_FET_ARR_SIZE. If the data fits in one fetch buffer, the internal setting of SQL_INFX_ATTR_FET_ARR_SIZE equals the application setting of SQL_ATTR_ROW_ARRAY_SIZE. In practice, this is only useful on large result sets.

Using the SPL Output Parameter Feature

IBM Informix ODBC Driver supports the ODBC defined method of getting the return value from a database procedure. Specifically, ODBC supports the parameter to the left of the equals sign in a procedure-call escape sequence. The host variable associated with that parameter is updated upon statement execution either using **SQLExecute** or **SQLExecDirect**.

In the IBM Informix ODBC Driver definition of a procedure-call escape sequence, there is only one return value; therefore, the following restrictions are placed on this feature:

- Procedures used with this feature must return only one value, although they might return multiple rows.

  If this condition is not met, the parameter and its binding are ignored.
- Data from the first row only will be placed in the host variable associated with the bound parameter, although procedures used with this feature can return multiple rows.

To return multiple-value, multiple-row result sets from an Informix database server, you have to fetch the data as though it were the result columns of a select statement. This output parameter feature works with existing applications that bind column(s) and call **SQLFetch** or call **SQLFetch** and **SQLGetData** when accessing data through a procedure call. Therefore, no error or warning is generated when more than one row is available to be returned.

You can use either or both methods for retrieving the data from a stored procedure. A host variable can be bound as a parameter or as a column, or both. If separate buffers are used, only the host variable bound as a parameter is updated upon statement execution, and only the host variable bound as a column is updated upon a fetch. Unbound columns accessed through **SQLGetData** remain unaffected.

Using Asynchronous Execution

Design your application to take advantage of data sources that support asynchronous execution. Asynchronous calls do not perform faster, but well-designed applications appear to run more efficiently.

Turning on asynchronous execution does not by itself improve performance. Well-designed applications, however, can take advantage of asynchronous query execution by allowing the user to work on other things while the query is being evaluated on the database server. Perhaps users will start one or more subsequent queries or choose to work in another application, all while the query is executing on the database server. Designing for asynchronous execution makes your application appear to run faster by allowing the user to work concurrently on multiple tasks.

By default, an application makes calls to an ODBC driver that then executes statements against the database server in a synchronous manner. In this mode of operation, the driver does not return control to the application until its own request to the database server is complete. For statements that take more than a few seconds to complete execution, this control return delay can result in the perception of poor performance.

Some data sources support asynchronous execution. When in asynchronous mode, an application makes calls to an ODBC driver and control is returned almost immediately. In this mode the driver returns the status SQL_STILL_EXECUTING to the application and then sends the appropriate request to the database server for execution. The application polls the driver at various intervals at which point the driver itself polls the database server to see if the query has completed execution. If the query is still executing, then the status SQL_STILL_EXECUTING is returned to the application. If it has completed, then a status such as SQL_SUCCESS is returned, and the application can then begin to fetch records.

Updating Data with Positioned Updates and Deletes

Although positioned updates do not apply to all types of applications, try to use positioned updates and deletes whenever possible. Positioned updates (with UPDATE WHERE CURRENT OF CURSOR) allow you to update data by positioning the database cursor to the row to be changed and signaling the driver to change the data. You are not forced to build a complex SQL statement; you simply supply the data to be changed.

Besides making the code more maintainable, positioned updates typically result in improved performance. Because the database server is already positioned on the row (for the SELECT statement currently in process), expensive operations to locate the row to be changed are unnecessary. If the row must be located, the database server typically has an internal pointer to the row available (for example, ROWID).

To support positioned UPDATE and DELETE statements with scrollable cursors, IBM Informix ODBC Driver constructs a new searched UPDATE or DELETE statement from the original positioned statement. However, the database server cannot update scroll cursors directly. Instead, IBM Informix ODBC Driver constructs a WHERE clause that references each column fetched in the SELECT statement referenced in the WHERE CURRENT OF CURSOR clause. Values from the rowset data cache of the SELECT statement are bound to each value in the constructed WHERE clause.

This method of positioning is both slower and perhaps more error prone than using a WHERE CURRENT OF CURSOR clause with FORWARD ONLY cursors. If the fetched rows do not contain a unique key value, the constructed WHERE clause might identify one or many rows, causing many rows to be deleted or updated. Deletion of rows in this manner affects both positioned UPDATE and DELETE statements, and **SQLSetPos** statements when you use scroll cursors.

Use **SQLSpecialColumns** to determine the optimal set of columns to use in the WHERE clause for updating data. Many times pseudocolumns provide the fastest access to the data; you can determine these columns only by using **SQLSpecialColumns**.

Many applications cannot be designed to take advantage of positioned updates and deletes. These applications typically update data by forming a WHERE clause that consists of some subset of the column values that are returned in the result set. Some applications might formulate the WHERE clause by using all searchable result columns or by calling **SQLStatistics** to find columns that might be part of a unique index. These methods typically work but can result in fairly complex queries.

Consider the following example:

```
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
   address, city, state, zip FROM emp", SQL_NTS);
// fetchdata
.
.
.
rc = SQLExecDirect (hstmt, "UPDATE EMP SET ADDRESS = ?
   WHERE first_name = ? AND last_name = ? AND ssn = ? AND
   address = ? AND city = ? AND state = ? AND zip = ?", SQL_NTS);
// fairly complex query
```

Applications should call **SQLSpecialColumns**/SQL_BEST_ROWID to retrieve the optimal set of columns (possibly a pseudocolumn) that identifies any given record. Many databases support special columns that are not explicitly user-defined in the table definition but are *hidden* columns of every table (for example, ROWID, TID, and so on). These pseudocolumns almost always provide the fastest access to the data because they typically are pointers to the exact location of the record. Because pseudocolumns are not part of the explicit table definition, they are not returned from **SQLSpecialColumns**. The only way to determine whether pseudocolumns exist is to call **SQLSpecialColumns**.

Consider the previous example, this time using **SQLSpecialColumns**:

```
.
.
.
rc = SQLSpecialColumns (hstmt, ..... 'emp', ...);
.
.
.
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
   address, city, state, zip, ROWID FROM emp", SQL_NTS);
// fetch data and probably "hide" ROWID from the user
.
.
.
rc = SQLExecDirect (hstmt, "UPDATE emp SET address = ? WHERE
   ROWID = ?", SQL_NTS);
// fastest access to the data!
```

If your data source does not contain special pseudocolumns, the result set of **SQLSpecialColumns** consists of the columns of the optimal unique index on the specified table (if a unique index exists). Therefore, your application need not additionally call **SQLStatistics** to find the smallest unique index.

Message Transfer Optimization

If you activate the message transfer optimization feature (OPTMSG), the driver minimizes message transfers with the database server for most IBM Informix ODBC functions. In addition, the driver chains messages from the database server together and eliminates some small message packets to accomplish optimized message transfers.

To activate message transfer optimization, set the SQL_INFX_ATTR_OPTMSG statement attribute to one (1). The optimization default is: OFF.

Message Chaining Restrictions

IBM Informix ODBC does not chain the following SQL functions even when you enable message transfer optimization:

- **SQLDisconnect**
- **SQLConnect**
- **SQLEndTran**
- **SQLExecute** (if the driver returns results using the select or call procedure and when the driver uses insert cursors to perform a bulk insert)
- **SQLExtendedFetch**
- **SQLFetch**
- **SQLFetchScroll**
- **SQLPrepare**

When the driver reaches one of the functions listed previously, it performs the following actions:

1. Flushes the message queue to the database server only when it encounters SQL statements that require a response from the database server.

   The driver does not flush the message queue when it encounters functions that do not require network traffic, such as **SQLAllocStmt**.
2. Continues message chaining for subsequent SQL statements.

Disabling Message Chaining

You can choose to disable message chaining. Before you disable message chaining, consider the following situations:

- Some SQL statements require immediate replies. If you disable message chaining, re-enable the OPTMSG feature once the restricted SQL statement completes.
- If you perform debugging, you can disable the OPTMSG feature when you are trying to determine how each SQL statement responds.
- If you enable OPTMSG, the message is queued up for the database server but it is not sent for processing. Consider disabling message chaining before the last SQL statement in the program to ensure that the database server processes all messages before the application exits.
- If you disable message chaining, you must reset the SQL_INFX_ATTR_OPTMSG attribute immediately after the SQL statement that requires it to avoid unintended chaining.

  The following example shows how to disable message chaining by placing the SQL_INFX_ATTR_OPTMSG attribute after the DELETE statement. If you place the attribute after the delete statement, the driver can flush all the queued messages when the next SQL statement executes.

```
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 1);
SQLExecDirect(hstmt, (unsigned char *)
"delete from customer", SQL_NTS);
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 0);
SQLExecDirect(hstmt, (unsigned char *)
"create index ix1 on customer (zipcode)", SQL_NTS);
```

  Unintended message chaining can make it difficult to determine which of the chained statements failed.

  At the CREATE INDEX statement, the driver sends both the DELETE and the CREATE INDEX statements to the database server.

Handling Errors with Optimized Message Transfers

When you enable the OPTMSG feature, IBM Informix ODBC does not perform error handling on any chained statement. If you are not sure whether a particular statement might generate an error, include error-handling statements in your code and do not enable message chaining for that statement.

The database server stops execution of subsequent statements when an error occurs in a chained statement. For example, in the following code fragment, the intent is to chain five INSERT statements:

```
SQLExecDirect(hstmt, "create table tab1 (col1 INTEGER)", SQL_NTS);
/* enable message chaining */
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 1);
/* these two INSERT statements execute successfully */
SQLExecDirect(hstmt, "insert into tab1 values (1)", SQL_NTS);
SQLExecDirect(hstmt, "insert into tab1 values (2)", SQL_NTS);
/* this INSERT statement generates an error because the data
* in the VALUES clause is not compatible with the column type */
SQLExecDirect(hstmt, "insert into tab1 values ('a')", SQL_NTS);
/* these two INSERT statements never execute */
SQLExecDirect(hstmt, "insert into tab1 values (3)", SQL_NTS);
SQLExecDirect(hstmt, "insert into tab1 values (4)", SQL_NTS);
/* disable message chaining */
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 0);
/* commit work */
rc = SQLEndTran (SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
if (rc != SQL_SUCCESS)
```

In this example, the following actions occur:

- The driver sends the five INSERT statements and the COMMIT WORK statements to the database server for execution.
- The database inserts col1 values of 1 and 2 into the tab1 table.
- The third INSERT statement generates an error, so the database server does not execute the subsequent INSERT statements or the COMMIT WORK statement.
- The driver flushes the message queue when the queue reaches the **SQLEndTran** function.
- The **SQLEndTran** function, which is the last statement in the chained statements, returns the error from the failed INSERT statement.

If you want to keep the values that the database server inserted into col1, you must commit them yourself.

Chapter

8

Error Messages

In This Chapter

This chapter describes the IBM Informix ODBC Driver, Version 3.82, error messages. The chapter provides information on:

- Diagnostic SQLSTATE values
- SQLSTATE values mapped to Informix error messages
- IBM Informix ODBC Driver error messages mapped to specific SQLSTATE values

For a detailed description of an error message, see **finderr** or *IBM Informix Error Messages* on the IBM Informix Online Documentation site at http://www.ibm.com/software/data/informix/pubs/library/.

Diagnostic SQLSTATE Values

Each IBM Informix ODBC Driver function can return an SQLSTATE value that corresponds to an Informix error code. A function can return additional SQLSTATE values that arise from implementation-specific situations. **SQLError** returns SQLSTATE values as defined by the X/Open and SQL Access Group SQL CAE specification (1992).

SQLSTATE values are character strings that consist of a two-character class value followed by a three-character subclass value. A class value of `01` indicates a warning and is accompanied by a return code of SQL_SUCCESS_WITH_INFO. Class values other than `01`, except for the class `IM`, indicate an error and are accompanied by a return code of SQL_ERROR. The class `IM` signifies warnings and errors that derive from the implementation of IBM Informix ODBC Driver. The subclass value `000` in any class is for implementation-defined conditions within the given class. ANSI SQL-92 defines the assignment of class and subclass values.

Mapping SQLSTATE Values to Informix Error Messages

The following table maps SQLSTATE values that IBM Informix ODBC Driver can return.

A return value of SQL_SUCCESS normally indicates a function has executed successfully, although the SQLSTATE 00000 also indicates success.

SQLSTATE	Error Message	Can be returned from
01000	General warning	All IBM Informix ODBC Driver functions except: **SQLAllocEnv** **SQLError**
01002	Disconnect error	**SQLDisconnect**
01004	Data truncated	**SQLBrowseConnect** **SQLColAttributes** **SQLDataSources** **SQLDescribeCol** **SQLDriverConnect** **SQLDrivers** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLGetCursorName** **SQLGetData** **SQLGetInfo** **SQLNativeSql** **SQLPutData** **SQLSetPos**
01006	Privilege not revoked	**SQLExecDirect** **SQLExecute**
01S00	Invalid connection string attribute	**SQLBrowseConnect** **SQLDriverConnect**
01S01	Error in row	**SQLExtendedFetch** **SQLSetPos**

(1 of 15)

SQLSTATE	Error Message	Can be returned from
01S02	Option value changed	**SQLSetConnectOption** **SQLSetStmtOption**
01S03	No rows updated or deleted	**SQLExecDirect** **SQLExecute** **SQLSetPos**
01S04	More than one row updated or deleted	**SQLExecDirect** **SQLExecute** **SQLSetPos**
07001	Wrong number of parameters	**SQLExecDirect** **SQLExecute**
07006	Restricted data type attribute violation	**SQLBindParameter** **SQLExtendedFetch** **SQLFetch** **SQLGetData**
08001	Unable to connect to data source	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
08002	Connection in use	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect** **SQLSetConnectOption**
08003	Connection not open	**SQLAllocStmt** **SQLDisconnect** **SQLGetConnectOption** **SQLGetInfo** **SQLNativeSql** **SQLSetConnectOption** **SQLTransact**
08004	Data source rejected establishment of connection	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
08007	Connection failure during transaction	**SQLTransact**

SQLSTATE	Error Message	Can be returned from
08S01	Communication link failure	**SQLBrowseConnect** **SQLColumnPrivileges** **SQLColumns** **SQLConnect** **SQLDriverConnect** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLFreeConnect** **SQLGetData** **SQLGetTypeInfo** **SQLParamData** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLPutData** **SQLSetConnectOption** **SQLSetStmtOption** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables**
21S01	Insert value list does not match column list	**SQLExecDirect** **SQLPrepare**
21S02	Degree of derived table does not match column list	**SQLExecDirect** **SQLPrepare** **SQLSetPos**
22001	String data right truncation	**SQLPutData**
22003	Numeric value out of range	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLGetData** **SQLGetInfo** **SQLPutData** **SQLSetPos**

SQLSTATE	Error Message	Can be returned from
22005	Error in assignment	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLGetData** **SQLPrepare** **SQLPutData** **SQLSetPos**
22008	Datetime field overflow	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLGetData** **SQLPutData** **SQLSetPos**
22012	Division by zero	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLGetData**
22026	String data, length mismatch	**SQLParamData**
23000	Integrity constraint violation	**SQLExecDirect** **SQLExecute** **SQLSetPos**

SQLSTATE	Error Message	Can be returned from
24000	Invalid cursor state	**SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLDescribeCol** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLGetData** **SQLGetStmtOption** **SQLGetTypeInfo** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLSetCursorName** **SQLSetPos** **SQLSetStmtOption** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables**
25000	Invalid transaction state	**SQLDisconnect**
28000	Invalid authorization specification	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
34000	Invalid cursor name	**SQLExecDirect** **SQLPrepare** **SQLSetCursorName**
37000	Syntax error or access violation	**SQLExecDirect** **SQLNativeSql** **SQLPrepare**
3C000	Duplicate cursor name	**SQLSetCursorName**

SQLSTATE	Error Message	Can be returned from
40001	Serialization failure	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch**
42000	Syntax error or access violation	**SQLExecDirect** **SQLExecute** **SQLPrepare** **SQLSetPos**
70100	Operation aborted	**SQLCancel**
IM001	Driver does not support this function	All ODBC functions except: **SQLAllocConnect** **SQLAllocEnv** **SQLDataSources** **SQLDrivers** **SQLError** **SQLFreeConnect** **SQLFreeEnv** **SQLGetFunctions**
IM002	Data source name not found and no default driver specified	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
IM003	Specified driver could not be loaded	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
IM004	Driver **SQLAllocEnv** failed	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
IM005	Driver **SQLAllocConnect** failed	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
IM006	Driver **SQLSetConnectOption** failed	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect**
IM007	No data source or driver specified; dialog prohibited	**SQLDriverConnect**

SQLSTATE	Error Message	Can be returned from
IM008	Dialog failed	**SQLDriverConnect**
IM009	Unable to load translation shared library	**SQLBrowseConnect** **SQLConnect** **SQLDriverConnect** **SQLSetConnectOption**
IM010	Data source name too long	**SQLBrowseConnect** **SQLDriverConnect**
IM011	Driver name too long	**SQLBrowseConnect** **SQLDriverConnect**
IM012	DRIVER keyword syntax error	**SQLBrowseConnect** **SQLDriverConnect**
IM013	Trace file error	All ODBC functions.
S0001	Base table or view already exists	**SQLExecDirect** **SQLPrepare**
S0002	Base table not found	**SQLExecDirect** **SQLPrepare**
S0011	Index already exists	**SQLExecDirect** **SQLPrepare**
S0012	Index not found	**SQLExecDirect** **SQLPrepare**
S0021	Column already exists	**SQLExecDirect** **SQLPrepare**
S0022	Column not found	**SQLExecDirect** **SQLPrepare**
S0023	No default for column	**SQLSetPos**
S1000	General error	All ODBC functions except: **SQLAllocEnv** **SQLError**

SQLSTATE	Error Message	Can be returned from
S1001	Memory allocation failure	All ODBC functions except: **SQLAllocEnv** **SQLError** **SQLFreeConnect** **SQLFreeEnv**
S1002	Invalid column number	**SQLBindCol** **SQLColAttributes** **SQLDescribeCol** **SQLExtendedFetch** **SQLFetch** **SQLGetData**
S1003	Program type out of range	**SQLBindCol** **SQLBindParameter** **SQLGetData**
S1004	SQL data type out of range	**SQLBindParameter** **SQLGetTypeInfo**

SQLSTATE	Error Message	Can be returned from
S1008	Operation canceled	All ODBC functions that can be processed asynchronously: **SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLDescribeCol** **SQLDescribeParam** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLGetData** **SQLGetTypeInfo** **SQLMoreResults** **SQLNumParams** **SQLNumResultCols** **SQLParamData** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLPutData** **SQLSetPos** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables**

SQLSTATE	Error Message	Can be returned from
S1009	Invalid argument value	**SQLAllocConnect** **SQLAllocStmt** **SQLBindCol** **SQLBindParameter** **SQLExecDirect** **SQLForeignKeys** **SQLGetData** **SQLGetInfo** **SQLNativeSql** **SQLPrepare** **SQLPutData** **SQLSetConnectOption** **SQLSetCursorName** **SQLSetPos** **SQLSetStmtOption**

(10 of 15)

SQLSTATE	Error Message	Can be returned from
S1010	Function sequence error	**SQLBindCol** **SQLBindParameter** **SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLDescribeCol** **SQLDisconnect** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLFreeConnect** **SQLFreeEnv** **SQLFreeStmt** **SQLGetConnectOption** **SQLGetCursorName** **SQLGetData** **SQLGetFunctions** **SQLGetStmtOption** **SQLGetTypeInfo** **SQLMoreResults** **SQLNumParams** **SQLNumResultCols** **SQLParamData** **SQLParamOptions** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLPutData** **SQLRowCount** **SQLSetConnectOption** **SQLSetCursorName** **SQLSetPos** **SQLSetScrollOptions** **SQLSetStmtOption** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables** **SQLTransact**

SQLSTATE	Error Message	Can be returned from
S1011	Operation invalid at this time	**SQLGetStmtOption** **SQLSetConnectOption** **SQLSetStmtOption**
S1012	Invalid transaction operation code specified	**SQLTransact**
S1015	No cursor name available	**SQLGetCursorName**
S1090	Invalid string or buffer length	**SQLBindCol** **SQLBindParameter** **SQLBrowseConnect** **SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLConnect** **SQLDataSources** **SQLDescribeCol** **SQLDriverConnect** **SQLDrivers** **SQLExecDirect** **SQLExecute** **SQLForeignKeys** **SQLGetCursorName** **SQLGetData** **SQLGetInfo** **SQLNativeSql** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLPutData** **SQLSetCursorName** **SQLSetPos** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables**
S1091	Descriptor type out of range	**SQLColAttributes**

SQLSTATE	Error Message	Can be returned from
S1092	Option type out of range	**SQLFreeStmt** **SQLGetConnectOption** **SQLGetStmtOption** **SQLSetConnectOption** **SQLSetStmtOption**
S1093	Invalid parameter number	**SQLBindParameter**
S1094	Invalid scale value	**SQLBindParameter**
S1095	Function type out of range	**SQLGetFunctions**
S1096	Information type out of range	**SQLGetInfo**
S1097	Column type out of range	**SQLSpecialColumns**
S1098	Scope type out of range	**SQLSpecialColumns**
S1099	Nullable type out of range	**SQLSpecialColumns**
S1100	Uniqueness option type out of range	**SQLStatistics**
S1101	Accuracy option type out of range	**SQLStatistics**
S1103	Direction option out of range	**SQLDataSources** **SQLDrivers**
S1104	Invalid precision value	**SQLBindParameter**
S1105	Invalid parameter type	**SQLBindParameter**
S1106	Fetch type out of range	**SQLExtendedFetch**
S1107	Row value out of range	**SQLExtendedFetch** **SQLParamOptions** **SQLSetPos** **SQLSetScrollOptions**
S1108	Concurrency option out of range	**SQLSetScrollOptions**
S1109	Invalid cursor position	**SQLExecute** **SQLExecDirect** **SQLGetData** **SQLGetStmtOption** **SQLSetPos**

SQLSTATE	Error Message	Can be returned from
S1110	Invalid driver completion	**SQLDriverConnect**
S1111	Invalid bookmark value	**SQLExtendedFetch**
S1C00	Driver not capable	**SQLBindCol** **SQLBindParameter** **SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLGetConnectOption** **SQLGetData** **SQLGetInfo** **SQLGetStmtOption** **SQLGetTypeInfo** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLSetConnectOption** **SQLSetPos** **SQLSetScrollOptions** **SQLSetStmtOption** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables** **SQLTransact**

(14 of 15)

SQLSTATE	Error Message	Can be returned from
S1T00	Time-out expired	**SQLBrowseConnect** **SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLConnect** **SQLDescribeCol** **SQLDriverConnect** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLGetData** **SQLGetInfo** **SQLGetTypeInfo** **SQLMoreResults** **SQLNumParams** **SQLNumResultCols** **SQLParamData** **SQLPrepare** **SQLPrimaryKeys** **SQLProcedureColumns** **SQLProcedures** **SQLPutData** **SQLSetPos** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables**

(15 of 15)

Mapping Informix Error Messages to SQLSTATE Values

The rest of this chapter describes diagnostic SQLSTATE values for IBM Informix ODBC Driver functions. The return code for each SQLSTATE value is SQL_ERROR unless a description indicates otherwise. When a function returns SQL_SUCCESS_WITH_INFO or SQL_ERROR, you can call **SQLError** to get the SQLSTATE value.

Core

SQLAllocConnect

The following table describes the SQLSTATE and error values for **SQLAllocConnect**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value

Core

SQLAllocEnv

SQLAllocEnv allocates memory for an environment handle and initializes the driver call level interface for application use. An application must call **SQLAllocEnv** before it calls any other driver function.

A driver cannot return SQLSTATE values directly after the call to **SQLAllocEnv** because no valid handle exists with which to call **SQLError**.

Two levels of **SQLAllocEnv** functions exist, one within the driver manager (if you are using one) and one within the driver. The driver manager does not call the driver-level function until the application calls **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect**. If an error occurs in the driver-level **SQLAllocEnv** function, the driver manager-level **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect** function returns SQL_ERROR. A subsequent call to **SQLError** with *henv*, SQL_NULL_HDBC, and SQL_NULL_HSTMT returns SQLSTATE IM004 (the driver **SQLAllocEnv** function failed), followed by one of the following errors from the driver:

- SQLSTATE S1000 (General error)
- An IBM Informix ODBC Driver SQLSTATE value, which ranges from `S1000` to `S19ZZ`.

  For example, SQLSTATE S1001 (Memory-allocation failure) indicates that the call from the driver manager to the driver-level **SQLAllocEnv** returned SQL_ERROR, and the *henv* from the driver manager was set to SQL_NULL_HENV.

Core

SQLAllocStmt

SQLAllocStmt allocates memory for a statement handle and associates the statement handle with the connection that *hdbc* specifies.

An application must call **SQLAllocStmt** before it submits SQL statements.

The following table describes the SQLSTATE and error values for **SQLAllocStmt**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
08S01	-11301	A protocol error has been detected. Current connection is closed.

Core

SQLBindCol

SQLBindCol assigns the storage and Informix ODBC Driver C data type for a column in a result set, as follows:

- A storage buffer that receives the contents of a column of data
- The length of the storage buffer
- A storage location that receives the actual length of the column of data returned by the fetch operation
- Data type conversion from the Informix SQL data type to the Informix ODBC Driver C data type

The following table describes the SQLSTATE and error values for **SQLBindCol**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1003	-11063	Program type out of range
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable

Important: *An application can call **SQLBindCol** to bind a column to a new storage location, regardless of whether data has already been fetched. The new binding replaces the old binding for bookmark columns as well as other bound columns. The new binding does not apply to data already fetched; it takes effect the next time **SQLFetch**, **SQLExtendedFetch,** or **SQLSetPos** is called.*

Level 1

SQLBindParameter

SQLBindParameter binds a buffer to a parameter marker in an SQL statement.

The following table describes the SQLSTATE and error values for **SQLBindParameter**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
07006	-11013	Restricted data type attribute violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1003	-11063	Program type out of range
S1004	-11064	SQL data type out of range
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1093	-11074	Invalid parameter number
S1094	-11075	Invalid scale value
S1104	-11084	Invalid precision value
S1105	-11085	Invalid parameter type
S1C00	-11092	Driver not capable

Level 2

SQLBrowseConnect

SQLBrowseConnect supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source. Each call to **SQLBrowseConnect** returns successive levels of attributes and attribute values. When all levels are enumerated, a connection to the data source is completed, and **SQLBrowseConnect** returns a complete connection string. A return code of SQL_SUCCESS_WITH_INFO or SQL_SUCCESS indicates that all connection information was specified, and the application is now connected to the data source.

The following table describes the SQLSTATE and error values for the function.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
01S00	-11005	Invalid connection string attribute
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver **SQLAllocEnv** failed
IM005	-11044	Driver **SQLAllocConnect** failed
IM006	-11045	Driver **SQLSetConnectOption** failed
IM009	-11048	Unable to load translation-shared library
IM010	-11049	Data-source name too long
IM011	-11050	Driver name too long

(1 of 2)

SQLSTATE	Error Value	Error Message
IM012	-11051	DRIVER keyword syntax error
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11303	Input connection string too large
S1000	-11317	Invalid CONNECTDATABASE value specified
S1000	-11318	Invalid VMBCHARLENEXACT value specified
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

Core

SQLCancel

SQLCancel cancels the processing on an *hstmt* or a query.

The following table describes the SQLSTATE and error values for the function.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01S05	-11010	Cancel treated as FreeStmt/Close.
70100	-11039	Operation aborted
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
08S01	-11301	A protocol error has been detected. Current connection is closed.

Core

SQLColAttributes

SQLColAttributes returns descriptor information for a column in a result set; it cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

The following table describes the SQLSTATE and error values for **SQLColAttributes**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1091	-11072	Descriptor type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

SQLColAttributes can return any SQLSTATE that can be returned by **SQLPrepare** or **SQLExecute** when it is called after **SQLPrepare** and before **SQLExecute,** depending on when the data source evaluates the SQL statement associated with the *hstmt*.

Level 2

SQLColumnPrivileges

SQLColumnPrivileges returns a list of columns and associated privileges for the specified table. The driver returns the information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for **SQLColumnPrivileges**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 1

SQLColumns

SQLColumns returns the list of column names in specified tables. The driver returns this information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for **SQLColumns**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Core

SQLConnect

SQLConnect loads a driver and establishes a connection to a data source. The connection handle references where all information about the connection, including status, transaction state, and error information is stored.

The following table describes the SQLSTATE and error values for **SQLConnect**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver **SQLAllocEnv** failed
IM005	-11044	Driver **SQLAllocConnect** failed
IM006	-11045	Driver **SQLSetConnectOption** failed
IM009	-11048	Unable to load translation shared library
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired

(1 of 2)

SQLSTATE	Error Value	Error Message
S1000	-11302	Insufficient connection information was supplied
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

Level 2

SQLDataSources

SQLDataSources lists data-source names.

The following table describes the SQLSTATE and error values for **SQLDataSources**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1103	-11083	Direction option out of range

Core

SQLDescribeCol

SQLDescribeCol returns the result descriptor (column name, type, precision, scale, and whether or not it can have a NULL value) for one column in the result set; it cannot be used to return information about the bookmark column (column 0).

The following table describes the SQLSTATE and error values for **SQLDescribeCol**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired

SQLDescribeCol can return any SQLSTATE that **SQLPrepare** or **SQLExecute** returns when **SQLDescribeCol** is called after **SQLPrepare** and before **SQLExecute**, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

SQLDisconnect

SQLDisconnect closes the connection associated with a specific connection handle.

The following table describes the SQLSTATE and error values for **SQLDisconnect**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01002	-11002	Disconnect error
08003	-11017	Connection not open
25000	-11032	Invalid transaction state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
08S01	-11301	A protocol error has been detected. Current connection is closed.

Usage

If an application calls **SQLDisconnect** after **SQLBrowseConnect** returns SQL_NEED_DATA and before it returns a different return code, the driver cancels the connection-browsing process and returns the *hdbc* to an unconnected state.

If an application calls **SQLDisconnect** while an incomplete transaction is associated with the connection handle, the driver returns SQLSTATE 25000 (Invalid transaction state), indicating that the transaction is unchanged and the connection is open. An incomplete transaction is one that was not committed or rolled back with **SQLTransact**.

If an application calls **SQLDisconnect** before it frees every *hstmt* associated with the connection, the driver frees each remaining *hstmt* after it successfully disconnects from the data source. However, if one or more of the *hstmts* associated with the connection are still executing asynchronously, **SQLDisconnect** returns SQL_ERROR with an SQLSTATE value of S1010 (Function sequence error).

Level 1

SQLDriverConnect

SQLDriverConnect is an alternative to **SQLConnect**. It supports data sources that require more connection information than the three arguments in **SQLConnect** dialog boxes to prompt the user for all connection information and data sources that are not defined data-source names.

SQLDriverConnect provides the following connection options:

- You can establish a connection using a connection string that contains the data-source name, one or more user IDs, one or more passwords, and other information that the data source requires.
- You can establish a connection using a partial connection string or no additional information; in this case, IBM Informix ODBC Driver can prompt the user for connection information.

Once a connection is established, **SQLDriverConnect** returns the completed connection string. The application can use this string for subsequent connection requests.

The following table describes the SQLSTATE and error values for **SQLDriverConnect**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
01S00	-11005	Invalid connection string attribute
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded

(1 of 2)

SQLSTATE	Error Value	Error Message
IM004	-11043	Driver **SQLAllocEnv** failed
IM005	-11044	Driver **SQLAllocConnect** failed
IM006	-11045	Driver **SQLSetConnectOption** failed
IM007	-11046	No data source or driver specified; dialog prohibited
IM008	-11047	Dialog failed
IM009	-11048	Unable to load translation shared library
IM010	-11049	Data-source name too long
IM011	-11050	Driver name too long
IM012	-11051	DRIVER keyword syntax error
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1110	-11090	Invalid driver completion
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11302	Insufficient connection information was supplied
S1000	-11303	Input connection string too large
S1000	-11317	Invalid CONNECTDATABASE value specified
S1000	-11318	Invalid VMBCHARLENEXACT value specified
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

Level 2

SQLDrivers

SQLDrivers lists driver descriptions and driver-attribute keywords.

The following table describes the SQLSTATE and error values for **SQLDrivers**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1103	-11083	Direction option out of range

Core

SQLError

SQLError returns error or status information.

SQLError does not post error values for itself. **SQLError** returns SQL_NO_DATA_FOUND when it cannot retrieve any error information (in which case *sqlstate* equals 00000). If **SQLError** cannot access error values for any reason that would normally return SQL_ERROR, **SQLError** returns SQL_ERROR but does not post any error values. If the buffer for the error message is too short, **SQLError** returns SQL_SUCCESS_WITH_INFO but still does not return an SQLSTATE value for **SQLError**.

To determine that a truncation occurred in the error message, an application can compare *cbErrorMsgMax* to the actual length of the message text written to *pcbErrorMsg*.

Core

SQLExecDirect

SQLExecDirect executes a preparable statement, using the current values of the parameter-marker variables if any parameters exist in the statement. **SQLExecDirect** is the fastest way to submit an SQL statement for one-time execution.

The following table describes the SQLSTATE and error values for **SQLExecDirect**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
01006	-11004	Privilege not revoked
01S03	-11008	No rows updated or deleted
01S04	-11009	More than one row updated or deleted
07001	-11012	Wrong number of parameters
07S01	-11014	Invalid use of default parameter
08S01	-11020	Communication-link failure
21S01	-11021	Insert value list does not match column list
21S02	-11022	Degree of derived table does not match column list
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
23000	-11030	Integrity-constraint violation
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name

(1 of 3)

SQLSTATE	Error Value	Error Message
37000	-11035	Syntax error or access violation
40001	-11037	Serialization failure
42000	-11038	Syntax error or access violation
S0001	-11053	Base table or view already exists
S0002	-11054	Table or view not found
S0011	-11055	Index already exists
S0012	-11056	Index not found
S0021	-11057	Column already exists
S0022	-11058	Column not found
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.

(2 of 3)

SQLSTATE	Error Value	Error Message
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(3 of 3)

Core

SQLExecute

SQLExecute executes a prepared statement, using the current values of the parameter-marker variables if any parameter markers exist in the statement.

The following table describes the SQLSTATE and error values for **SQLExecute**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
01006	-11004	Privilege not revoked
01S03	-11008	No rows updated or deleted
01S04	-11009	More than one row updated or deleted
07001	-11012	Wrong number of parameters
07S01	-11014	Invalid use of default parameter.
08S01	-11020	Communication-link failure
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
23000	-11030	Integrity constraint violation
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
42000	-11038	Syntax error or access violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled

(1 of 2)

SQLSTATE	Error Value	Error Message
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

SQLExecute can return any SQLSTATE that **SQLPrepare** can return based on when the data source evaluates the SQL statement associated with the *hstmt*.

Level 2

SQLExtendedFetch

SQLExtendedFetch extends the functionality of **SQLFetch** in the following ways:

- It returns row-set data (one or more rows), in the form of an array, for each bound column.
- It scrolls through the result set according to the setting of a scroll-type argument.

SQLExtendedFetch works with **SQLSetStmtOption**.

To fetch one row of data at a time in a forward direction, an application should call **SQLFetch**.

The following table describes the SQLSTATE and error values for **SQLExtendedFetch**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
01S01	-11006	Error in row
07006	-11013	Restricted data type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure

(1 of 2)

SQLSTATE	Error Value	Error Message
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1106	-11086	Fetch type out of range
S1107	-11087	Row value out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11307	In **SQLExtendedFetch**, only SQL_FETCH_NEXT is supported for SQL_SCROLL_FORWARD_ONLY cursors

(2 of 2)

If an error occurs that pertains to the entire row set, such as SQLSTATE S1T00 (Time-out expired), the driver returns SQL_ERROR and the appropriate SQLSTATE. The contents of the row set buffers are undefined, and the cursor position is unchanged.

If an error occurs that pertains to a single row, the driver performs the following actions:

- Sets the element in the *rgfRowStatus* array for the row to SQL_ROW_ERROR
- Posts SQLSTATE 01S01 (Error in row) in the error queue
- Posts zero or more additional SQLSTATE values for the error after SQLSTATE 01S01 (Error in row) in the error queue

After the driver processes the error or warning, it continues the operation for the remaining rows in the row set and returns SQL_SUCCESS_WITH_INFO. Thus, for each error that pertains to a single row, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.

After the driver processes the error, it fetches the remaining rows in the row set and returns SQL_SUCCESS_WITH_INFO. Thus, for each row that returns an error, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATE values.

If the row set contains rows that are already fetched, the driver is not required to return SQLSTATE values for errors that occurred when the rows were first fetched. However, it is required to return SQLSTATE 01S01 (Error in row) for each row in which an error originally occurred and to return SQL_SUCCESS_WITH_INFO. For example, a static cursor that maintains a cache might cache row-status information (so that it can determine which rows contain errors) but might not cache the SQLSTATE associated with those errors.

Error rows do not affect relative cursor movements. For example, suppose the result set size is 100, and the row-set size is 10. If the current row set is rows 11 through 20 and the element in the *rgfRowStatus* array for row 11 is SQL_ROW_ERROR, calling **SQLExtendedFetch** with the SQL_FETCH_NEXT fetch type still returns rows 21 through 30.

If the driver returns any warnings, such as SQLSTATE 01004 (Data truncated), it returns warnings that apply to the entire row set or to unknown rows in the row set before it returns error information that applies to specific rows. It returns warnings for specific rows with any other error information about those rows.

Core

SQLFetch

SQLFetch fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with **SQLBindCol**.

The following table describes the SQLSTATE and error values for **SQLFetch**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
07006	-11013	Restricted data-type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

Level 2

SQLForeignKeys

SQLForeignKeys can return either of the following items:

- A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables)
- A list of foreign keys in other tables that refer to the primary key in the specified table

The driver returns each list as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for **SQLForeignKeys**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication link failure
24000	-11031	Invalid cursor state
IM001	-11040	Driver does not support this function
S1000	-11060	General error
S1001	-11061	Memory allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Timeout expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.

(1 of 2)

SQLSTATE	Error Value	Error Message
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

Core

SQLFreeConnect

SQLFreeConnect releases a connection handle and frees all memory associated with the handle.

The following table describes the SQLSTATE and error values for **SQLFreeConnect**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
S1000	-11060	General error
S1010	-11067	Function-sequence error

Core

SQLFreeEnv

SQLFreeEnv frees the environment handle and releases all memory associated with the environment handle.

The following table describes the SQLSTATE and error values for **SQLFreeEnv**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1010	-11067	Function-sequence error

Core

SQLFreeStmt

SQLFreeStmt stops the processing that is associated with a specific *hstmt*, closes any open cursors that are associated with the *hstmt*, discards pending results, and, optionally, frees all resources associated with the statement handle.

The following table describes the SQLSTATE and error values for **SQLFreeStmt**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1092	-11073	Option type out of range

Level 1

SQLGetConnectOption

SQLGetConnectOption returns the current setting of a connection option.

The following table describes the SQLSTATE and error values for **SQLGetConnectOption**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable

Core

SQLGetCursorName

SQLGetCursorName returns the cursor name associated with a specified *hstmt*.

The following table describes the SQLSTATE and error values for **SQLGetCursorName**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1015	-11070	No cursor name available
S1090	-11071	Invalid string or buffer length

Level 1

SQLGetData

SQLGetData returns result data for a single unbound column in the current row. The application must call **SQLFetch** or **SQLExtendedFetch** and (optionally) **SQLSetPos** to position the cursor on a row of data before it calls **SQLGetData**. It is possible to use **SQLBindCol** for some columns and use **SQLGetData** for others within the same row. This function can be used to retrieve character or binary data values in parts from a column with a character, binary, or data source-specific data type (for example, data from SQL_LONGVARBINARY or SQL_LONGVARCHAR columns).

The following table describes the SQLSTATE and error values for **SQLGetData**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
07006	-11013	Restricted data- type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime-field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1003	-11063	Program type out of range
S1008	-11065	Operation canceled

(1 of 2)

SQLSTATE	Error Value	Error Message
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

(2 of 2)

Level 1

SQLGetFunctions

SQLGetFunctions returns information about whether the driver supports a specific function.

The following table describes the SQLSTATE and error values for **SQLGetFunctions**.

SQLSTATE	Error Value	Error Message
01000	-1101	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1095	-11076	Function type out of range

Level 1

SQLGetInfo

SQLGetInfo returns general information about the driver and data source associated with an *hdbc*.

The following table describes the SQLSTATE and error values for **SQLGetInfo**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
08003	-11017	Connection not open
22003	-11025	Numeric value out of range
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1090	-11071	Invalid string or buffer length
S1096	-11077	Information type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

Level 1

SQLGetStmtOption

SQLGetStmtOption returns the current setting of a statement option.

The following table describes the SQLSTATE and error values for **SQLGetStmtOption**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable

Level 1

SQLGetTypeInfo

SQLGetTypeInfo returns information about data types that the data source supports. The driver returns the information in the form of an SQL result set.

The following table describes the SQLSTATE and error values for **SQLGetTypeInfo**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1004	-11063	SQL data type out of range
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11305	SQLGetTypeInfo supported for FORWARD_ONLY cursors
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 2

SQLMoreResults

SQLMoreResults determines whether more results are available on an *hstmt* that contains SELECT, UPDATE, INSERT, or DELETE statements and, if so, initializes processing for those results.

The following table describes the SQLSTATE and error values for **SQLMoreResults**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

Level 2

SQLNativeSql

SQLNativeSql returns the SQL string that the driver translates.

The following table describes the SQLSTATE and error values for **SQLNativeSql**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
08003	-11017	Connection not open
37000	-11035	Syntax error or access violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1090	-11071	Invalid string or buffer length
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Usage

The following example shows what **SQLNativeSql** might return for an input SQL string that contains the scalar function LENGTH:

```
SELECT {fn LENGTH(NAME)} FROM EMPLOYEE
```

IDS

Dynamic Server might return the following translated SQL string:

```
SELECT length(NAME) FROM EMPLOYEE
```

♦

Level 2

SQLNumParams

SQLNumParams returns the number of parameters in an SQL statement.

The following table describes the SQLSTATE and error values for **SQLNumParams**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired

Core

SQLNumResultCols

SQLNumResultCols returns the number of columns in a result set.

The following table describes the SQLSTATE and error values for **SQLNumResultCols**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired

SQLNumResultCols can return any SQLSTATE that **SQLPrepare** or **SQLExecute** can return when **SQLNumResultCols** is called after **SQLPrepare** and before **SQLExecute** is called, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

Level 1

SQLParamData

SQLParamData is used with **SQLPutData** to supply parameter data when a statement executes.

The following table describes the SQLSTATE and error values for **SQLParamData**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
22026	-11029	String data, length mismatch
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.

If **SQLParamData** is called while sending data for a parameter in an SQL statement, it can return any SQLSTATE that can be returned by the function that was called to execute the statement (**SQLExecute** or **SQLExecDirect**). If it is called while sending data for a column being updated or added with **SQLSetPos**, it can return any SQLSTATE that can be returned by **SQLSetPos**.

If **SQLParamData** is called while sending data for a parameter in an SQL statement, it can return any SQLSTATE that can be returned by the function that was called to execute the statement (**SQLExecute** or **SQLExecDirect**).

Core

Level 2

SQLParamOptions

SQLParamOptions allows an application to specify multiple values for the set of parameters assigned by **SQLBindParameter**. The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. For example, an application can specify three sets of values for the set of parameters associated with an INSERT statement, and then execute the INSERT statement once to perform the three insert operations.

The following table lists the SQLSTATE values commonly returned by **SQLParamOptions** and explains each one in the context of this function; the notation "(DM)" precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL_ERROR unless noted otherwise.

SQLSTATE	Error Value	Error Message
01000		General warning
S1000		General error
S1001		Memory-allocation failure
S1010		Function-sequence error
S1107		Row value out of range

SQLPrepare

SQLPrepare prepares an SQL string for execution.

The following table describes the SQLSTATE and error values for **SQLPrepare**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
21S01	-11021	Insert value list does not match column list
21S02	-11022	Degree of derived table does not match column list
22005	-11026	Error in assignment
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
37000	-11035	Syntax error or access violation
42000	-11038	Syntax error or access violation
S0001	-11053	Base table or view already exists
S0002	-11054	Base table not found
S0011	-11055	Index already exists
S0012	-11056	Index not found
S0021	-11057	Column already exists
S0022	-11058	Column not found
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error

(1 of 2)

SQLSTATE	Error Value	Error Message
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

Level 2

SQLPrimaryKeys

SQLPrimaryKeys returns the column names that comprise the primary key for a table. The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.

The following table describes the SQLSTATE and error values for **SQLPrimaryKeys**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 2

SQLProcedureColumns

SQLProcedureColumns returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. The driver returns the information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for **SQLProcedureColumns**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication link failure
24000	-11031	Invalid cursor state
IM001	-11040	Driver does not support this function
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 2

SQLProcedures

SQLProcedures returns the list of procedure names stored in a specific data source. *Procedure* is a generic term used to describe an *executable object*, or a named entity that can be invoked using input and output parameters, and which can return result sets similar to the results that SELECT statements return.

The following table describes the SQLSTATE and error values for **SQLProcedures**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 1

SQLPutData

SQLPutData allows an application to send data for a parameter or column to the driver at statement execution time. This function can send character or binary data values in parts to a column with a character, binary, or data-source-specific data type (for example, parameters of SQL_LONGVARBINARY or SQL_LONGVARCHAR).

The following table describes the SQLSTATE and error values for **SQLPutData**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01004	-11003	Data truncated
07S01	-11014	Invalid use of default parameter
08S01	-11020	Communication-link failure
22001	-11023	String data right truncation
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime-field overflow
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired

Important: *An application can use* ***SQLPutData*** *to send sections of character C data to a column with a character, binary, or data source-specific data type or to send binary C data to a column with a character, binary, or data source-specific data type. If* ***SQLPutData*** *is called more than once under any other conditions, it returns SQL_ERROR and SQLSTATE 22003 (Numeric value out of range).*

Core

SQLRowCount

SQLRowCount returns the number of rows affected by an UPDATE, INSERT, or DELETE statement or by an SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in **SQLSetPos**.

The following table describes the SQLSTATE and error values for **SQLRowCount**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error

Level 1

SQLSetConnectOption

SQLSetConnectOption sets options that govern aspects of connections.

The following table describes the SQLSTATE and error values for **SQLSetConnectOption**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01S02	-11007	Option value changed
08002	-11016	Connection in use
08003	-11017	Connection not open
08S01	-11020	Communication-link failure
IM009	-11048	Unable to load translation shared library
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

When *fOption* is a statement option, **SQLSetConnectOption** can return any SQLSTATE that **SQLSetStmtOption** returns.

Core

SQLSetCursorName

SQLSetCursorName associates a cursor name with an active *hstmt*. If an application does not call **SQLSetCursorName**, the driver generates cursor names as needed for SQL statement processing.

The following table describes the SQLSTATE and error values for **SQLSetCursorName**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
3C000	-11036	Duplicate cursor name
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length

Level 1

SQLSetStmtOption

SQLSetStmtOption sets options that are related to an *hstmt*. To set an option for all the statements associated with a specific *hdbc*, an application can call **SQLSetConnectOption**.

The following table describes the SQLSTATE and error values for **SQLSetStmtOption**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
01S02	-11007	Option value changed
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option value out of range
S1C00	-11092	Driver not capable

Level 1

SQLSpecialColumns

SQLSpecialColumns retrieves the following information about columns within a specified table:

- The optimal set of columns that uniquely identifies a row in the table
- Columns that are automatically updated when any value in the row is updated by a transaction

The following table describes the SQLSTATE and error values for **SQLSpecialColumns**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1097	-11078	Column type out of range
S1098	-11079	Scope type out of range
S1099	-11080	Nullable type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported

(1 of 2)

SQLSTATE	Error Value	Error Message
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

(2 of 2)

Level 1

SQLStatistics

SQLStatistics retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns this information as a result set.

The following table describes the SQLSTATE and error values for **SQLStatistics**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory- allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1100	-11081	Uniqueness option type out of range
S1101	-11082	Accuracy option type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 2

SQLTablePrivileges

SQLTablePrivileges returns a list of tables and the privileges associated with each table. The driver returns the information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for **SQLTablePrivileges**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Level 1

SQLTables

SQLTables returns the list of table names that are stored in a specific data source. The driver returns this information as a result set.

The following table describes the SQLSTATE and error values for **SQLTables**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contains an escape clause not supported by this database driver

Core

SQLTransact

SQLTransact requests a commit or rollback operation for all active operations on all *hstmts* associated with a connection. **SQLTransact** can also request that a commit or rollback operation be performed for all connections associated with the *henv*.

The following table describes the SQLSTATE and error values for **SQLTransact**.

SQLSTATE	Error Value	Error Message
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1012	-11069	Invalid transaction operation code
S1C00	-11092	Driver not capable
08S01	-11301	A protocol error has been detected. Current connection is closed.

Chapter

9

Unicode

In This Chapter

This chapter provides a brief overview of the Unicode standard and shows how it is used within ODBC applications.

Overview of Unicode

Unicode is a character encoding standard that provides a means of representing each character used in every major language. In the Unicode standard, each character is assigned a unique numeric value and name. These values can be used consistently between applications across multiple platforms.

Unicode versions

Although Unicode provides a consistent way of representing text across multiple languages, there are different versions which provide different data sizes for each character. The following table describes the versions that are supported within an IBM Informix ODBC application.

UCS-2 ISO encoding standard that maps Unicode characters to 2 bytes each. UCS-2 is the common encoding standard on Windows.

IBM Informix ODBC Driver for IBM AIX platforms supports UCS-2 encoding. IBM Informix ODBC Driver for Windows supports only UCS-2.

UCS-4 ISO encoding standard that maps Unicode characters into 4 bytes each.

The IBM Informix ODBC Driver supports UCS-4 on UNIX platforms.

UTF-8 Encoding standard that is based on a single (8 bit) byte. UTF-8 defines a mechanism to transform all Unicode characters into a variable length (1 to 4) encoding of bytes.

The IBM Informix ODBC Driver uses UTF-8 encoding for all UNIX applications that connect to the Data Direct (formerly Merant) driver manager.

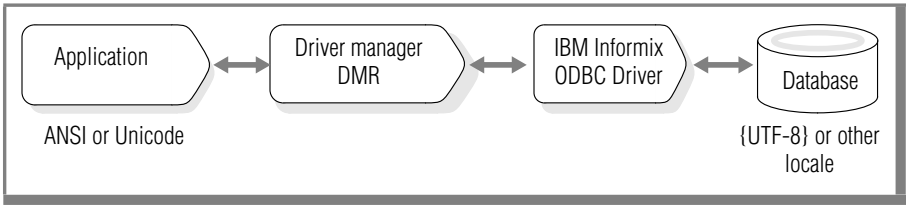
7-bit ASCII characters have the same encoding under both ASCII and UTF-8. This has the advantage that UTF-8 can be used with much existing software without extensive revision.

Important: *In applications that use Unicode, the driver does the work of codeset conversion from Unicode to the database local and vice versa. Please note that UTF-8 is the only type of Unicode codeset that can be set as the client locale.*

Unicode in an ODBC Application

The following diagram shows the architecture of a typical ODBC application using a driver manager and the IBM Informix ODBC Driver.

Figure 9-1
Typical ODBC Application Architecture



In this scenario, if an application is making calls to Unicode enabled APIs, then it must be connected to a Unicode enabled IBM Informix ODBC Driver (Version 3.8 and above) to ensure there is no loss of data. If the application is making calls to ANSI ODBC APIs, the application can be linked to either a Unicode enabled driver or an ANSI driver.

Note that the IBM Informix ODBC Driver continues to support IBM Informix GLS. Hence all data fetched in character buffers will be fetched in the client locale codeset. Only data fetched using wide character buffers will use Unicode.

On Windows, if the ODBC driver is not Unicode enabled, the ODBC Driver Manager maps all Unicode API function calls to ANSI ODBC APIs.

If the ODBC driver is Unicode enabled, the Windows ODBC driver manager (version 3.5 or later) maps all ANSI ODBC APIs to Unicode ODBC APIs. The Data Direct (formerly Merant) driver manager for UNIX also works this way.

Important: *The IBM Informix Driver Manager Replacement (DMR) for UNIX platforms does not map between Unicode and ANSI APIs.*

For details about how the Windows ODBC driver manager handles mapping, please refer to the following URL:

```
http://msdn.microsoft.com/library/en-us/odbc/htm
/odbcfunction_mapping_in_the_driver_manager.asp
```

Using Unicode in an ODBC Application

This section provides details on compiling and configuring Unicode within an IBM Informix ODBC application.

Configuration

Since the IBM Informix ODBC Driver supports different types of Unicode on UNIX platforms, the type of Unicode used by an application must be indicated in the ODBC section of the **odbc.ini** file as follows:

```
[ODBC]
.
.
.
UNICODE=UCS-4
```

On IBM AIX platforms with versions lower than 5L, the supported values for the UNICODE parameter is UCS-2 . For all other UNIX platforms the supported values are UCS-4 and UTF-8.

Important: *A Unicode enabled application must indicate the type of Unicode used in the* ***odbc.ini*** *file. If the Unicode parameter is not set in odbc.ini, the default type is UTF-8.*

It is required that all UNIX ODBC applications must set the Unicode type in the odbc.ini file as follows:

- An ANSI ODBC application on UNIX must set UNICODE=UCS-4
- An ANSI ODBC application on IBM AIX must set UNICODE-UCS-2
- An ANSI ODBC application using the Data Direct (formerly Merant) ODBC driver manager should never indicate a Unicode type other than UTF-8 in the **odbc.ini** file.

The following table provides an overview of the **odbc.ini** settings:

Platform	Driver Manager	odbc.ini Setting
AIX	Data Direct	UTF-8
AIX	DMR or none	UCS-2
UNIX	Data Direct	UTF-8
UNIX	DMR or none	UCS-4
Windows	Windows ODBC Driver Manager	N/A

Unicode Functions Supported

The IBM Informix ODBC Driver supports both ANSI and Unicode version of all functions that accept pointer to character strings or SQLPOINTER in their arguments. The following table describes the two types of functions supported:

ODBC "A" functions	These are the normal ODBC functions that accept single byte (ASCII) data as input for all the character/string parameters.
ODBC "W" functions	These are the Unicode functions that accept "wide characters" as input for all character/string parameters.

The ODBC specification defines these functions using the wchar_t data type. This is the standard C library wide character data type.

The following Unicode "wide" functions are supported by the IBM Informix ODBC Driver:

SQLColAttribute W	SQLColAttributesW	SQLConnectW
SQLDescribeColW	SQLErrorW	SQLExecDirectW
SQLGetConnectAttrW	SQLGetCursorNameW	SQLSetDescFieldW
SQLGetDescFieldW	SQLGetDescRecW	SQLGetDiagFieldW
SQLGetDiagRecW	SQLPrepareW	SQLSetConnectAttrW
SQLSetCursorNameW	SQLColumnsW	SQLGetConnectOptionW
SQLGetTypeInfoW	SQLSetConnectOptionW	SQLSpecialColumnsW
SQLStatisticsW	SQLTablesW	SQLDataSourcesW
SQLDriverConnectW	SQLBrowseConnectW	SQLColumnPrivilegesW
SQLGetStmtAttrW	SQLSetStmtAttrW	SQLForeignKeysW
SQLNativeSqlW	SQLPrimaryKeysW	SQLProcedureColumnsW
SQLProceduresW	SQLTablePrivilegesW	SQLDriversW

Appendix

A

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

> © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X