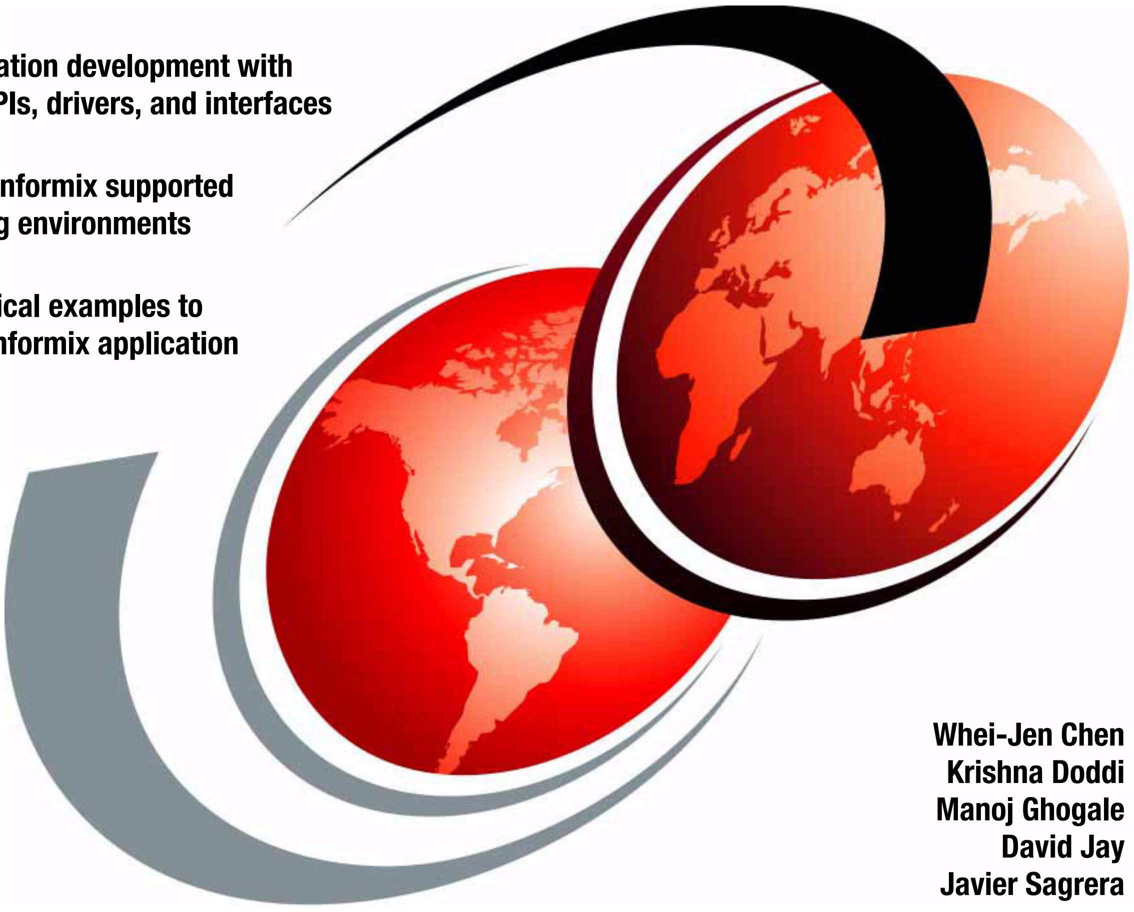
Information Management

IBM

IBM Informix Developer's Handbook



Learn application development with supported APIs, drivers, and interfaces

Understand Informix supported programming environments

Follow practical examples to develop an Informix application

Whei-Jen Chen
Krishna Doddi
Manoj Ghogale
David Jay
Javier Sagrera

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Informix Developer's Handbook

October 2010

SG24-7884-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (October 2010)

This edition applies to IBM Informix Version 11.5.

Contents

Notices

Trademarks

Preface

IBM® Informix® is a low-administration, easy-to-use, and embeddable database that is ideal for application development. It supports a wide range of development platforms, such as Java™, .NET, PHP, and web services, enabling developers to build database applications in the language of their choice. Informix is designed to handle RDBMS data and XML without modification and can be extended easily to handle new data sets.

This IBM Redbooks® publication provides fundamentals of Informix application development. It covers the Informix Client installation and configuration for application development environments. It discusses the skills and techniques for building Informix applications with Java, ESQL/C, OLE DB, .NET, PHP, Ruby on Rails, DataBlade®, and Hibernate.

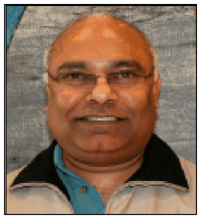
The book uses code examples to demonstrate how to develop an Informix application with various drivers, APIs, and interfaces. It also provides application development troubleshooting and considerations for performance.

This book is intended for developers who use IBM Informix for application development. Although some of the topics that we discuss are highly technical, the information in the book might also be helpful for managers or database administrators who are looking to better understand their Informix development environment.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Rochester Center.

Whei-Jen Chen is a Project Leader at the International Technical Support Organization, San Jose Center. She has extensive experience in application development, database design and modeling, and DB2® system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development, and an IBM Certified IT Specialist



Krishna Doddi (also known as *Prasad*) has been with IBM Informix since 1996. He worked in Informix Advanced Support for Client Products. The main products he supported were Informix Client Software Development Kit (Client SDK), mostly on the Microsoft Windows® platform. After the IBM acquisition in 2001, he moved to DB2 Advanced Support, again in the Client Products division. After five years in DB2, he moved to Informix QA for Informix Client products, including JCC Common Client and PureQuery. He is currently the contact for Informix Integration projects.

Manoj Ghogale is a Tech Lead in the Informix team at ISL India. Manoj has 9 years of industry experience and has been with IBM for 5 years. He has worked on numerous automation projects and new features of Informix Server. He holds a Bachelor's degree in Engineering from National Institute of Engineering, Mysore, India.

David Jay is a Staff Software Engineer in the IBM North Americas Technical Support Team, providing advanced technical support, defect discovery, and support training for Informix products and SolidDB. David joined the Informix Support team in September 1996, and has served in various software and support roles since the 1980's. He has a Bachelor of Science from Pennsylvania State University, and enjoys public speaking assignments whenever he gets them through Toastmasters and IBM.

Javier Sagrera is a software engineer on the Common Client Technologies (CCT) group. He joined the Informix team in 2000 and has over 15 years experience in application development for Informix database servers and Informix clients. Currently based on the IBM UK Bedfont Lab in London, he has extensive knowledge on all the Microsoft® technologies and is considered as a subject matter expert worldwide on all the Informix development tools.

Acknowledgements

Thanks to the following people for their contributions to this project:

Jacques Roy
Ted Wasserman
Rakeshkumar Naik
Jonathan Leffler
Robert Uleman
Richard Snoke
Guy Bowerman
IBM Software Group

Greg Holmes
Alberto Bortolan
Adam Hattrell
IBM Bedfont Laboratory

Emma Jacobs
International Technical Support Organization, Rochester Center

Now you can become a published author, too!

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction to IBM Informix

We live in a business world that requires a variety of software components to fully handle the scope of business needs. When considered as a group, the products have to work together, and they must scale well to cover the demands of business to assure continued growth. Frequently, a significant criteria list might be needed to specify all the facets that are encountered in daily, monthly, and yearly periods of business operations. Developers must be sensitive to how the database engine functions, what server-side capabilities can be brought into play, and what development tasks need to occur on the client or application side.

In this chapter, we provide an overview of Informix products and capabilities. Our approach is somewhat reference oriented. In addition to the overview of Informix products, we also provide an awareness of the architecture of IBM Informix from both the server-side and the client-side, to better acquaint the developer with perspectives that you might need to know.

We also discuss the capabilities of the products in the Informix suite. The IBM Informix products that we describe demonstrate how to manage business performance, handle all types of business applications, minimize administration through various self-automated and self-monitored options and, at all times, optimize the setup to help minimize costs for maintenance and resource allocations.

1.1 Server options

There are a number of options for IBM Informix servers that are designed to meet the needs of business enterprises that range in size from a small group with a few employees up to several thousand people. The database needs to meet a variety of requirements of a diverse group. This section includes a brief discussion of each of the server possibilities and how they are generally used.

With the wealth of different server types available, unless otherwise noted, each comes with the opportunity to use the IBM Informix Client Software Development Kit (Client SDK) for development of applications. Prior to renaming the brand editions in May 2010, IBM Informix had several versions. You can expect that each version is backward and forward compatible, beginning with 7.2x and moving forward through 7.3x, 9.2x, 9.30, 9.40, and 10.00. The Cheetah series began with Informix Dynamic Server 11.10 and quickly advanced into the 11.50 family. At this writing, the IBM Informix Ultimate series is now available, which begins with 11.50.xC7 and following.

It is helpful to be aware of version numbering conventions:

- The letters *UC* or *UD* at the end of a version number indicate 32-bit UNIX®.
- The letters *TC* indicate a 32-bit Windows version.
- The letters *FC* indicate a 64-bit version (UNIX or Windows).

These naming conventions become important in application and memory addressability. A 32-bit application works with a 64-bit server, but the application can access only 32-bit memory.

1.1.1 Informix servers

Several options are available for IBM Informix servers. Each option is designed to address specific needs in business. This section discusses these different options.

No-charge editions

The following no-charge editions are available as separate offerings subject to the IBM International License Agreement for Non-Warranted Programs (ILAN):

- Informix Developer Edition
- Informix Innovator-C Edition

Informix Developer Edition

The Informix Developer Edition provides all the features of Informix at no cost, when used solely for application development and testing. It can be used for application development and prototyping with no time limits. Support is available by way of the Informix user community through Internet forums.

Informix Developer Edition offers the following application development choices:

- C
- C++
- .NET
- Java
- Ruby on Rails
- Perl
- Python
- PHP
- 4GL

Informix Developer Edition supports the following operating systems:

- AIX®
- HP
- UNIX
- Linux®
- Macintosh
- Sun Solaris
- Windows

This Informix edition is limited to 1 CPU, 1 GB memory, and 8 GB storage.

Informix Developer Edition includes the following bundle:

- Informix Client Software Development Kit
- Informix DataBlade Developers Kit
- Informix Spatial Datablade

Informix Innovator-C Edition

The Innovator-C Edition provides a robust and powerful environment that is designed to support small production workloads. It features the most widely used data processing functionality, including limited Enterprise Replication and high availability clustering.

This edition offers the following application development choices:

- C
- C++
- .NET
- Java
- Ruby on Rails
- Perl
- 4GL

Innovator-C Edition supports the following operating systems:

- AIX
- HP
- UNIX
- Linux
- Macintosh
- Sun Solaris
- Windows

This edition is limited to one socket with no more than four cores and with a total of 2 GB of RAM that are operating from the same installation.

Note: Innovator-C Edition is an offering to be used for development, test, and user production workloads without a license fee. This edition can be used only by user organizations. It cannot be redistributed without signing a redistribution contract. Support is community-based though an optional for-charge service and support package is available.

Licensed versions

In a business enterprise that demands constant uptime and consistent delivery requirements, it helps to have a license and professional, timely support to cover every need. There are several licensing options available.

In this section, we discuss the Informix Server versions that are licensed subscriptions. Table 1-1 describes the terms and abbreviations for these editions. The criteria might be subject to change over time. Consult with a sales representative for the most accurate definitions.

Table 1-1 Informix licensed versions

Licensing term	Defined
Processor Value Unit (PVU) (also known as *processor-based pricing*)	Calculated using the number of processor cores in the physical server multiplied by the corresponding value units based on processor architecture. An unlimited user or connection license and is usually the optimal choice when the user or session load cannot be controlled or counted.
Authorized User Single Install	A single named user or specific individual accessing one installation of Informix on each physical or virtual server. That authorized user can establish multiple connections to an Informix instance on the server. Each connection is for the exclusive use of that one authorized user from a single client device.
Concurrent Session	A single logical connection from a client device to an Informix instance on each physical or logical server. Each connection, whether active or not, requires a license, regardless of whether it comes from one client device with multiple users or from a single user establishing multiple connections. The number of concurrent sessions is counted from the client device, rather than at the server level, regardless of whether the connection is directly to the Informix instance or indirectly or whether it is through application servers, persistent connectivity layers, connection multiplexers or concentrators, or any other technology inserted between the actual user and the Informix instance.
Limited Use Socket (LU Socket)	Available only on Informix Growth Edition, this license allows for licensing on a physical socket potentially containing multiple cores. A LU Socket license is required for each active processor socket. This licensing metric can be used only on a physical server with no more than four physical sockets. You can purchase licenses for up to four physical sockets and use up to 16 cores.
Install	An “Install” is an installed copy of an Informix product on a physical server (or partition thereof) or in a virtual machine image. For example, if a physical server is segmented into partitions, whether logical (also know as LPARs) or physical, each partition containing Informix is considered a separate IBM Informix “Install” for licensing purposes and restrictions. The concept of an “Install” applies to the licensing limits specified for all Informix Editions.

Informix Choice Edition

Informix Choice Edition is ideal for small to medium size business. This edition is available for both Microsoft Windows and Apple Macintosh operating systems. This edition is limited to a total of eight cores over a maximum of two sockets and 8 GB of RAM operating. The Informix Choice Edition includes limited Enterprise Replication (ER) clustering with 2-root nodes to send or receive data updates within the cluster. This edition also provides limited high availability (HA) cluster functionality. Along with the primary server, you can have one secondary node, either an HDR secondary or RSS secondary. The HA cluster secondary node can be used for SQL operations. This edition does not support use of the Shared Disk secondary (SD secondary) node type.

Informix Growth Edition

The Informix Growth Edition is available on all platforms (Linux, UNIX, Windows, and Apple Macintosh). The Growth Edition is ideal for mid-sized companies or departmental servers in an enterprise deployment. It can be deployed on up to 16 cores over a maximum of four sockets and 16 GB of RAM operating from the same Install. License options include Authorized User Single Install, Concurrent Session, PVU, and LU Socket metrics. LU Socket enables licensing by physical processor socket and is limited to physical servers with no more than four physical processor sockets.

Informix Growth Edition gives you additional database functionality, including unlimited ER cluster nodes of any type for sending or receiving data updates within the cluster. From a licensing perspective, because ER nodes are stand-alone, each ER node must be fully licensed. Informix Growth edition supports up to two HA cluster secondary nodes of any type. As long as the secondary node or nodes are functioning only as a backup secondary, they can be deployed without charge. However, if you use a secondary node for SQL operations (read or write), the secondary node must be fully licensed.

Informix Ultimate Edition

The IBM Informix Ultimate Edition includes nearly all the Informix features and functionality with unlimited scalability required for the highest OLTP and warehousing performance.

Informix Ultimate Edition supports the following operating systems:

- AIX
- HP
- UNIX
- Linux
- Macintosh
- Sun Solaris
- Windows

This edition can be licensed by PVU, Concurrent Session, or Authorized User Single Install metrics.

With this edition, full HA cluster and ER functionality is available, including unlimited ER nodes and all HA cluster secondary instance types. From a licensing perspective, because ER nodes are stand-alone, each ER node must be fully licensed, but HA secondary nodes can be deployed without charge if they are functioning only as a backup secondary. If you use any secondary node for SQL operations (read or write), the secondary node must be fully licensed.

This edition offers the Geodetic and Excalibur DataBlades add-on.

The Storage Optimization Feature was released with Informix version11.5 xC4 and following. The Storage Optimization Feature provides data compression to dramatically reduce data storage and backup/recovery costs and administration. The reduced data footprint also provides a significant increase in performance for data retrieval.

Extended Parallel Server

The IBM Informix Extended Parallel Server is a high-end database that is typically used for scalable data warehousing with fast data loading and comprehensive data management. It is designed for a broad range of enterprises that require complex, query-intensive analytical applications.

IBM Informix Extended Parallel Server uses the following examples:

- Reliable data manipulation
- Ad hoc queries from data warehouses
- A combined data warehouse and data mart
- Rapid and concurrent data loading and query execution

IBM Informix Extended Parallel Server provides full parallel query processing, while being able to use hardware resources to deliver mainframe-caliber scalability, manageability, and performance with minimal OS and administrative overhead. For more information, see *Database Strategies: Using Informix XPS and DB2 Universal Database*, SG24-6437.

Informix OnLine

IBM Informix OnLine is an easy to use, embeddable relational database server for low-to-medium workloads.

It has less features and functionality than the Informix Ultimate Server, but it scales well while providing online transaction processing support and the assurance of data integrity. Informix OnLine has rich multimedia data management capabilities, supporting the storage of a wide range of media such as documents, images, and audio by way of text and byte columns. Also, it is not designed to handle extended data types or replication.

This server supports a wide variety of application development tools, along with a large number of other third-party tools, through support for the ODBC and JDBC industry standards for client connectivity.

There are three edition options for this server:

- Informix OnLine Extended Edition 5.20

  This edition option is a full-featured, easy-to-use SQL database with low administrative overhead. It contains two popular Informix products, Informix OnLine and Informix STAR, and provides distributed computing with a proven database server. It allows the Rapid Application Development tools to use pipes to communicate with OnLine Extended Edition 5.20 servers running on the same system, instead of having to use a TCP/IP loopback connection with Informix Star or Informix Net. It supports greater than 2 GB chunk offsets, to use the entire capacity of the disk drive without partitioning it into smaller logical devices and employing logical volume managers.

  The operating systems this edition supports include AIX, HP UNIX, and Sun Solaris.

- Informix OnLine Extended Edition for Linux

  This edition is same as the Informix OnLine Extended Edition 5.20 but is but only available for Linux.

- Informix OnLine Personal Edition

  This is the single-user product version, available only on Linux. It provides the same functionality as IBM Informix OnLine Extended Edition, at an economical cost. It enables you to quickly become familiar with the ease-of-use and multimedia capability of this proven relational database management system.

Standard Engine

IBM Informix Standard Engine is an embeddable database server that runs on UNIX, Linux and Windows. It provides an ideal solution for developing small to medium-sized applications that need the power of SQL without database administration requirements (low-maintenance, high-reliability). For limited scale databases, it delivers excellent performance, adheres to data consistency standards, and still provides client/server capabilities. It can be seamlessly integrated with Informix application development tools and third-party development tools compliant with the ODBC and JDBC standards.

This edition supports the following operating systems:

- AIX
- HP UNIX
- Linux
- Sun Solaris
- Windows

1.2 Informix tools for developers

There is a significant list of tools available from Informix for developers. We focus on the available application programming interface (API) options and language possibilities. There are also other utilities and tools that are provided with the engine that promote further ease of use.

The 4GL developer's edition includes the ACE Report Writer, which can be used to generate forms and reports quickly. The High Performance Loader, onload, onunload, and External table are bulk load and unload utilities that provide fast flat file movements. Other tools include I-Spy for auditing, MaxConnect, and the OpenAdmin Tool (OAT) for remote server administration and SQL.

For more information about these or other tools, consult with a sales representative or visit the Informix support website at:

http://www-947.ibm.com/support/entry/portal/Overview/Software/Information_Management/Informix_Product_Family

1.2.1 Informix Connect

Informix Connect is the run time version of Client SDK that comes with the server engine. You use Client SDK and related tools to develop your application. When it is ready for use in production, Informix Connect is the tool used to deploy the application. It is supplied with the server engine software and provides the connectivity and runtime libraries which permit interaction between the engine and the application. While some Informix Connect development can be done without Client SDK, the great majority should be handled through the Client SDK API.

1.2.2 Informix Client Software Development Kit

Client SDK is a package of several APIs that are optimized for developing applications for IBM Informix servers. Client SDK allows developers to write applications in the language they prefer and to build applications that can access multiple IBM Informix databases. In this section, we discuss the API packages that are included in Client SDK.

Open Database Connectivity

Open Database Connectivity (ODBC) is a specification for a database API. It is based on the Call Level Interface specifications from X/Open and the International Standards Organization and 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 can perform the following operations:

- Connect to and disconnect from data sources.
- Retrieve information about data sources.
- Retrieve information about the 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 before or after the SQL 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. ODBC supports Secure Sockets Layer (SSL) connections. For information about using the SSL protocol, see *IBM Informix Version 11.5 Security Guide,* SC23-7754.

Informix ODBC supports the following additional features and capabilities:

- Microsoft Transaction Server (MTS) environment. For more information about MTS, see the MTS sections in *IBM Informix ODBC Driver Programmers Manual,* SC23-9423.
- ODBC can handle extended data types such as:
  - Collection (LIST, MULTISET, and SET)
  - Distinct
  - Opaque (fixed, unnamed)
  - Row (named, unnamed)
  - Smart large objects (BLOB and CLOB)
  - Client functions that support extended data types
- Long identifiers
- Global Language Support (GLS) data types (NCHAR,NVARCHAR).
- Support for Unicode and XA.
- IPv6 internet protocol.

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

- Driver manager

  An application can link to a driver manager that links to the driver specified by the data source. The driver manager also checks parameters and transitions. You can purchase the ODBC Driver Manager from a third-party vendor for most UNIX platforms. On Microsoft Windows platforms, the ODBC Driver Manager is a part of the Operating System.

- IBM Informix ODBC Driver

  This driver provides an interface to the 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 and the driver
  – To link to the driver directly

- Data sources

  The driver provides access to the following data sources:

  – Database management systems (DBMS) and database servers

  – Databases

  – Operating systems and network software required for accessing the database

JDBC and SQLJ

JDBC is an application programming interface (API) that Java applications use to access relational databases. IBM Informix database systems can be supported by one of two APIs for client applications and applets written in Java:

- IBM Data Server Driver

  You can use the IBM Data Server Driver (also known as the IBM common client driver) with either DB2 or Informix. This JDBC driver lets you write Java applications to access a local Informix Server, DB2 data, or any remote relational data on a server that supports DRDA®. Using this API, you can access these database systems using JDBC, SQLJ or pureQuery. SQLJ provides support for embedded static SQL in Java applications.

  Note: The IBM Data Server Driver is introduced with Client SDK beginning with Client SDK 3.50.XC7. At this writing, the Client SDK distributed with Informix database server is one (1) version less than the server version. As an example, Client SDK 3.50.XC6 is the version distributed with Server edition 11.50.XC7.

The IBM Data Server Driver for JDBC and SQLJ is a single driver that includes JDBC Type 2 and JDBC Type 4 behavior. For connections to IBM Informix databases, only Type 4 behavior is supported. IBM Data Server Driver for JDBC and SQLJ Type 4 driver behavior is also referred to as *IBM Data Server Driver for JDBC and SQLJ type 4 connectivity*. For more information about these APIs, see the Java discussions at:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.apdv.java.doc/doc/rjvjdb2o.html`

- IBM Informix JDBC Driver

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

When deciding which JDBC interface to use, it is important to understand the differences between these two APIs. If you need to work with extended data types and features that are unique to IBM Informix databases, use IBM Informix JDBC Driver.

To support `DataSource` objects, connection pooling, and distributed transactions, IBM Informix JDBC Driver provides classes that implement interfaces and classes described in the JDBC 3.0 API from Sun Microsystems.

Informix classes implementing Java interfaces

Table 1-2 lists the Java interfaces and classes and the Informix classes that implement them.

Table 1-2 Java interfaces and classes

JDBC interface class	Informix class
java.io.Serializable	com.informix.jdbcx.IfxCoreDataSource
java.sql.Connection	com.informix.jdbc.IfmxConnection
javax.sql.ConnectionEventListener	com.informix.jdbcx.IfxConnectionEvent-Listener
javax.sql.ConnectionPoolDataSource	com.informix.jdbcx.IfxConnectionPoolData-Source
javax.sql.DataSource	com.informix.jdbcx.IfxDataSource
javax.sql.PooledConnection	com.informix.jdbcx.IfxPooledConnection
javax.sql.XADataSource	com.informix.jdbcx.IfxXADataSource
java.sql.ParameterMetaData	com.informix.jdbc.IfxParameterMetaData

IBM Informix JDBC Driver, Version 3.0, and later implements the `updateXXX()` methods defined in the ResultSet interface by the JDBC 3.0 specification. These methods, such as updateClob, are further defined in the J2SDK 1.4.x API (and later versions) and require that the ResultSet object can be updated. The `updateXXX` methods allow rows to be updated using Java variables and objects and extend to include additional JDBC types. These methods update JDBC types implemented with locators, not the data that is designated by the locators.

Informix classes Java specification

To support the Informix implementation of SQL statements and data types, IBM Informix JDBC Driver provides classes that extend the JDBC 3.0 API. For information about the Java classes and the Informix classes that application programs can use to extend them, see *IBM Informix JDBC Driver Programmer's Guide, v3.50*, SC23-9421.

Because IBM Informix has extended functionality, extra data types, and smart large objects, several Informix classes provide support for functionality that is not present in the JDBC 3.0 specification. Table 1-3 lists these classes.

Table 1-3 Informix classes beyond the Java Specification

JDBC interface or class	Informix class	Provides support for
java.lang.object	UDTManager	Deploying opaque data types in the database server
java.lang.object	UDTMetaData	Deploying opaque data types in the database server
java.lang.object	UDRManager	Deploying user-defined routines in the database server
java.lang.object	UDRMetaData	Deploying user-defined routines in the database server

In releases prior to JDK Version 1.4, the UDTManager and UDRManager helper classes included in `ifxtools.jar` were not accessible from a packaged class. As of IBM Informix JDBC Driver 2.21.JC3, all these classes are in the `udtudrmgr` package. For backwards compatibility, unpackaged versions of these classes are also included. To access a packaged class, use the following import statements in your program:

- `Import udtudrmgr.UDTManager;`
- `Import udtudrmgr.UDRManager;`

OLEDB

Microsoft OLE DB is a specification for a set of data access interfaces that are designed to enable a variety of data stores to work together seamlessly. OLE DB includes the following components:

- Data providers
- Data consumers
- Service components

Each *data provider* makes data available to consumers in a tabular form through virtual tables. *Data consumers* use the OLE DB interfaces to access data. You can use the IBM Informix OLE DB Provider to enable client applications, such as ActiveX Data Object (ADO) applications and web pages, to access data on an Informix server.

You can find detailed information about the characteristics of the IBM Informix OLE DB Provider in *IBM Informix OLE DB Provider Programmer's Guide, Version 3.50*, SC23-9424. The IBM OLEDB provider works with any IBM Informix Server version greater than or equal to 7.3 (7.3, 8.x, 9.x,10.x or 11.x). For information about OLE DB architecture and programming, go to the Microsoft website and search for “Introduction to OLE DB” topic:

`http://www.microsoft.com`

ESQL/C

Informix ESQL/C is an API that enables you to embed SQL statements directly into a C program by means of the Informix ESQL/C preprocessor, **`esql`**. The preprocessor converts each SQL statement and IBM Informix-specific code to C-language source code and then invokes the C compiler to compile it.

Informix ESQL/C includes the following components:

- The Informix ESQL/C libraries of C functions, which provide access to the database server, and all the Informix data types
- The Informix ESQL/C header files, which define the data structures, constants, and macros useful to an Informix ESQL/C program
- The **`esql`** command, which processes the Informix ESQL/C source code to create a C source file that it passes to the C compiler
- The `finderr` utility on the UNIX system and the Informix Error Messages Windows-based utility that provides information about IBM Informix specific error messages
- GLS locale and code set conversion files for locale specific information

1.2.3 4GL

IBM Informix 4GL is a fourth-generation application development and production environment that provides power and flexibility without the need for third-generation languages such as C. The following package options are available for all UNIX and Linux operating systems:

- Informix 4GL Rapid Development System and Informix 4GL Interactive Debugger provide a pseudo-compiled development environment for applications.
- Informix 4GL C Compiler provides the components needed to develop and compile a high-performance application for a production environment.

1.2.4 Ruby on Rails

IBM Informix supports database access for client applications written in the popular and dynamic open source programming language called Ruby. To accomplish this, the developer must use a Ruby Driver and a Rails Adapter for the standard Ruby framework Rails. IBM Informix offers the following methods for using Ruby and Ruby on Rails:

- The Ruby driver and Rails Adapter for IBM Data Servers is supported on Informix, Version 11.10 or later, using the DRDA protocol. This option requires the IBM Data Server Driver for ODBC and the call level interface (CLI) which are available as part of the Informix Client SDK.
- Ruby/Informix and Rails Informix_adapter are specific for Informix database server and work with Informix database connections. Support all versions of the IBM Informix database servers and require the Client SDK libraries for the communication with the database server.

You can download the both packages from the Rubyforge website at:

`http://rubyforge.org/projects/ruby-informix/`

1.2.5 Informix DataBlade Developers Kit

The IBM Informix DataBlade Developers Kit (DBDK) is an aid for developing DataBlade modules. The kit runs on Microsoft Windows and generates much of the code you need for a DataBlade. DataBlades can be developed without a DBDK on operating systems other than Windows; however, it is noteworthy that the procedures for setting up the development environment on a UNIX system are complex without help from DBDK.

The DataBlade Developers Kit provides several graphical user interfaces for creating and working with Informix DataBlade modules.:

- BladeSmith: A tool for organizing a DataBlade module development project. You can use BladeSmith to create a project and define the objects, such as data types and routines, that belong to the DataBlade module. BladeSmith generates source files, header files, make files, functional test files, SQL scripts, messages files, and packaging files.
- DBDK Visual C++ Add-In and IfxQuery: Tools for debugging a DataBlade module using Microsoft Visual C++ on Windows. The add-in automates many of the debugging tasks and calls the IfxQuery tool to run unit tests for DataBlade module routines.
- BladePack: A tool for creating a DataBlade module package. BladePack can create a simple directory tree containing files to be installed or an installation that includes an interactive user interface.
- BladeManager: A command line tool on UNIX (or Windows) that comes with the Informix Server. It is a utility that is needed for registering and unregistering DataBlade modules in Informix databases.

1.2.6 Informix Spatial DataBlade

Many of the IBM Informix database servers support DataBlades. Of these DataBlades, a couple of blades are fairly popular because of their usefulness and flexibility. We introduce one of these blades here to open your interest into an area that you might not have considered. Location based data is one of several features and benefits that can be found in the IBM Informix Spatial DataBlade.

The Spatial DataBlade can transform both traditional and location-based data into essential information through the following functions:

- Expands IBM Informix Server to provide SQL-based spatial data types and functions that can be used directly through standard SQL queries or with client-side Geographic Information Systems (GIS) software.
- Delivers innovative spatial technology through a convenient no-charge download.
- Generates vital business intelligence for a competitive edge.
- Maximizes spatial data capabilities to enable critical business decisions.
- Works in an enterprise replication environment which includes spatial data types.
- Enables organizations to manage complex geospatial information alongside traditional data, without sacrificing the efficiency of the relational database model.

- Includes R-tree indexing.

  R-tree is built into the database kernel and works directly with extended data types to enable proper geospatial data management. Unlike standard indices, the R-tree does not divide space into a full coverage of non- overlapping, adjacent cells. Instead, it uses data partitioning, where each object is automatically represented by a bounding box that is entirely determined by its own shape. These bounding boxes might overlap and do not need to cover the entire space. As a result there is no need to know the spatial extent of the data in advance.

1.2.7 PHP on Informix

PHP, the powerful and popular server-side scripting language for creating web content, has become an important platform for Informix development. The Informix OpenAdmin Tool (OAT), which provides the ability to administer multiple Informix instances from a single location, is written entirely in the PHP language.

Informix supports database access for client applications written in the PHP programming language by using a PDO (PHP Data Object) extension that functions as a database extraction layer. The primary PHP driver available for Informix is called PDO_IBM, and is supported on Informix Version 11.10, and following. The other available PHP driver is PDO_INFORMIX. It is the older of these two, and is the driver used for the Informix OAT. You can find the drivers for PHP on the PECL for PHP website at:

`http://pecl.php.net/package/PDO_IBM`

1.3 Informix overview

This section provides a description of the database architecture from a developer perspective. There are entire manuals to describe server side functions and administration. In this section, we limit our overview to things that the developer should consider. For more information about the details of the IBM Informix Server engine, consult the *IBM Informix Administrators Guide*, which is available at:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.adref.doc/adref.htm`

1.3.1 Architecture overview

An understanding of the Informix database server architecture is beneficial for application development.

Client/server architecture

IBM Informix servers are based on a multi-threaded client/server architecture. Regardless of the edition that you use, the server can support a large number of client connections while users are accessing data. A quick overview provides insights into what aspects influence development and what aspects might need DBA attention.

The IBM Informix Server edition environments consist of several parts that are usually not directly visible to the developer. These parts include shared memory, disk storage, and virtual processors (VPs). The virtual processors are divided into VP classes to manage specific tasks, such as SQL query processing (CPU), physical and logical log processes (PIO and LIO), network connection processes (NET), engine administration (ADM), miscellaneous (MSC), and disk I/O (AIO and KAIO) processes. On UNIX and Apple Macintosh systems, VPs are visible as virtual processes. On Windows environments, the VPs are seen as threads.The program execution process is visible as *oninit.* Virtual processes communicate by way of shared memory structures known as *mutexes* (mutually exclusive). Because shared memory is also used to move database data in buffers, it helps to know that some memory buffers are reserved for engine processing and some buffers allocated (and managed dynamically) for data processing.

Network protocols and other connection types

Users can connect to IBM Informix Server using a network connection, shared memory, pipes, DRDA, or multiplexed connections. Shared memory and pipe connections are only available on the same system as the server instance. Network connections are the most popular, because they can connect different systems and will use either a TCP/IP or a socket connection, depending on the operating system. Regardless of the method that you choose, it is important to minimize the overhead costs associated with opening and closing connections. Keep sessions open only as long as it is prudent, and re-use connections when possible.

Authentication and user connections

By default, authentication is determined for users connecting to a database by the existence of a user name in the operating system environment. Other authentication methods include lightweight directory access protocol (LDAP), or single sign-on (SSO) through Kerberos.

If you need a secured authentication method, IBM Informix Server also has two secured connection methods by way of the encrypted Communication Support Module (CSM) and single password CSM. You can use one or the other, but not both at the same time. Neither the encrypted CSM not the single password CSM method work over multiplexed connections. Enterprise Replication (ER) and high availability (HA) replication support encryption but not with CSM. Secure Sockets Layer (SSL) communications are an alternative to Informix-specific encryption CSMs. SSL encrypts data end-to-end and secures TCP/IP and Distributed Relational Database Architecture™ (DRDA) connections between two points over a network. For setup information, see the developerWorks® article *Protect your data with Secure Sockets Layer support in Informix Dynamic Server, Part 1: Setting up SSL support in IDS*, which is available at:

http://www.ibm.com/developerworks/data/library/techarticle/dm-0912securesockets1/index.html

Programming and user process considerations

Any SQL or stored procedure call is parsed and optimized when the engine first receives the statement. Parsing and optimization methods require a little extra running time. If you can minimize this time, you can speed up query execution. The following factors can speed up execution time:

- If you cannot optimize the query syntax in advance, have the DBA set the OPTCOMPIND parameter in the `onconfig` file to 2 to use distribution data (determined by Update statistics usage).
- If you can optimize all of your query syntax expressions in advance, you might want your DBA to consider using OPTCOMPIND 1 or 2.
- If the same statement is executed multiple times, it is placed in a procedure or dictionary cache. Cache statements run faster.
- If the statement is a prepared statement, the statement can be shared by multiple users and does not need reparsing or optimizing.
- When a statement execution thread is ready to run, it is scheduled to run on a CPU VP. On a system with multiple processors where each CPU VP maps to a physical processor, multiple statements can run concurrently on multiple CPU VPs. Take advantage of multiple CPUs whenever possible by means of concurrent queries, parallel subqueries, and the degree of parallelism.
- When a statement runs on the CPU VP, it is given a brief block of time to run. At the end of the time unit, it is placed into a VP wait cycle, and the next statement thread gets a time block on the VP queue. This use of a wait cycle gives all executing processes an opportunity to make progress, regardless of the number of processes that are running.

Physical and logical logging

As each of the various CPU processes are running, the database engine keeps track of a number of operations. When data is retrieved from disk, the data image is placed in a physical log buffer. Change process information is placed in the logical log. When image data is affected by SQL processing, the physical image is modified periodically and written back to disk.

During this process the physical log image and logical log information are also written into log records. Depending on the commit protocols and isolation levels, this information is available to the DBA through recovery operations in the event that the information is needed. The DBA can also use automatic recovery features that are built in to the engine can if the engine abruptly goes offline due to a power failure or crash.

Commit protocols

If transactions are made against a database that uses *unbuffered* logging, the records in the logical-log buffer are guaranteed to be written to disk during commit processing. When control returns to the application after the COMMIT statement (and before the PREPARE statement for distributed transactions), the logical-log records are on the disk. The database server flushes the records as soon as any transaction in the buffer is committed (that is, a commit record is written to the logical-log buffer).

If transactions are made against a database that uses *buffered* logging, the records are held (buffered) in the logical-log buffer for as long as possible. They are not flushed from the logical-log buffer in shared memory to the logical log on disk until one of the following situations occurs:

- The buffer is full.
- A commit on a database with unbuffered logging flushes the buffer.
- A checkpoint occurs.
- The connection is closed.

If you use buffered logging and a failure occurs, you cannot expect the database server to recover the transactions that were in the logical-log buffer when the failure occurred. Thus, you might lose some committed transactions. In return for this risk of using buffered logging, performance during alterations improves slightly.

Consider using buffered logging only if the following situations:

- If the database is updated frequently (speed of updating is important).
- If the application that is performing the transaction can continue as it is and if you decide that the price (in time and effort) of returning the database to a consistent state by either removing the effects or reapplying the transaction is too high or you can re-create the updates in the event of failure.

  If a transaction failure does occur, you can simply choose to leave the database in its inconsistent state if the transaction does not significantly affect database data.

As you consider whether to use a buffered or unbuffered logging method, remember that no automatic process or utility can perform a rollback of a committed transaction or can commit part of a transaction that has been rolled back. Without detailed knowledge of the application, messages are not enough to determine what has happened. Based on your knowledge of your application and your system, you might need to help the DBA determine when to roll back or to follow though on interrupted transactions. For more information about this topic, see the *Guide to SQL: Tutorial* section on interrupted modifications, which is available at:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.sqlt.doc/ids_sqt_277.htm`

1.3.2 Informix developer environment

The environment references that you need have some variations, depending on whether you use the IBM common client API or the native Client SDK and which operating system you use. The environment settings must include identifying the following references to the components:

- Setting the INFORMIXDIR environment variable to the installation directory of the software installation location.
- Setting up Informix Server information (INFORMIXSERVER).
- Adding the `$INFORMIXDIR/bin` to the PATH environment variable.

On Windows, you have to set only the Informix database server information with the `setnet32.exe` utility. On UNIX-type platforms, you might want to set the environment variables so that they are set in your user profile at login or use a setup script. You also need to reference the `sqlhosts` file to identify Informix Server information. In the chapters that follow, we explain how to install the server and provide specific details for the relevant API environment details.

Before you select an API or library, be aware that some operating systems are not available for some of the tools and that server functionality might not be accessible with some of the APIs. Consider the following information when making an API selection:

- Client SDK
  - Optimized for IBM Informix database servers.
  - Works with PHP.
  - Directly interfaces with Informix Server; no middle tier required.
  - Supports all the Informix data types, extended data types, and smart BLOBs.
- IBM Data Server Client
  - Optimized for compatibility and easy development with Informix and DB2 IBM database servers.
  - Works with PHP and Ruby.
  - No available interface for Apple Mac systems.
  - Includes support for C and Fortran development.
  - No DataBlade API.

1.3.3 Informix capabilities

IBM Informix database servers provide a broad spectrum of features that allow resilience in response to fast changing systems and applications in a modern business environment. If you are aware of what the product can already do before you begin to develop, it aids and enables you to support applications and easily grow to meet new demands.

This section provides a brief overview of the capabilities of Informix.

Resilient

High availability and fast recovery are standard aspects of an on-demand environment. Informix database server engines have a variety of ways to configure the instance, so there is little chance of the engine being unavailable at critical points in time.

Reliable

In addition to availability, there is a need for continuity. Informix has options to protect a server environment, such as backup servers, full independent copies of the processing environment for failover, and workload balancing at alternate locations around the world.

Available

There is an Informix solution to fit any situation that requires availability:

- Continuous availability feature
- High availability replication with allowance for multiple secondary servers

Secure

Informix supports open, industry-standard security mechanisms such as roles, password-based authentication, and RDBMS schema authorizations. These open standards ensure flexibility and security with easy validation and verification. Column-level encryption and Pluggable Authentication Modules (PAM) are also available. The Advanced Access Control Feature offers cell-, column-, and row-level label-based access control (LBAC). Thus, access to data can be controlled down to the individual cell level.

Adaptable

Informix is adaptable from the server side and from the client side. The engine component can be stripped down, embedded, and run with little user intervention. Using the developer tools that we discuss in this book, a developer can provide customized deployment by way of client applications and server-side processes, such as stored procedures, multiple triggers on tables and views, user-defined functions, and DataBlades

Fast

Informix is known for fast OLTP performance. Application performance is helped by capabilities such as committed isolation level and non-blocking checkpoints, which provide maximum concurrency. Direct I/O calls to file systems can result in performance similar to raw device I/O. SQL performance can be improved through techniques or configuration options that redirect or focus the engine's optimizer decisions by way of optimizer directives in the SQL or by way of automated update statistics collection that inform the method used for running the query. When the DBA and the developer both focus on performance, it enables things to run more smoothly and reduces infrastructure costs.

Flexible

There are a number of APIs that are available, both as specific programming language supplements and as interfaces that extend the architecture of the server instance. In Informix 11 and following, the Web Feature Service API allows developers to use location-based services or location-enabled IT services. It is implemented in the Open GeoSpatial Consortium Web Feature Service (OGC WFS) API. This API also interacts with location-based data provided by the IBM Informix Spatial and Geodetic DataBlade modules. There is a significant amount of extensibility when using DataBlade technology.

Hidden, behind-the- scenes tasking

Business systems and applications need to work with minimum invasions into the operations of a business environment. From the server side, applications can run in an automated mode and perform self-maintenance. From an application perspective, database administrative tasks can be controlled and run from inside an application. The SQL Administration API allows you to do tasks such as space management, monitoring and manipulating memory, running scheduled tasks, and monitoring user sessions without developer intervention after they are setup and running.

Customizable install footprint

The installation of Informix can be automated, and the installation footprint can be customized using the Deployment Wizard so that you can limit the installation to only the data server functionality that you need and can reduce the size and costs of a solution for your software deployment.

Affordable, reduced complexity

When IBM or the developer adds features, the first impression is that the application will be more complex. With the features described up to this point, it is a surprise to discover that administration can actually be reduced by means of an application or one of the APIs and GUIs that we explain in this book.

2

Setting up an Informix development environment

This chapter describes how to set up an environment for Informix application development, including the installation and configuration of Informix Server and Informix Client. We focus our discussion on Informix Client products selection, installation, and configuration. Reading this chapter can help you understand how to complete the following tasks:

- Decide which Informix Client product is suitable for your application.
- Install and configure all Informix Client products that can connect to Informix.
- Know any special consideration for each Informix Client in terms of connectivity.

Note: In this chapter, we use the terms *Informix database server*, *Informix Server*, and *Informix* interchangeably. In addition, we also use the terms *Client products*, *Client*, and *Informix Client* interchangeably.

2.1 Server setup

In this section, we describe the various methods of installing Informix Server. We discuss how you can plan for the installation depending upon your specific requirement. We focus the discussion on the configuration that is required at the database server side to enable Informix Client connectivity.

2.1.1 Planning for the installation

In general, an application developer is not involved in the Informix Server product planning and installation. However, you need to communicate the application requirements to the DBA, who plans, installs, and configures the Informix database server for you. For example, the DBA needs to know if you need a smart blob space or a reduced footprint of the server or DataBlades. The Informix installer provides options to install only a small footprint of the server where you can add or remove the features as needed. For more information about DataBlades, see 10.3, “DataBlades and bladelets” on page 352.

2.1.2 Installing Informix Server

IBM provides the Informix Server editions, described in 1.1, “Server options” on page 2, in `.zip`, `.tar`, or `.dmg` format, depending on the platform on which Informix Server is installed. You can install Informix Server using several methods. The common installation methods include using the console, a GUI, or silent mode.

Because we focus on Informix Client products for application development in this book, we describe only the silent mode of Informix Server installation on a UNIX platform. For other methods and platforms, refer to the individual installation PDF file that comes with the product.

To perform a silent installation on UNIX, use the following steps:

1. Extract the compressed `.tar` file into any directory using the following command:

   ```
   tar -xvpf IIF.11.50.tar
   ```

2. Create the installation `.ini` file that specifies the installation options. You can use the sample installation `.ini` file, `server.ini`, as a template. The `server.ini` file is located in the directory that is used to extract the server media package. Edit the following information in the `server.ini` file, and save the file with a new name, for example `myserver.ini`:
   - Change `-P installLocation="/opt/IBM/informix"` to the location of directory where you want to install the product.
   - Change `-G licenseAccepted= option` to `true` to accept the software license. Otherwise, the installation process stops:
3. Start the silent installation with the following command:

   ```
   ./ids_install -silent -options myserver.ini
   ```

Performing these steps installs Informix Server in your specified directory along with a demonstration instance by the name `demo_on`.

For more information about Informix installation and creating new instances, refer to the installation guide PDF that comes with the media file.

2.1.3 Configuring Informix Server

The connectivity information allows a client application to connect to any IBM Informix database server in the network. The connectivity data for a particular database server includes the following connectivity data and information is required:

- ► The host name of the computer or node on which the database server runs
- ► The type of connection that an application can use to connect to the server
- ► The service name or port number used by the database server
- ► The database instance (also called the *database server name*)

Even if the client application and the database server are on the same computer or node, you might need to specify connectivity information for your client application.

Connectivity on UNIX

On UNIX, the `sqlhosts` file contains the connectivity information. By default, this file resides in the `$INFORMIXDIR/etc` directory. You can use the INFORMIXSQLHOSTS environment variable to specify an alternative location for the connectivity information. Edit this file according to the protocol that you are using or the port number to which your client application will connect.

A typical `sqlhosts` file contains the following fields:

```
Server Protocol Hostname Service/Port Options
```

Here is an example:

```
demo_on onipcshm on_hostname on_servername k=0
demo_se seipcpip se_hostname sqlexec
```

Each line in the `sqlhosts` file defines the following information about an Informix database server:

- *`Server`*

  Specifies the name of the IBM Informix database server. This value normally correspond to the value in the INFORMIXSERVER environment variable.

- *`Protocol`*

  Specifies the protocol that is used for the communication with the database server. It must be the same protocol that the database server uses.

- *`Hostname`*

  Specifies the system where the Informix database server is running.

- *`Service/Port`*

  Specifies the TCP port or service name, which must be defined in the `/etc/services` file, that is used by the Informix database server to accept incoming connections.

- *`Options`*

  Specifies additional options for the communication, such as buffer size or connection redirection rules.

APIs such as ESQL/C or ODBC rely on the `sqlhosts` file to obtain the connection parameters for the database server.

Example 2-1 shows how a ESQL/C application connects to an Informix database server.

Example 2-1 UNIX connection example

```
informix@kodiak:/work$ cat $INFORMIXDIR/etc/sqlhosts
#server         protocol        hostname        service/port
demo_on         onsoctcp        kodiak             demo_on_tcp

informix@kodiak:/work$ grep demo_on_tcp /etc/services
demo_on_tcp     9089/tcp                    # Service for demo_on IDS server

informix@kodiak:/work$ cat connect.ec
#include <stdio.h>
int main()
```

```
{
  EXEC SQL CONNECT TO 'stores_demo@demo_on';
  printf("Connected\n");
}
informix@kodiak:/work$ esql connect.ec -o connect
informix@kodiak:/work$ ./connect
Connected
informix@kodiak:/work$
```

The definition for the `demo_on` Informix Server specifies the communication protocol that is used as `onsoctcp`. The system where the database server is running is `kodiak`, and the TCP port that the server uses is `demo_on_tcp`. The service name is defined as port `9089` in the `/etc/services` configuration file.

The ESQL/C example connects to the server and opens the `stores_demo` database using the following instruction:

```
EXEC SQL CONNECT TO 'stores_demo@demo_on';
```

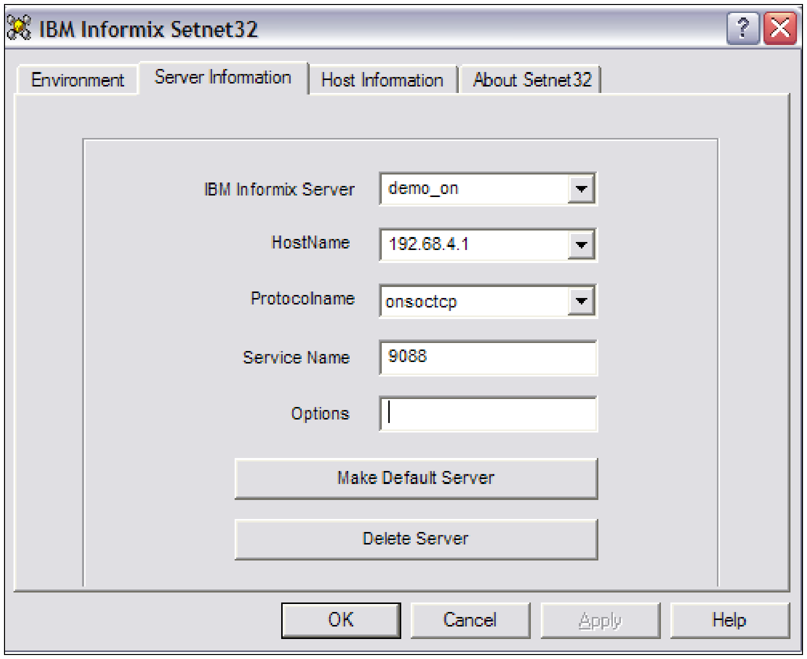
For more information about the `sqlhosts` file, refer to the IBM Informix Dynamic Server (IDS) Information Center topic:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.admin.doc/ids_admin_0158.htm

Connectivity on a Windows system

You can set the connectivity on a Windows system using the `setnet32.exe` utility that resides in the `$INFORMIXDIR/bin` directory. This utility is bundled with IBM Informix Client Software Development Kit (Client SDK). Use this utility to create or change the server name, protocol, service name, and host name. You also can use the `setnet32.exe` utility to set any environment variable that Informix products use.

Figure 2-1 shows the Server Information tab of the `setnet32.exe` utility on a Windows system. Here, you can specify the server name, host IP address, network protocol, and service name.



IBM Informix Server: demo_on
HostName: 192.68.4.1
Protocolname: onsoctcp
Service Name: 9088
Options:

Figure 2-1 The Setnet32 Server Information tab

Note: If you specify the name of the service (for example `sqlexec`) rather than the port number, make sure the service name that you use is defined in the Windows system `services` file that is located in the `C:\WINDOWS\system32\drivers\etc` directory.

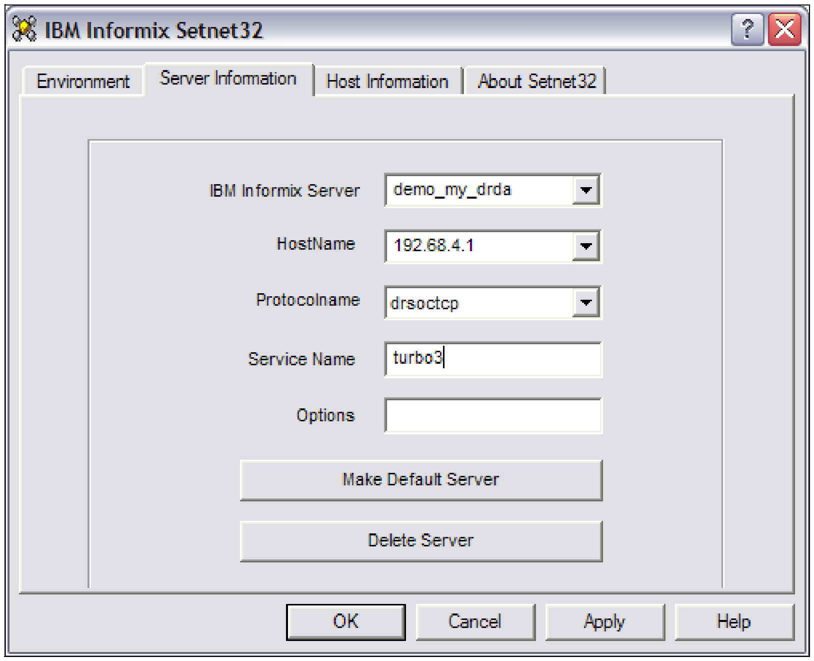
Enabling DRDA support

Since version 11.x, Informix supports the DRDA protocol. If you plan to use the Data Server driver packages to connect to an Informix database server, you must enable DRDA support as described in this section.

Windows systems

Use the `setnet32` utility to configure an Informix instance to use DRDA protocol:

1. Start the `setnet32` utility from any command window.
2. Go to Server Information tab (Figure 2-2).
   a. Select the required instance from the IBM Informix Server field. Our example is `demo_my`.
   b. Change the name of the Informix server, and specify a new name for the DRDA alias, for example `demo_my_drda`.
   c. The HostName field is pre-filled. If it is not, enter your host name or IP address.
   d. Change the Protocolname field from **olsoctcp** to **drsoctcp** to make it DRDA compliant.
   e. Change the Service Name field to specify the service name or port number that the DRDA alias will use.



IBM Informix Server: demo_my_drda
HostName: 192.68.4.1
Protocolname: drsoctcp
Service Name: turbo3
Options:

Figure 2-2 Configuring DRDA support

3. Click **Apply**, and then click **OK**.

After adding all the connection details for the DRDA alias, the configuration file for the Informix database server must be updated to link the new server definition with the existing Informix instance. In our example, we use `demo_my` for the Informix database server and `demo_my_drda` for the DRDA instance.

You can use the DBSERVERALIASES parameter in the `onconfig` file to specify an alias for a database server, as shown in Example 2-2.

Example 2-2 Using the DBSERVERALIASES parameter

```
...
DBSERVERNAME    demo_my                # Name of default Dynamic Server
#     DBSERVERNAME or DBSERVERALIASES that uses a
# DBSERVERALIASES - The list of up to 32 alternative UDRs,
DBSERVERALIASES demo_my_drda            # List of alternate dbserver names
#     DBSERVERNAME or DBSERVERALIASES that uses ...
...
```

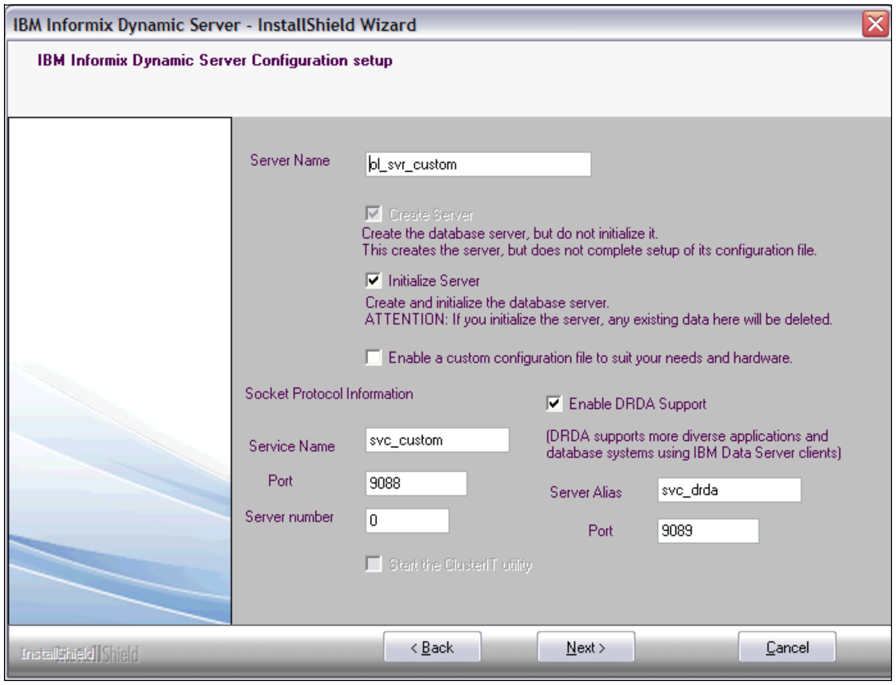
The database configuration file is located in `etc` directory of your Informix database server. You can use the ONCONFIG environment variable to specify the name of this file. The complete path using environment variables for the database configuration file is:

- On a UNIX platform, `$INFORMIXDIR\etc\$ONCONFIG`
- On a Windows platform, `%INFORMIXDIR%\etc\%ONCONFIG%`

You must restart the Informix database engine for the changes to take effect.

After you complete these steps, the instance is now ready to accept connections as a DRDA compliant server.

On a Windows system, you can enable the DRDA support for the demonstration instance during the installation if you use the Custom installation option. Figure 2-3 shows the installation panel where you enable the DRDA support.


IBM Informix Dynamic Server - InstallShield Wizard

IBM Informix Dynamic Server Configuration setup

Server Name: ol_svr_custom

[✓] Create Server
Create the database server, but do not initialize it.
This creates the server, but does not complete setup of its configuration file.

[✓] Initialize Server
Create and initialize the database server.
ATTENTION: If you initialize the server, any existing data here will be deleted.

[ ] Enable a custom configuration file to suit your needs and hardware.

Socket Protocol Information

Service Name: svc_custom

Port: 9088

Server number: 0

[ ] Start the ClusterIT utility

[✓] Enable DRDA Support
(DRDA supports more diverse applications and database systems using IBM Data Server clients)

Server Alias: svc_drda

Port: 9089

Figure 2-3 Enable DRDA support for the default instance

Linux and UNIX systems

On Linux and UNIX systems, you can enable the DRDA support by editing the following Informix configuration files:

- Edit the `sqlhosts` file specified in the $INFORMIXSQLHOSTS environment variable, and add the following line:

  ```
  <drda_name> drsoctcp <hostname> <drda_service_name>
  ```

  where:

  – *`<drda_name>`* is the name for the DRDA alias

  – `drsoctcp` is the protocol used by the new alias

  – *`<hostname>`* the name or IP address of the system that is running Informix Server

  – *`<drda_service_name>`* is the name of the TCP service or port number for the DRDA instance

- Edit the Informix Server configuration file located in the `etc` directory of your Informix Server directory to include the new database definition as an alias for the existing Informix Server.

  You can use the environment variables to access this file, for example `$INFORMIXDIR/etc/$ONCONFIG`. This file is the file that contains a list of configuration parameters for Informix Server. Example 2-3 shows the configuration file that we used in our database server.

  Example 2-3 Linux onconfig file sample

```
informix@irk:/usr3/11.50$ grep DBSERV $INFORMIXDIR/etc/$ONCONFIG
# DBSERVERNAME    - The name of the default database server
# DBSERVERALIASES - The list of up to 32 alternative UDRs,
#                     DBSERVERNAME or DBSERVERALIASES that uses a
DBSERVERNAME demo_my
DBSERVERALIASES demo_my_drda
```

You must restart Informix Server for these changes to take effect.

You can find additional information about the `sqlhosts` file at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.admin.doc/ids_admin_0158.htm

2.2 Client setup

Informix is a low-administration, easy-to-use, and embeddable database that supports various application development languages and technologies, such as Java, .NET, PHP, and Web Services. Informix can handle XML data easily and can be extended to handle new data sets using DataBlades.

In this section, we describe how to install and configure Informix Client products to support the application development that you need.

2.2.1 Informix Client options

IBM offers the following Informix Client products that you can use to develop and use an IBM Informix database server:

- IBM Informix Client Software Development Kit (Client SDK) includes several APIs that are designed for developing with an Informix database.
- IBM Informix Connect is a runtime version of Client SDK.

- IBM Informix JDBC Driver is Java driver that is optimized for Informix databases.
- IBM Data Server Client includes the API tools that are needed for development with IBM databases such as DB2 and Informix, plus functionality for database administration and client/server configuration.
- IBM Data Server Runtime Client is the runtime version of the IBM Data Server Client.
- IBM Data Server Driver Package is a lightweight version of the IBM Data Server Runtime that includes only the interfaces and drivers.
- IBM Data Server Driver for JDBC and SQLJ is a Java driver for IBM Data Servers.

Client SDK and Informix Connect have distinct functions:

- Client SDK contains a group of application-programming interfaces (APIs) that developers can use to write applications for Informix, plus other client components. Client SDK must be installed on computers that application programmers will use to write applications.
- Informix Connect contains runtime libraries of the Client SDK APIs, plus other client components. Install Informix Connect on computers that users use to connect to Informix database servers.

The same rule applies to IBM Data Server products. Do not use packages that are designed for development, such as IBM Data Server Client, to deploy an application. Some components, such as Informix JDBC Driver, do not have a specific runtime version, so you can use the same product in both cases.

In addition to these products, you can use open source drivers, such as PHP or Ruby, to develop with an IBM Informix database.

Figure 2-4 illustrates the options that are available to develop with an IBM Informix database.

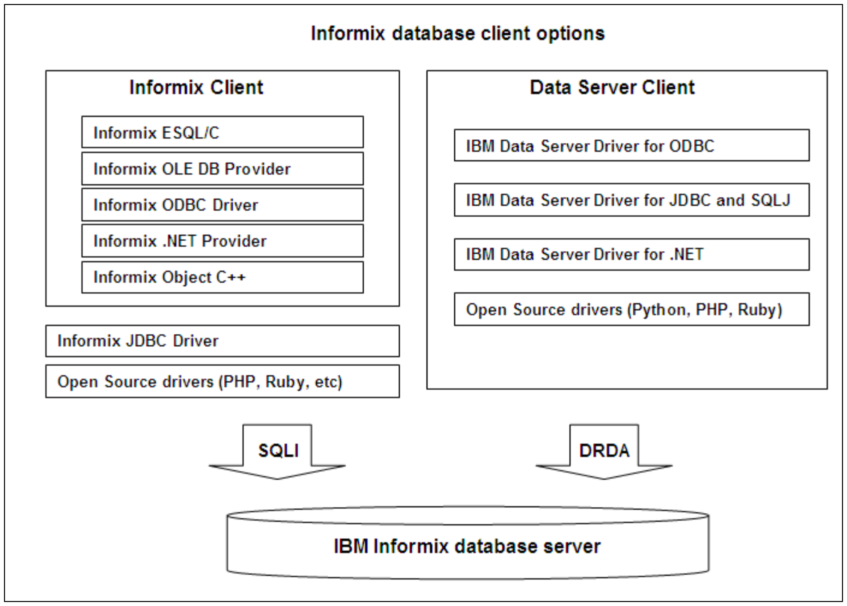

Informix database client options
Informix Client
Informix ESQL/C
Informix OLE DB Provider
Informix ODBC Driver
Informix .NET Provider
Informix Object C++
Informix JDBC Driver
Open Source drivers (PHP, Ruby, etc)
Data Server Client
IBM Data Server Driver for ODBC
IBM Data Server Driver for JDBC and SQLJ
IBM Data Server Driver for .NET
Open Source drivers (Python, PHP, Ruby)
SQLI
DRDA
IBM Informix database server

Figure 2-4 Available Informix Client products for application development

2.2.2 Installing and setting up Client SDK

IBM Informix Client Software Development Kit (Client SDK) is a collection of components for developing and running client applications that connect to the Informix database server. Client SDK bundles with the following components and utilities:

- ESQL/C with XA support
- IBM Informix Object Interface for C++
- IBM Informix OLE DB Provider (Windows only)
- IBM Informix ODBC Driver with MTS support
- IBM Informix .NET Provider (Windows only)
- IBM Informix GLS
- The Global Security Kit (GSKit)
- Password Communication Support Modules (CSM)
- Documentation Viewer

- The `finderr` utility on UNIX systems and the Informix Error Messages utility on Windows systems
- The `iLogin` utility (Windows only)

Starting from version 3.50.xC5, the IBM Data Server Driver Package is also bundled with the Windows system version of Client SDK and Informix Connect.

Client SDK provides access to all the Informix database servers. Table 2-1 lists the Informix database servers that supports Client SDK.

Table 2-1 Supported database servers

Database server	**Versions**
IBM Informix	7.x, 9.x,10.x and 11.50
IBM Informix Extended Parallel Server	8.50 and higher
IBM Informix Standard Engine	7.25
IBM Informix Online	5.20 and higher

Some components, such as Informix OLE DB Provider, do not allow a connection to an IBM Informix Standard Engine server. Refer to the component release notes for more information:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.relnotes.doc/relnotes_ClientSDK350xc7.html`

Completing the preliminary tasks

Before you install Client SDK, perform the following preliminary tasks:

1. Determine the following locations:
   - *Media location*: This directory is the directory where all the media files reside.
   - *Installation location*: This directory is the location where the installer places all the binaries and the configuration files. This directory is represented by the environment variable $INFORMIXDIR. If $INFORMIXDIR is set, the defined location is the default installation location.
   - *Java location*: The script that is used during the installation process requires the use of a Java virtual machine (JVM). A JVM is bundled together with Client SDK. However, if required, it is possible to specify a different JVM for the installation process. The minimum version is Sun JRE 1.4.2.

2. Prepare the environment.

   You need to check if you have the *informix* user and the *informix* group on your computer. If you do not, create them. For additional information, refer to:

   http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.igul.doc/ids_in_005x.htm

Installing the Client SDK

Informix provides various methods for installing Client SDK. In this section, we demonstrate using the GUI to install in both UNIX and Windows environments.

Installing Client SDK on UNIX using the GUI

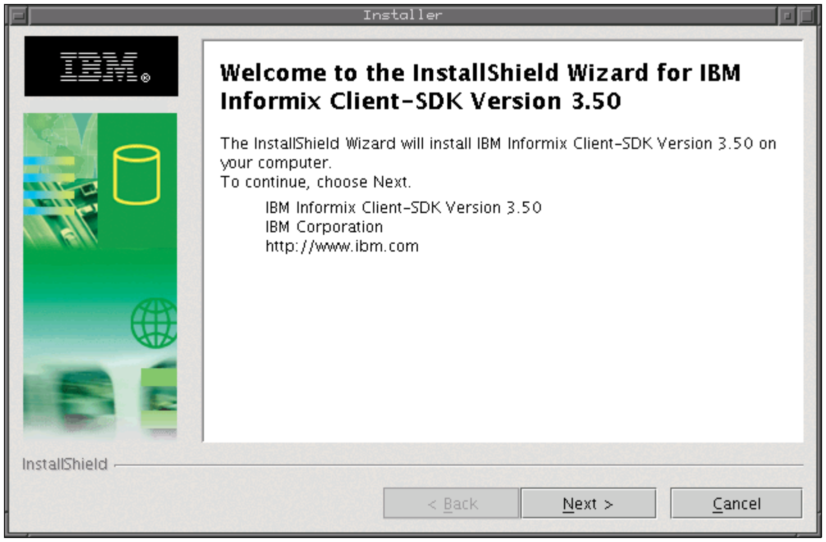
To install Client SDK in GUI mode:

1. Start the installation GUI with the following command from the `root` user:

   ```
   ./installclientsdk -gui
   ```

   Remember to set the DISPLAY environment variable to the window that you want to project the installation. Figure 2-5 shows the Welcome panel.

   Click **Next**.



Welcome to the InstallShield Wizard for IBM Informix Client-SDK Version 3.50

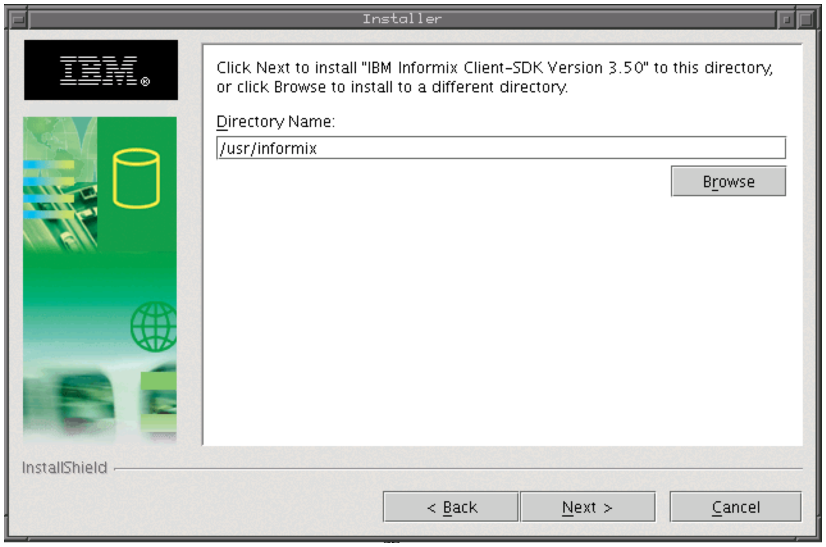
The InstallShield Wizard will install IBM Informix Client-SDK Version 3.50 on your computer.
To continue, choose Next.

IBM Informix Client-SDK Version 3.50
IBM Corporation
http://www.ibm.com

Figure 2-5 Installation of Client SDK on UNIX Welcome panel

2. On the license agreement panel, select **I accept the terms and the license agreement** and click **Next**.

3. Specify the directory where you want to install Informix Client (Figure 2-6). Enter the full path name. Click **Next**.

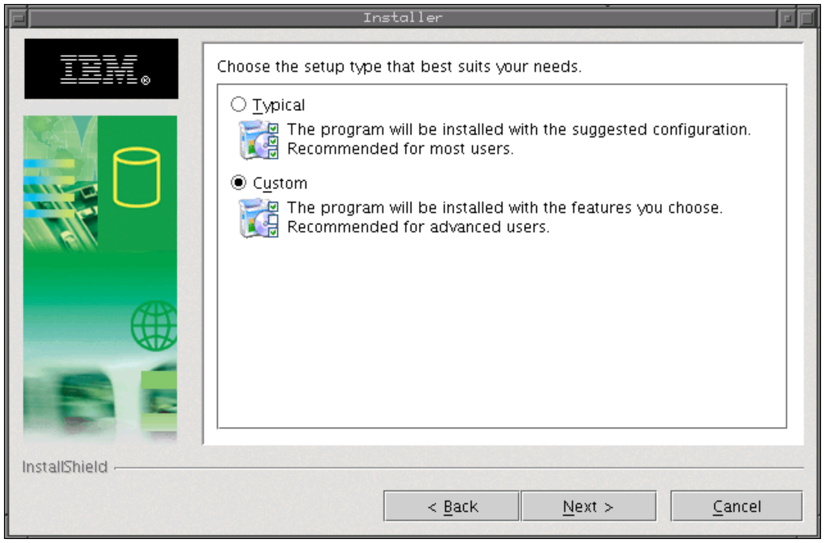


Click Next to install "IBM Informix Client-SDK Version 3.50" to this directory, or click Browse to install to a different directory.

Directory Name:
/usr/informix

Figure 2-6 Specify the installation directory

4. Select **Typical** or **Custom** (Figure 2-7). A Custom installation allows you to select the features to install. Click **Next**.



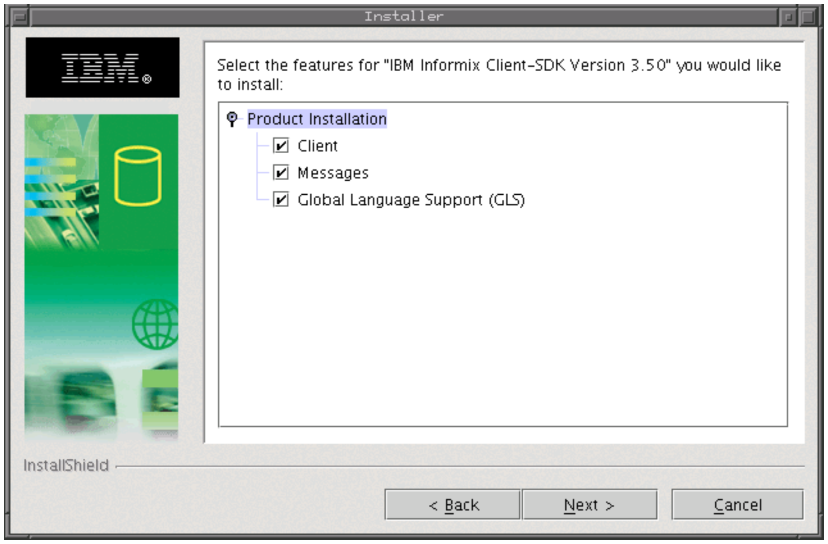
Choose the setup type that best suits your needs.

○ Typical
The program will be installed with the suggested configuration. Recommended for most users.

● Custom
The program will be installed with the features you choose. Recommended for advanced users.

Figure 2-7 Select the type of installation

5. If you selected a Custom installation, you next need to select the features that you want to install and click **Next** (Figure 2-8). If you want to install whatever the installer offers, select a Typical installation.

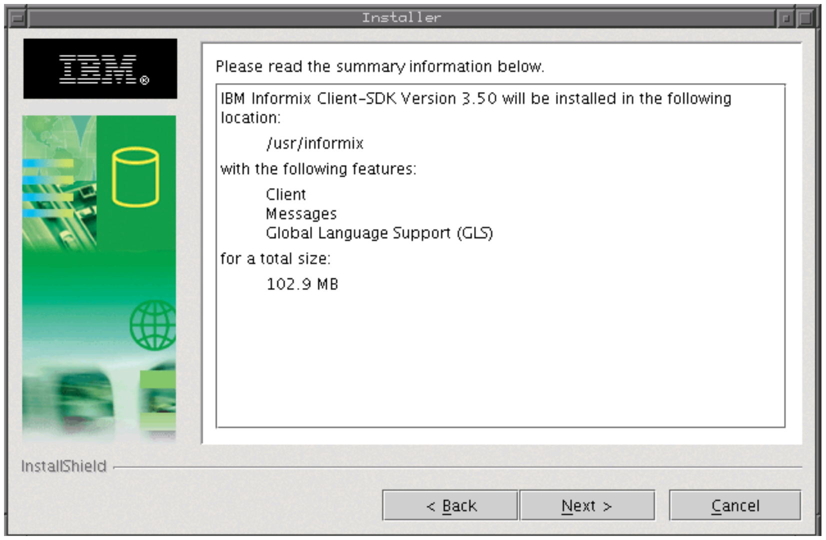

Installer

Select the features for "IBM Informix Client-SDK Version 3.50" you would like to install:

- Product Installation
  - [✓] Client
  - [✓] Messages
  - [✓] Global Language Support (GLS)

Figure 2-8 Custom options to choose

6. The installer presents the summary of available space and the components that you have chosen for installation, as shown in Figure 2-9. If needed, you can go back and change the installation selections. Click **Next** to start the installation.



Please read the summary information below.

IBM Informix Client-SDK Version 3.50 will be installed in the following location:

/usr/informix

with the following features:

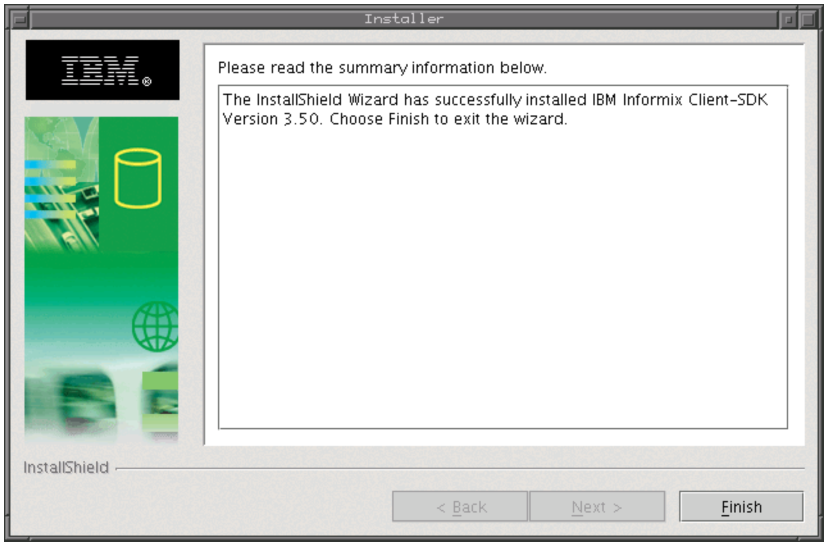
Client
Messages
Global Language Support (GLS)

for a total size:

102.9 MB

Figure 2-9 Installation summary

7. Figure 2-10 shows that the installation is complete without any errors.



Please read the summary information below.

The InstallShield Wizard has successfully installed IBM Informix Client-SDK Version 3.50. Choose Finish to exit the wizard.

Figure 2-10 Installation complete without any errors

At this stage, the Client SDK installation is complete. If the Informix database server is located on a different system, you might need to set up the connection information and modify the `sqlhosts` file, using the details of your Informix Server.

Installing Client SDK on a Windows system

Installation on a Windows system for the Client SDK is similar to UNIX the installation except that the Windows system installation has the option to install IBM Data Server Driver Package. If you select the option to install these drivers, refer to “Installing the IBM Data Server Driver package” on page 44 for Installation details.

Configuration utility

On Windows systems, the Client SDK comes bundled with the `setnet32.exe` utility for configuring the Client SDK. The Client SDK has default values set and is ready to use unless you want to change the protocol, server information, or any environment variable under which the client application runs.

The `setnet32.exe` utility sets environment variables and network parameters that IBM Informix products use at run time. The environment variables and connectivity parameters are stored in the Windows system registry and are valid for every Informix Client product that you install, except Informix JDBC Driver.

The `setnet32.exe` utility has the following tabs:

- The Environment tab allows you to set environment variables.
- The Server Information tab allows you to set database server network information.
- The Host Information tab allows you to set your host computer and login information.
- The About Setnet32 tab provides information about the `setnet32.exe` utility.

For more information about these tabs, refer to the Client SDK installation PDF file that comes with Client SDK.

Special consideration for 32- and 64-bit products

Beginning with version 3.50.FC4, you can install both 32-bit and 64-bit Client SDK on the same Windows system.

Before installing the version 3.50.FC4 product, you must uninstall the existing version 3.50.FC3 or earlier Client SDK from your system.

To uninstall your existing products, use one of the following methods:

- Use the Add/Remove program.
- Run the `setup.exe` program from the installed product media. When the Maintenance Menu displays, select **Remove**.

You must install the new 32-bit and 64-bit products in separate directories.

The 32-bit Client SDK installs in the following directory by default:

```
C:\Program Files (x86)\IBM\Informix\Client-SDK
```

The 64-bit Client SDK installs in the following directory by default:

```
C:\Program Files\IBM\Informix\Client-SDK
```

The 64-bit product installation adds a suffix to the `Program` group folder and the Add/Remove entry. This suffix provides an easy identification between the 32-bit (no suffix) and 64-bit (suffix) product installations. For more information, refer to the release notes.

2.2.3 Setting up IBM Data Server drivers

Several types of IBM Data Server Clients and Drivers are available.

The following IBM Data Server Clients are available:

- IBM Data Server Client, contains all the APIs and tools that are needed for development against DB2 and Informix databases, including tools for remote administration.
- IBM Data Server Runtime Client, contains all the APIs but only the minimum number of utilities for setup and configure the development environment.
- IBM Data Server Driver Package, contains only the APIs and drivers without any additional tools.

The client products contain the APIs and drivers plus additional tools that are designed for remote database administration and configuration of the client environment.

The following IBM Data Server Drivers are available:

- IBM Data Server Driver for JDBC and SQLJ
- IBM Data Server Driver for ODBC and CLI
- IBM Data Server Driver for .NET
- IBM Data Server open source drivers such as PHP or Ruby

You can download some of these components and install them as separate products.

Each IBM Data Server Client and driver provides a particular type of support:

- For Java applications only, use IBM Data Server Driver for JDBC and SQLJ.
- For applications using ODBC or CLI only, use IBM Data Server Driver for ODBC and CLI
- For applications using ODBC, CLI, .NET, OLE DB, PHP, Ruby, JDBC, or SQLJ, use IBM Data Server Driver Package.
- IBM Data Server Runtime Client and IBM Data Server Client includes all of above drivers and some functions of DB2.

In this section, we discuss the installation and configuration of the following drivers for UNIX and Windows platforms:

- IBM Data Server Driver for JDBC and SQLJ
- IBM Data Server Driver for ODBC and CLI
- IBM Data Server Driver Package

For the installation and configuration of IBM Data Server Runtime Client and IBM Data Server Client, refer to:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.swg.im.dbclient.install.doc/doc/c0022615.html`

Installing the IBM Data Server Driver package

The IBM Data Server Driver package is a lightweight solution and the recommended package for user code deployment. It provides robust runtime support for applications using ODBC, CLI, .NET, OLE DB, PHP, Ruby, JDBC, or SQLJ without the need of installing Data Server Runtime Client or Data Server Client. Here we discuss the installation of the drivers presented under the IBM Data Server Driver Package.

Installing the driver package on UNIX and Linux

You can download the IBM Data Server Driver Package and individual drivers such as IBM Data Server Driver for JDBC and SQLJ from the IBM *Support and downloads* website at:

`http://www.ibm.com/support/docview.wss?rs=4020&uid=swg21385217`

Extract IBM Data Server Drivers Package to an empty directory after you download it.

If you want to install the complete IBM Data Server Driver Package, run the **`installIDSDriver`** command. This driver package includes database drivers for

Java, ODBC/CLI, PHP, and Ruby on Rails, each of which is stored in its own subdirectory. The Java and ODBC/CLI drivers are compressed.

If you want to install individual drivers that are contained in the driver package, follow these steps (first two steps are common for all drivers):

1. Extract the Data Server Driver Package archive onto the target system.
2. For the Java and ODBC/CLI drivers, extract the driver file into the chosen installation directory.
3. Install the driver that you need:
   - Java

     This driver provides support for client applications written in Java using JDBC. The path and file name of the Java driver are

     - Path: `jdbc_sqlj_driver/platform`
     - File name: `db2_db2driver_for_jdbc_sqlj.zip`

     For additional information, refer to "Installing IBM Data Server Driver for JDBC and SQLJ" on page 51.
   - ODBC and CLI

     The open source drivers included in the IBM Data Server Driver Package, such as PHP or Ruby driver, require the use of the Data Server CLI driver for connection to the database engine. If you plan to use any of these drivers, you need to install the ODBC driver.

     - Path: `odbc_cli_driver/platform`
     - File name: `ibm_data_server_driver_for_odbc_cli.tar.Z`

     For additional information, refer to "Installing IBM Data Server Driver for ODBC and CLI" on page 52.
   - PHP

     The IBM_DB2 and PDO_IBM are the PHP extensions for IBM Data Servers including Informix. Installing the IBM_DB2 or PDO_IBM extensions enables any application in the PHP environment to interact with Informix database servers. The extensions included with the IBM Data Server Driver Package might be of a lower version as compared to the one available on the PHP repository.

     Download the latest extension from PHP repository page because it might contain important fixes and new features:

     - For IBM_DB2

       http://pecl.php.net/package/ibm_db2
     - For PDO_IBM

       http://pecl.php.net/package/pdo_ibm

The PHP driver path and file names in the IBM Data Server Driver Packages are as follows:

- Path

  `php_driver/platform/php32 or php_driver/platform/php64`

- Files

  `ibm_db2_`*`n.n.n`*`.so and pdo_ibm_n.n.n.so`

  where *n* represents the version of the extension.

Consider the following information before you install the PHP driver:

- Prerequisite: The PHP drivers require the ODBC/CLI driver that is also included in the *IBM Data Server Driver Package* to be installed.
- Installation instructions: The following online information explains how to install PECL extensions.

  Through the source code:

  `http://www.php.net/manual/en/install.pecl.phpize.php`

  Through the PECL command:

  `http://www.php.net/manual/en/install.pecl.pear.php`

  Installing the IBM_DB2 extension:

  `http://www.php.net/manual/en/ibm-db2.installation.php`

  Installing the PDO_IBM extension:

  `http://www.php.net/pdo_ibm`

– Ruby on Rails

The IBM_DB adapter and driver as a gem enables any application in the Ruby environment, including Rails, to interact with IBM data servers. The path, file name, and installation information for Ruby on Rails Driver is as follows:

- Path

  `ruby_driver/platform`

- File

  `ibm_db-0.10.0.gem`

Consider the following information before you install the Ruby on Rails Driver:

- Prerequisite: The Ruby on Rails Driver requires the ODBC/CLI driver that is also included in the *IBM Data Server Driver Package* to be installed.
- To install the Ruby on Rails Driver, from the location of the gem file, run the following command:

  ```
  gem install ibm_db-0.10.0.gem
  ```

You are now ready to use any of these drivers on UNIX or Linux.

Installing the driver package on Windows

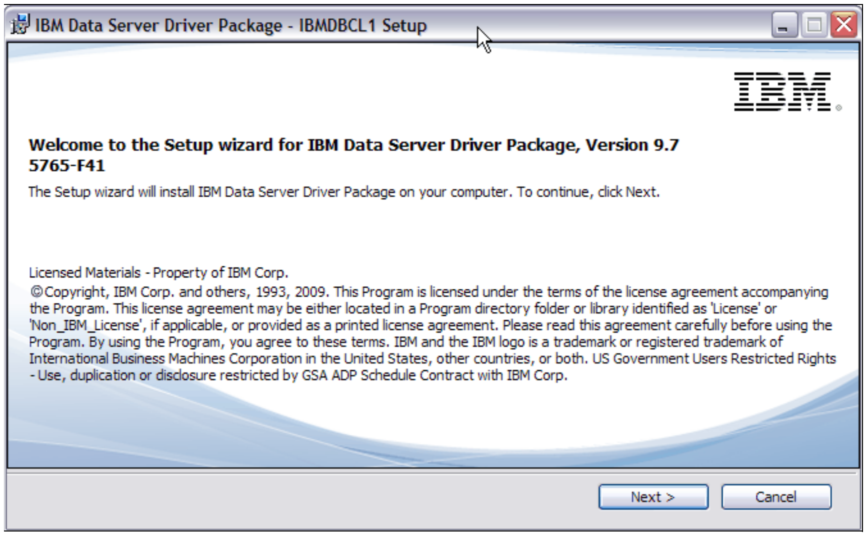
You can obtain the Data Server Driver Package for windows from the same site as the package for UNIX:

http://www.ibm.com/support/fixcentral/

The file for Windows is an `.exe` file, which is the installation file for the driver package.

To install the driver package:

1. Start the installation by executing the `.exe` file. The welcome window opens, as shown in Figure 2-11. Click **Next**.



IBM Data Server Driver Package - IBMDBCL1 Setup

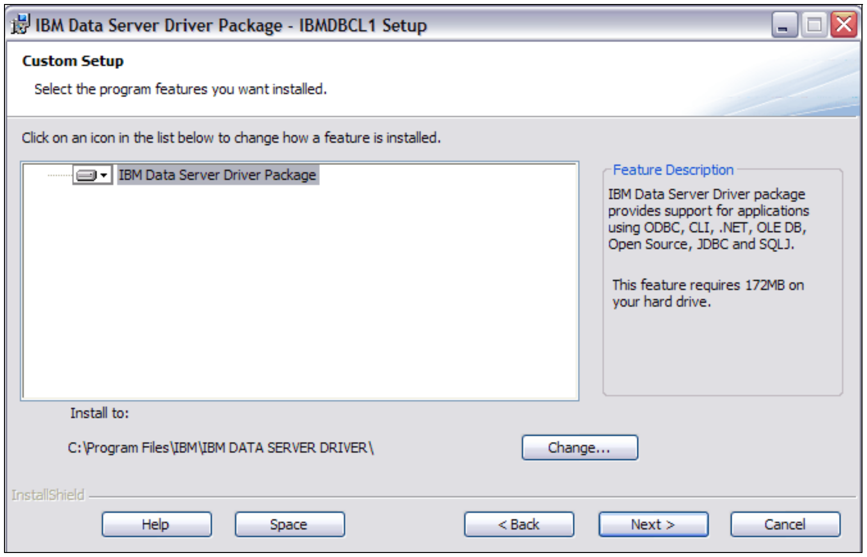
Welcome to the Setup wizard for IBM Data Server Driver Package, Version 9.7 5765-F41

The Setup wizard will install IBM Data Server Driver Package on your computer. To continue, click Next.

Licensed Materials - Property of IBM Corp.
©Copyright, IBM Corp. and others, 1993, 2009. This Program is licensed under the terms of the license agreement accompanying the Program. This license agreement may be either located in a Program directory folder or library identified as 'License' or 'Non_IBM_License', if applicable, or provided as a printed license agreement. Please read this agreement carefully before using the Program. By using the Program, you agree to these terms. IBM and the IBM logo is a trademark or registered trademark of International Business Machines Corporation in the United States, other countries, or both. US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Figure 2-11 Welcome panel for Data Server Driver Package

2. On the Software License Agreement panel, accept the license agreement, and click **Next**.
3. Specify the installation directory in the Custom Setup panel (Figure 2-12). You can change the installation directory to a location other then the default. Click **Next**.


IBM Data Server Driver Package - IBMDBCL1 Setup
Custom Setup
Select the program features you want installed.
Click on an icon in the list below to change how a feature is installed.
IBM Data Server Driver Package
Feature Description
Install to:
Change...
InstallShield
Help
Space
< Back
Next >
Cancel

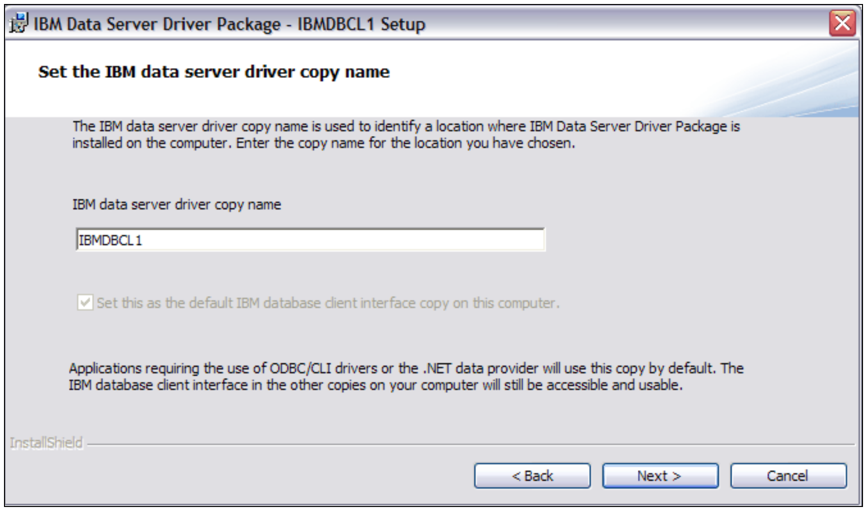
IBM Data Server Driver package provides support for applications using ODBC, CLI, .NET, OLE DB, Open Source, JDBC and SQLJ.

This feature requires 172MB on your hard drive.

C:\Program Files\IBM\IBM DATA SERVER DRIVER\

Figure 2-12 Change the installation directory

4. The IBM data server driver copy name is used to identify a location where IBM Data Server Driver Package is installed on the computer. Enter the copy name for the location that you have chosen, the default is IBDBCL1 (Figure 2-13). Click **Next**.


IBM Data Server Driver Package - IBMDBCL1 Setup

Set the IBM data server driver copy name

The IBM data server driver copy name is used to identify a location where IBM Data Server Driver Package is installed on the computer. Enter the copy name for the location you have chosen.

IBM data server driver copy name

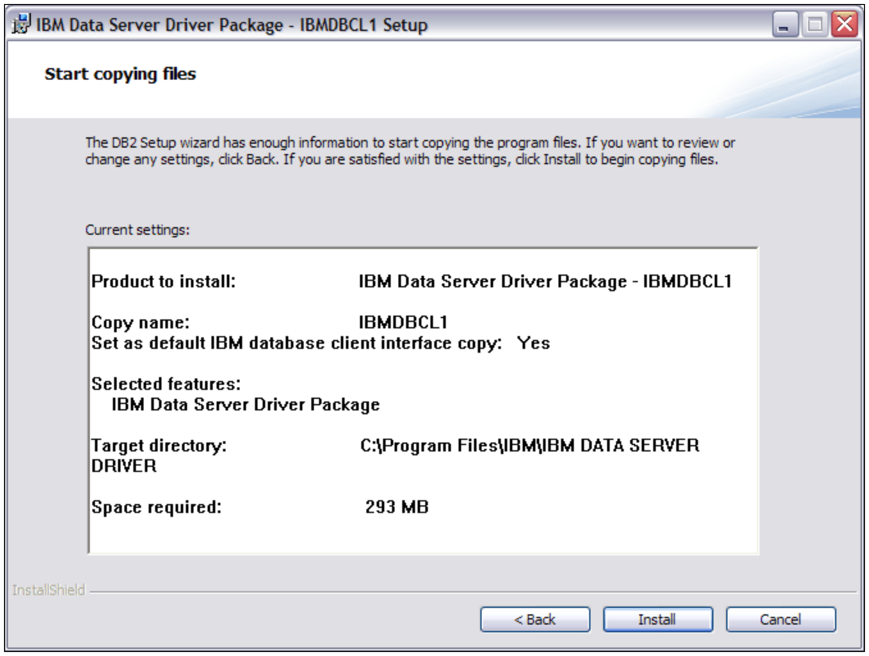
IBMDBCL1

☑ Set this as the default IBM database client interface copy on this computer.

Applications requiring the use of ODBC/CLI drivers or the .NET data provider will use this copy by default. The IBM database client interface in the other copies on your computer will still be accessible and usable.

Figure 2-13   Set the IBM data server driver copy name

5. Verify the installation selections on the summary panel (Figure 2-14). If everything is correct, begin the installation by clicking **Install**.



The DB2 Setup wizard has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Install to begin copying files.

Current settings:

Product to install: IBM Data Server Driver Package - IBMDBCL1

Copy name: IBMDBCL1
Set as default IBM database client interface copy: Yes

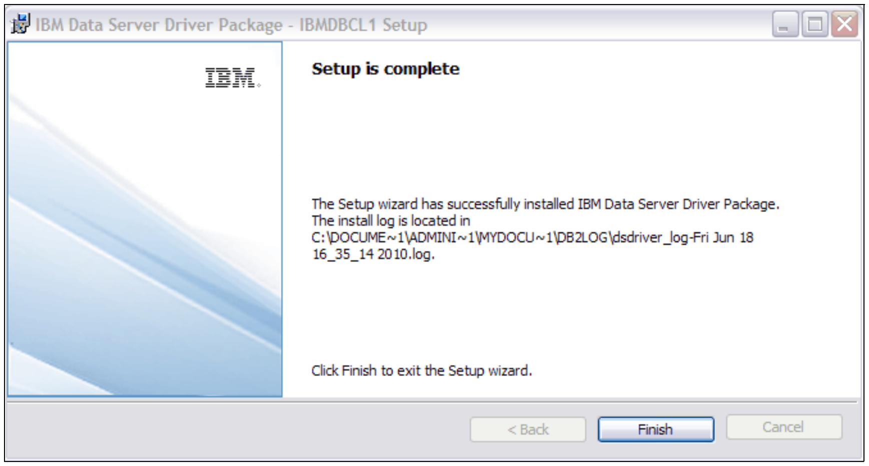
Selected features:
IBM Data Server Driver Package

Target directory: C:\Program Files\IBM\IBM DATA SERVER DRIVER

Space required: 293 MB

Figure 2-14 Summary of the installation

6. When the installation completes, the complete panel opens as shown in Figure 2-15. Click **Finish**. The installation of Data Server Driver for Informix is complete.



Setup is complete

The Setup wizard has successfully installed IBM Data Server Driver Package.
The install log is located in
C:\DOCUME~1\ADMINI~1\MYDOCU~1\DB2LOG\dsdriver_log-Fri Jun 18
16_35_14 2010.log.

Click Finish to exit the Setup wizard.

Figure 2-15 Installation complete

Note: The Windows installer of the *IBM Data Server Driver Package* does not provide any option to install individual components. All the drivers and interfaces that are included in the package are installed by default.

Installing IBM Data Server Driver for JDBC and SQLJ

To install IBM Data Server Driver for JDBC and SQLJ:

1. Obtain Java SDK.

   Before you install the IBM Data Server Driver for JDBC and SQLJ, you must have an SDK for Java installed on your computer:

   – For JDBC 3.0 functions, you need Java SDK 1.4.2 or later.
   – For JDBC 4.0 functions, you need Java SDK 6 or later.

2. Download the `.zip` file for the latest version of the IBM Data Server Driver for JDBC and SQLJ at:

   `http://www.ibm.com/software/data/support/data-server-clients/download.html`

3. Extract the IBM Data Server Driver for JDBC and SQLJ compressed file to the installation location. The compressed file contains the following files:
   - `db2jcc.jar`
   - `db2jcc4.jar`
   - `sqlj.zip`
   - `sqlj4.zip`
4. Modify the CLASSPATH environment variable to include the appropriate files, You can set the CLASSPATH using the following command:

   ```
   java -classpath <dir>\<file>.jar
   ```

   - For JDBC:

     Include the `db2jcc.jar` file in the CLASSPATH if you plan to use the version of the IBM Data Server Driver for JDBC and SQLJ that includes only JDBC 3.0 and earlier functions.

     Include the `db2jcc4.jar` file in the CLASSPATH if you plan to use the version of the IBM Data Server Driver for JDBC and SQLJ that includes JDBC 4.0 and earlier functions.

     Important: Include either the `db2jcc.jar` or the `db2jcc4.jar` files in the CLASSPATH. Do not include both files.

   - For SQLJ

     The steps are similar to the JDBC except that the files to be included in CLASSPATH are `sqlj.zip` and `sqlj4.zip`.

For more information regarding configuration and customizing of the *IBM Data Server Driver for JDBC and SQLJ*, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.jccids.doc/ids_jcc_0008.htm

Installing IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI solution is designed mainly for independent software vendor (ISV) deployments.To install the IBM Data Server Driver for ODBC and CLI:

1. Depending on your operating system, download the compressed file that is available at:

   https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?land=en_US&source=swg-informixfpd
2. Extract the file to an installation directory of your choice.

3. On Mac OS X, set the DYLD_LIBRARY_PATH environment variable to the `clidriver/lib` directory path.

For specific information about the setup and configuration of the ODBC, refer to 3.2, “Setup and configuration” on page 58.

2.2.4 Setting up Informix JDBC

Informix JDBC is a platform independent, industry-standard driver that provides enhanced support for distributed transactions. It consists of a set of interfaces and classes written in the Java programming language which can run on AIX, HP-UX, Linux, Solaris, Windows, and all other platforms that support Java. It also supports extensibility with a user-defined type (UDT) routine manager that simplifies the creation and use of UDTs.

JCC support is available through the IBM Data Server Driver package that has a driver to run Java application using JCC.

This section describes how to install Informix JDBC Driver, which is bundled with the Informix database server. There is a separate JDBC directory with a `.jar` file inside when you extract the Informix database server from the installation media.

Informix JDBC Driver, Version 3.50, strives to be compliant with the Sun Microsystems JDBC 3.0 specification. Nearly all the features that are required for the Sun Microsystems JDBC 3.0 specification have the specified behavior. For the Sun Microsystems JDBC 3.0 driver optional features, if the feature is supported by IBM Informix Version 11.50, then it is supported by Informix JDBC Driver, Version 3.50.

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

Installing the JDBC driver

The JDBC driver installation procedure is same for both Windows and UNIX platforms.

To install the JDBC driver in console mode, use the following command:

```
java -cp dir/setup.jar run -console
```

For a silent installation on both Windows and UNIX, run the following command from a command prompt:

```
java -cp <dir>/setup.jar run -silent -P
product.installLocation=<destination-dir>
```

where:

- `<dir>` is the location of the `setup.jar` file
- `<destination-dir>` is the directory where you want to install the JDBC driver

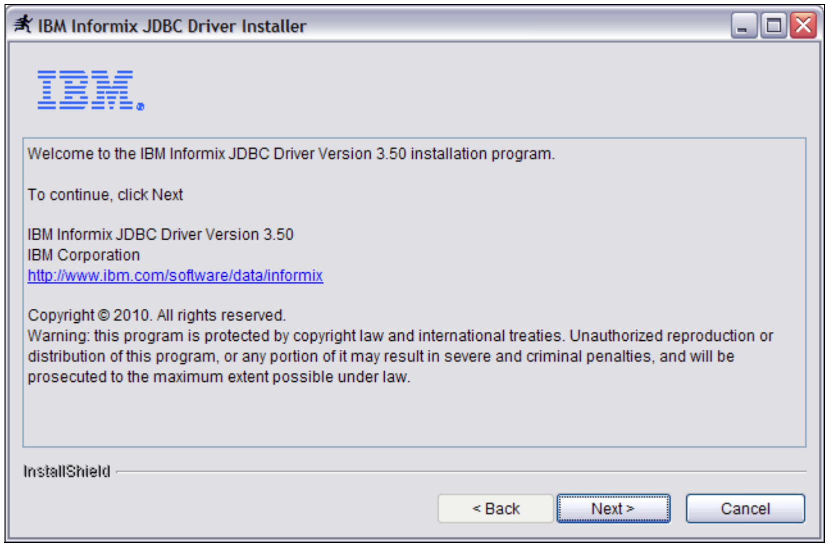
The installation is complete when the command finishes executing. If you want to log information during the installation, specify the `-log` parameter.

To install Informix JDBC Driver in GUI mode on Windows:

1. Start the GUI mode installation with the following command:

   ```
   java -cp dir/setup.jar run
   ```

   The Welcome panel opens (Figure 2-16). Click **Next**.



Welcome to the IBM Informix JDBC Driver Version 3.50 installation program.

To continue, click Next

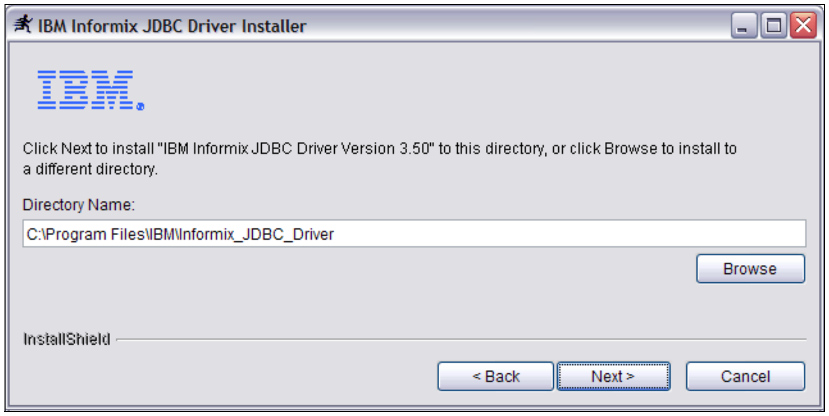
IBM Informix JDBC Driver Version 3.50
IBM Corporation
http://www.ibm.com/software/data/informix

Copyright © 2010. All rights reserved.
Warning: this program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it may result in severe and criminal penalties, and will be prosecuted to the maximum extent possible under law.

Figure 2-16 Starting panel for JDBC informix installation

2. Accept the license agreement, and click **Next**.

3. Specify the installation directory or use the default location as shown in Figure 2-17, and click **Next**.


IBM Informix JDBC Driver Installer

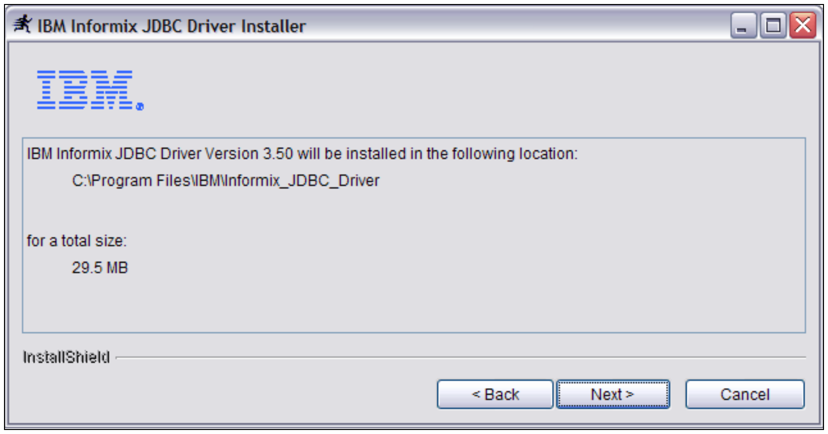
Click Next to install "IBM Informix JDBC Driver Version 3.50" to this directory, or click Browse to install to a different directory.

Directory Name:

C:\Program Files\IBM\Informix_JDBC_Driver

Figure 2-17 Specify the installation directory

4. Review the summary information that displays (Figure 2-18), and click **Next** complete the installation.


IBM Informix JDBC Driver Installer

IBM Informix JDBC Driver Version 3.50 will be installed in the following location:

C:\Program Files\IBM\Informix_JDBC_Driver

for a total size:

29.5 MB

Figure 2-18 Summary window

Configuring the JDBC driver

To use JDBC driver in an application, you must set the CLASSPATH environment variable to point to the driver files. The CLASSPATH environment variable tells the JVM and other applications where to find the Java class libraries that are used in a Java program-log parameter.

UNIX

Use one of the following methods to set the CLASSPATH environment variable on a UNIX system:

- Add the full path name of the `ifxjdbc.jar` file to CLASSPATH:

  ```
  setenv CLASSPATH /jdbcdriv/lib/ifxjdbc.jar:$CLASSPATH
  ```

- Extract the `ifxjdbc.jar` file, and add its directory to CLASSPATH:

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

Windows system

Use one of the following methods to set the CLASSPATH environment variable on a Windows system:

- Add the full path name of the `ifxjdbc.jar` file to CLASSPATH:

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

- Extract the `ifxjdbc.jar` file, and add its directory to CLASSPATH:

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

You are now ready to develop a Java application using JDBC.

3

Working with the ODBC driver

This chapter discusses the configuration and development of applications using Open Database Connectivity (ODBC) interfaces to access an Informix database server.

This chapter includes the following topics:

- ODBC and Informix
- Setup and configuration
- Developing an ODBC application

3.1 ODBC and Informix

ODBC is a software API method based on the X/Open Call Level Interface to access data from a database management system. ODBC allows developers to write database applications without having to know a proprietary interface for a specific database. In an ODBC environment, the ODBC driver is the component that directly communicates with the database server. It receives calls from the application or the ODBC Driver Manager and converts those calls into messages that the database server can understand.

The following ODBC drivers for Informix are available:

- IBM Informix ODBC Driver
- IBM Data Server Driver for ODBC and CLI (CLI Driver)

3.2 Setup and configuration

In this section, we discuss the setup and configuration parameters of the ODBC driver.

3.2.1 IBM Informix ODBC Driver

The Informix ODBC Driver is installed by default as part of Informix Client Software Development Kit (Client SDK). It is based on the Microsoft Open Database Connectivity (ODBC) Version 3.0 standard.

Table 3-1 lists the Informix database server that support Informix ODBC.

Table 3-1 Supported databases

Database server	Versions
IBM Informix	10.0,11.10,11.50
IBM Informix Extended Parallel Server	8.50 and higher
IBM Informix Standard Engine	7.25
IBM Informix Online	5.20 and higher

Windows system configuration

On a Windows system, the ODBC drivers is registered automatically within the system during the Informix installation process.

The following directory is the default installation directory:

```
C:\Program Files\IBM\Informix\Client-SDK
```

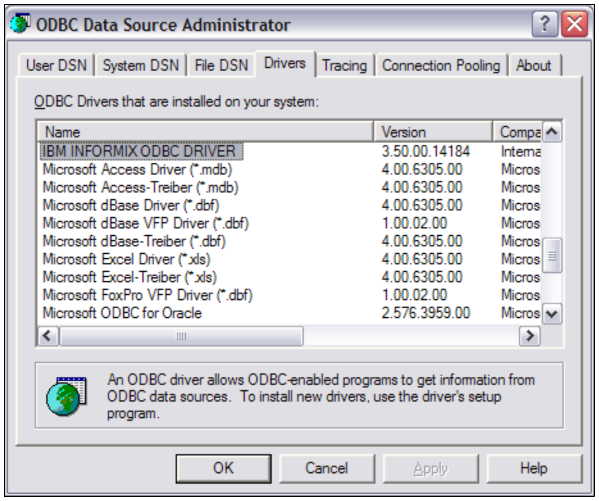
The INFORMIXDIR environment variable points to the directory where the product was installed.

Note: If you install a 32-bit version of Client SDK on a Windows x64, the default directory is `C:\Program Files (x86)\IBM\Informix\Client-SDK`.

The ODBC driver requires that additional libraries are loaded. These libraries are located in the `%INFORMIXDIR%\bin` directory. Make sure that this directory is part of your PATH variable.

The name of the ODBC driver depends on the version:

- IBM Informix ODBC Driver: For the Windows 32-bit driver, see Figure 3-1.



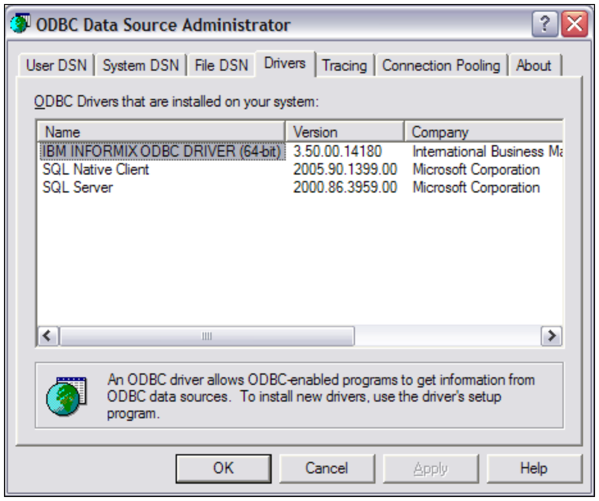
ODBC Drivers that are installed on your system:

Name	Version	Compa
IBM INFORMIX ODBC DRIVER	3.50.00.14184	Interna
Microsoft Access Driver (*.mdb)	4.00.6305.00	Micros
Microsoft Access-Treiber (*.mdb)	4.00.6305.00	Micros
Microsoft dBase Driver (*.dbf)	4.00.6305.00	Micros
Microsoft dBase VFP Driver (*.dbf)	1.00.02.00	Micros
Microsoft dBase-Treiber (*.dbf)	4.00.6305.00	Micros
Microsoft Excel Driver (*.xls)	4.00.6305.00	Micros
Microsoft Excel-Treiber (*.xls)	4.00.6305.00	Micros
Microsoft FoxPro VFP Driver (*.dbf)	1.00.02.00	Micros
Microsoft ODBC for Oracle	2.576.3959.00	Micros

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

Figure 3-1 Windows x86 ODBC driver administrator

- IBM Informix ODBC Driver (64-bit): for the Windows 64-bit driver. See Figure 3-2.



ODBC Drivers that are installed on your system:

Name	Version	Company
IBM INFORMIX ODBC DRIVER (64-bit)	3.50.00.14180	International Business Ma
SQL Native Client	2005.90.1399.00	Microsoft Corporation
SQL Server	2000.86.3959.00	Microsoft Corporation

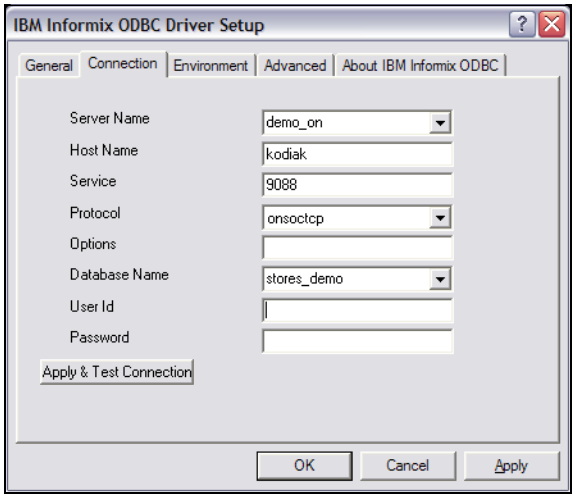
An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

Figure 3-2 Windows x64 ODBC driver administrator

To create an Informix ODBC Data Source (DSN), open the ODBC Administrator, choose a DSN type, and select the Informix ODBC driver, the DSN-configuration parameters are common to both drivers (32-bit and 64-bit).

Note: The default ODBC Data Source Administrator on a Windows x64 system is the 64-bit version. If you want to create a 32-bit ODBC DSN you must use the 32-bit version
`C:\WINDOWS\SysWOW64\odbcad32.exe`

Figure 3-3 shows the Informix ODBC Connection tab.


IBM Informix ODBC Driver Setup

Server Name: demo_on
Host Name: kodiak
Service: 9088
Protocol: onsoctcp
Options:
Database Name: stores_demo
User Id:
Password:

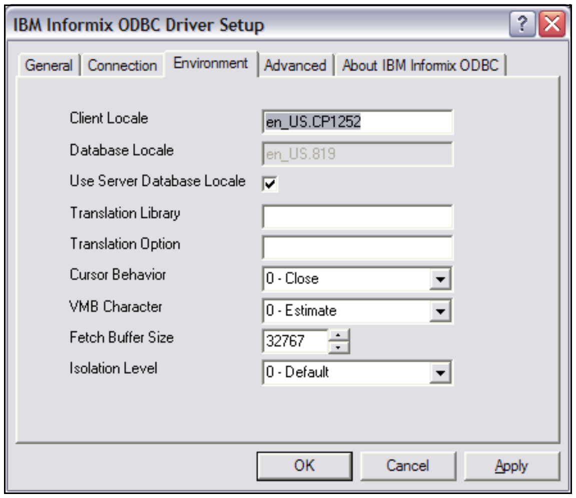
Figure 3-3 Connection parameters

Table 3-2 lists the required DSN parameters. If you have already defined an Informix server using the `setnet32.exe` utility, which is a component of Client SDK, most of these values are updated automatically when selecting an Informix server from the drop-down box.

Table 3-2 Required DSN values

Parameter	**Description**
Data source name	Name of the DSN
Server name	Name of the Informix database server
Host name	Name or IP address of the computer where the database server runs
Service	Name or value of the tcp service used by the database server
Protocol	Communication Protocol used by the database server
Database	Name of the database to which the DSN connects by default

You can set optional configuration parameters in the Environment and Advanced tabs (Figure 3-4).



Client Locale: en_US.CP1252
Database Locale: en_US.819
Use Server Database Locale: ✓
Translation Library:
Translation Option:
Cursor Behavior: 0 - Close
VMB Character: 0 - Estimate
Fetch Buffer Size: 32767
Isolation Level: 0 - Default

Figure 3-4 Environment parameters

Table 3-3 describe the parameters for the Environment tab.

Table 3-3 Environment parameters

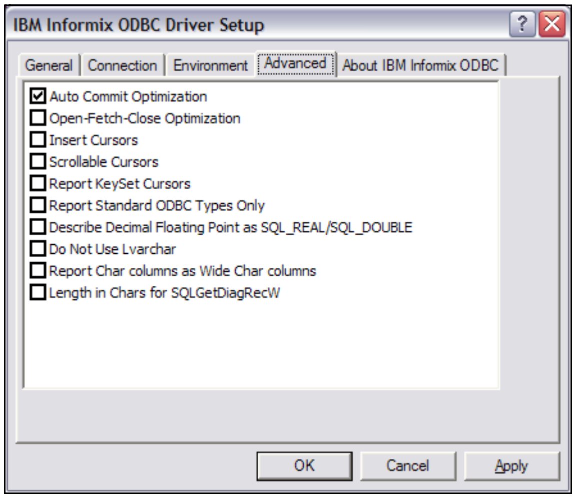
Parameter	Description
Client Locale	Locale for the client machine
Database Locale	Locale used when the database was created
Use Server Database Locale	Using this option the Database Locale is automatically retrieved from the database
Translation Library	Code set conversion library
Translation Option	Options for a third-party conversion library
Cursor Behavior	Close or preserve a cursor when a transaction is resolved
VMB Character	How to report the length for Varying Multi byte characters
Fetch Buffer Size	Size of the communication data package
Isolation Level	Default Isolation Level for the connection

Note: The code set value for the Client Locale parameter is determined by the code set used in the application, which in most cases is the same as the operating system. The default code set on a Windows system is *CP1252*.

The default code set for an Informix database is en_us.8859-1 (ISO 8859-1). Some versions of the IBM Informix database server (for example, Version 9.40 and Version 10.00) allow a connection from a client with a mismatched DB_LOCALE value. This option is no longer available. A client application must set the Database Locale parameter (DB_LOCALE) to the same value as the locale of the database (locale used at creation time).

The typical syntax for an Informix locale is `language_territory@codeset`.

Figure 3-5 shows the Advanced tab configuration parameters.



- [✓] Auto Commit Optimization
- [ ] Open-Fetch-Close Optimization
- [ ] Insert Cursors
- [ ] Scrollable Cursors
- [ ] Report KeySet Cursors
- [ ] Report Standard ODBC Types Only
- [ ] Describe Decimal Floating Point as SQL_REAL/SQL_DOUBLE
- [ ] Do Not Use Lvarchar
- [ ] Report Char columns as Wide Char columns
- [ ] Length in Chars for SQLGetDiagRecW

Figure 3-5 Advanced parameters

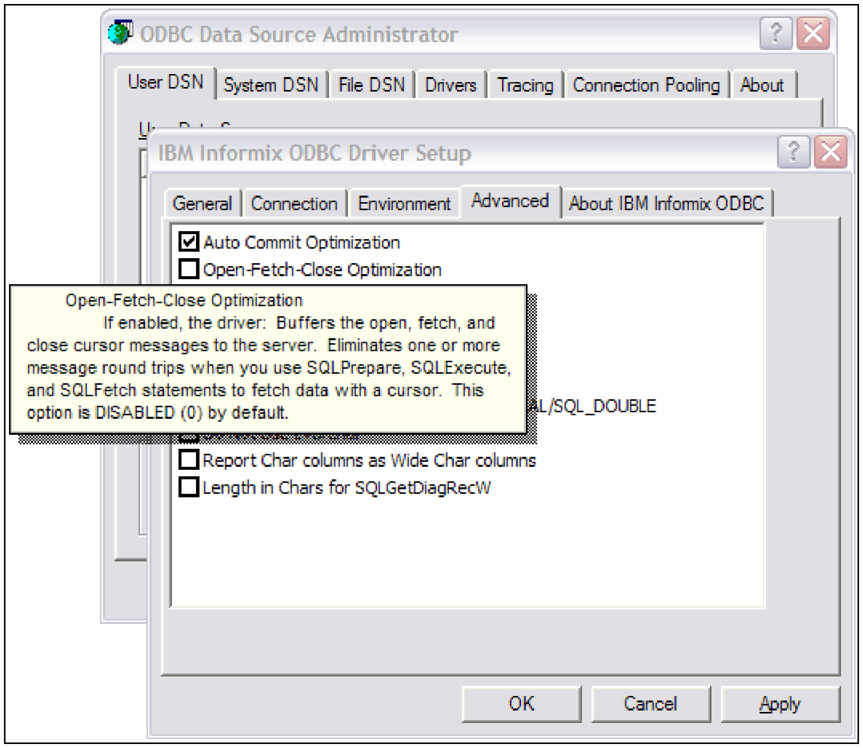
Table 3-4 describes the parameters for the Advanced tab.

Table 3-4 Advanced parameters

Parameter	**Description**
Auto Commit Optimization	Optimize client-server communication deferring the commit work after all the cursors are close.
Open-Fetch-Close Optimization	Optimize client-server communication automatically closing the cursor after all the rows have been retrieved.

Parameter	Description
Insert Cursors	Buffers all the rows send by an insert cursor and uses only one package to send the data to the server.
Scrollable Cursors	Enable scrollable cursors.
Report KeySet Cursors	Report support for KeySet-driver cursors.
Report Standard ODBC Types Only	Report standard ODBC types Only. Extended types are mapped to standard ODBC types.
Describe Decimal Floating Point as SQL_REAL / SQL_DOUBLE	Decimal columns without a scale are reported as SQL_REAL or SQL_DOUBLE.
Do Not Use Lvarchar	Do not report Lvarchar columns when using SQL_VARCHAR.
Report Char columns as Wide Char columns	Char columns described by SQLDescribeCol are reported as Wide char columns (SQLWCHAR)
length in Chars for SQLGetDiagRecW	Returns the number of characters rather than the number of bytes.

For more information about each one of these parameters, place the cursor over a parameter, and press F1 (Figure 3-6).



☑ Auto Commit Optimization
☐ Open-Fetch-Close Optimization

Open-Fetch-Close Optimization
If enabled, the driver: Buffers the open, fetch, and close cursor messages to the server. Eliminates one or more message round trips when you use SQLPrepare, SQLExecute, and SQLFetch statements to fetch data with a cursor. This option is DISABLED (0) by default.

AL/SQL_DOUBLE

☐ Report Char columns as Wide Char columns
☐ Length in Chars for SQLGetDiagRecW

Figure 3-6 ODBC DSN Help

UNIX configuration

On UNIX platforms, the default installation directory for Client SDK is the `/opt/IBM/informix` directory.

The INFORMIXDIR environment variable should point to the directory where the product was installed.

On UNIX platforms, an ODBC driver manager is not normally supplied as part of the operating system. Client SDK does not include an ODBC driver manager library, however, the Client SDK does have a Driver Manager Replacement (DMR) library which provides most of the features of an ODBC driver manager.

If the application does not use an ODBC driver manager, such as unixODBC or Data Direct Driver Manager, the application must be linked directly to the Informix ODBC libraries. The ODBC libraries are located in the `$INFORMIXDIR/lib/cli` directory.

Table 3-5 shows the ODBC libraries that are included with Client SDK.

Table 3-5 UNIX ODBC libraries

Library	Description
libifcli.a or libcli.a	Static version for single (nonthreaded)
libifcli.so or iclis09b.so	Shared version for single (nonthreaded)
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)

The shared-library path environment variable specifies the library search path. This variable should contains at least `$INFORMIX/lib`, `$INFORMIXDIR/lib/esql`, and `$INFORMIXDIR/lib/cli` for the ODBC driver to work.

There are three configuration files for the ODBC driver:

- `sqlhosts`
- `odbc.ini`
- `odbcinst.ini`

Note: The information in the `sqlhosts` file is also used by all the other Client SDK components and by the Informix database server. Be aware that any changes in this file can have an impact in other clients and servers.

On a Windows system, instead of using a text file, this information is stored in the registry through the `setnet32.exe` utility.

The sqlhosts file

This text file contains most of the information that is required to connect to an IBM Informix database server.

The default location for this file is the `$INFORMIXDIR/etc/sqlhosts` directory. You can use the INFORMIXSQLHOSTS environment variable to point to a different location.

Example 3-1 shows a simple `sqlhosts` file.

Example 3-1 The sqlhosts file

```
#server protocol hostname service/port
on_demo onsoctcp kodiak 9088
```

For more information about the `sqlhosts` file, refer to the Client/Server Communications manual at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.admin.doc/ids_admin_0158.htm

The odbcinst.ini file

This file contains a list of installed ODBC drivers on the UNIX system and specific attributes for each driver, such as the location of the shared library.

The default location of this file is under the home directory as `$HOME/.odbcinst.ini`. A sample `odbcinst.ini` file is located in the `$INFORMIXDIR/etc` directory.

Example 3-2 shows a simple `odbcinst.ini` file.

Example 3-2 The odbcinst.ini file

```
;---------------------------------------------------------------------------
[ODBC Drivers]
IBM INFORMIX ODBC DRIVER=Installed
[IBM INFORMIX ODBC DRIVER]
Driver=/extra/informix/lib/cli/iclit09b.so
Setup=/extra/informix/lib/cli/iclit09b.so
APILevel=1
ConnectFunctions=YYY
DriverODBCVer=03.51
FileUsage=0
SQLLevel=1
smProcessPerConnect=Y
```

The `odbinst.ini` file has two sections:

- ODBC Drivers: Lists the ODBC drivers installed in the system.
- Driver Properties: Lists the properties for a specific ODBC driver.

Table 3-6 lists the parameters in the Driver Properties section.

Table 3-6 Parameters in the Driver Properties section

Keyword	Description
Driver	Location of the ODBC library
Setup	Location of the Setup library
APILevel	ODBC interface conformance level that the driver supports
ConnectFunctions	Support for SQLConnect, SQLDriverConnect, and SQLBrowserConnect

Keyword	Description
DriverODBCVer	Supported version of the ODBC driver
FileUsage	How handle File-system DSN
SQLLevel	Type of SQL-92 grammar supported

The odbc.ini file

The `odbc.ini` file is the data source configuration information. The default location of this file is under the home directory as `$HOME/.odbc.ini`.

Alternatively, you can use the ODBCINI environment variable to point the `odbc.ini` to a different location. A sample `odbc.ini` file is located in the `$INFORMIXDIR/etc` directory.

Example 3-3 shows a simple `odbc.ini` file.

Example 3-3 The odbc.ini file

```
[ODBC Data Sources]
Infdrv1=IBM INFORMIX ODBC DRIVER
;
[Infdrv1]
Driver=/extra/informix/lib/cli/iclis09b.so
Description=IBM INFORMIX ODBC DRIVER
Database=stores_demo
LogonID=odbc
pwd=odbc
Servername=demo_on
CursorBehavior=0
CLIENT_LOCALE=en_us.8859-1
DB_LOCALE=en_us.8859-1
TRANSLATIONDLL=/extra/informix/lib/esql/igo4a304.so
;
[ODBC]
;UNICODE=UCS-4
;
; Trace file Section
;
Trace=0
TraceFile=/tmp/odbctrace.out
InstallDir=/extra/informix
TRACEDLL=idmrs09a.so
```

This file contains three sections:

- ODBC Data Sources: DSN name and description of the driver used
- Data Source Specification: Configuration parameters for this particular DSN
- ODBC: Global options such as Unicode mode or Tracing

Table 3-7 lists the available parameters for the `odbc.ini` configuration file.

Table 3-7 Parameters for the odbc.ini file

Keyword	**Description**
Driver	Location of the ODBC shared library
Description	DSN description
Database	Database name
LogonID	User Id
pwd	Password
Server	Database server
CLIENT_LOCALE	Locale used by the client application
DB_LOCALE	Locale of the database
TRANSLATIONDLL	Location of the code set conversion library
CURSORBEHAVIOR	Close or preserve a cursor when a transaction is resolved
DefaultUDTFetchType	Default type for a UDT (SQL_C_BINARY or SQL_C_CHAR)
ENABLESCROLLABLECURSORS	Enable Scrollable cursors
ENABLEINSERTCURSORS	Optimize insert cursors process
OPTIMIZEAUTOCOMMIT	Defer the commit message after all cursor are closed
NEEDODBCTYPESONLY	Extended types are mapped to standard ODBC types
OPTOFC	Close cursors after all the rows have been fetched
REPORTKEYSETCURSORS	Report support for keyset-driven cursors
FETCHBUFFERSIZE	Set the size of the fetch buffer

Keyword	Description
DESCRIBEDECIMALFLOATPOINT	Describe float types as SQL_REAL or SQL_DOUBLE
USESERVERDBLOCALE	Use the server database locale
DONOTUSELVARCHAR	Don't report Lvarchar columns when using SQL_VARCHAR
REPORTCHARCOLASWIDECHARCOL	Char columns described by SQLDescribeCol are reported as Wide char columns (SQLWCHAR)
ISOLATIONLEVEL	Default isolation level
UNICODE	Type of Unicode (UCS-2, UCS-4)
TRACE	Trace enable or disable
TRACEFILE	Location of the trace file
TRACEDLL	Trace library name (idmrs09a.so)

3.2.2 IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI (CLI Driver) is installed as part of the IBM Data Server Driver Package or as a separate product. The IBM Data Server Package is bundle with Client SDK. It allows you to connect to IBM Informix and IBM DB2 databases with the same set of libraries.

The communication protocol used is DRDA instead of the native Informix SQLI. This driver is supported against Informix Version 11.10 and Version 11.50.

Windows system configuration

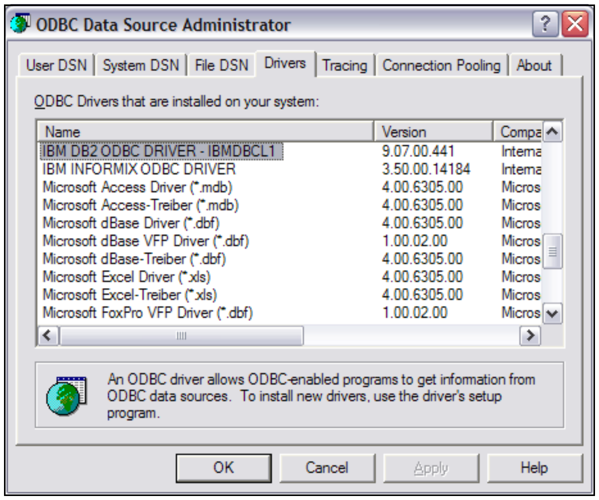
The IBM Data Server Driver installs in the following directory by default:

```
C:\Program Files\IBM\IBM DATA SERVER DRIVER
```

The ODBC driver is registered in the Windows system during the installation of the IBM Data Server Driver package. The name of the ODBC driver is *IBM DB2 ODBC Driver*.

Figure 3-7 shows the Drivers tab of the ODBC Data Source Administrator, which lists both ODBC drivers:

- IBM DB2 ODBC DRIVER
- IBM INFORMIX ODBC DRIVER



ODBC Drivers that are installed on your system:

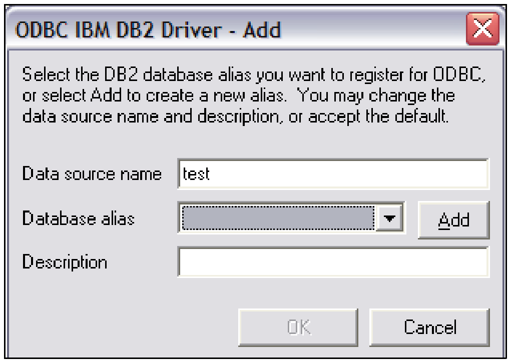
Name	Version	Compa
IBM DB2 ODBC DRIVER - IBMDBCL1	9.07.00.441	Interna
IBM INFORMIX ODBC DRIVER	3.50.00.14184	Interna
Microsoft Access Driver (*.mdb)	4.00.6305.00	Micros
Microsoft Access-Treiber (*.mdb)	4.00.6305.00	Micros
Microsoft dBase Driver (*.dbf)	4.00.6305.00	Micros
Microsoft dBase VFP Driver (*.dbf)	1.00.02.00	Micros
Microsoft dBase-Treiber (*.dbf)	4.00.6305.00	Micros
Microsoft Excel Driver (*.xls)	4.00.6305.00	Micros
Microsoft Excel-Treiber (*.xls)	4.00.6305.00	Micros
Microsoft FoxPro VFP Driver (*.dbf)	1.00.02.00	Micros

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

Figure 3-7 ODBC Data Source Administrator

To create an ODBC Data Source (DSN) using the IBM Data Server Driver, open the ODBC Administrator, choose a DSN type, and select the IBM DB2 ODBC Driver.

Figure 3-8 shows the Add dialog box.



Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name: test

Database alias:

Description:

Figure 3-8 DSN Add dialog box

If there is no database alias defined, you must add one with the connection details of your IBM Informix database server (Figure 3-9).

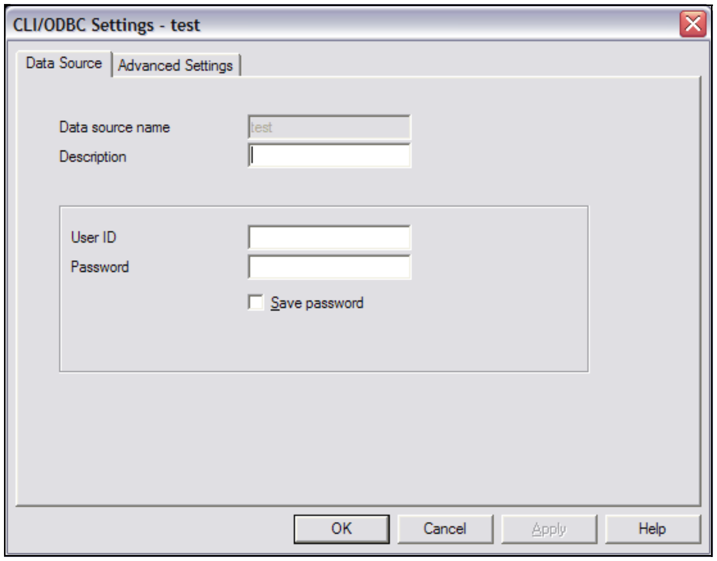
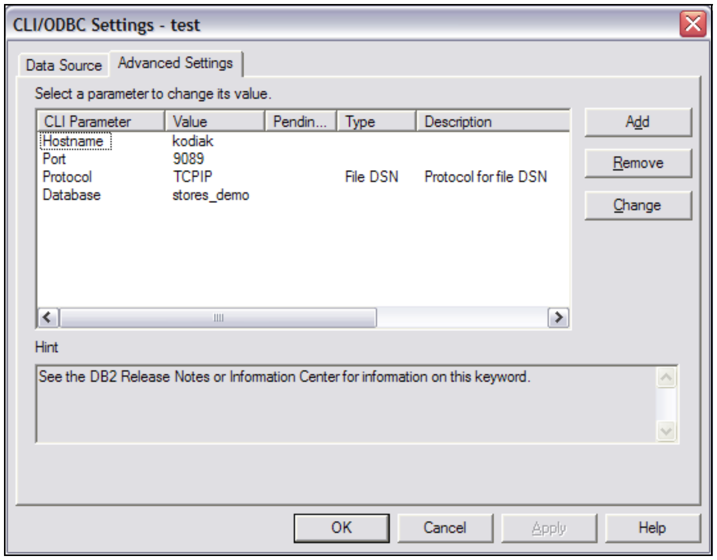


Figure 3-9 DSN Settings dialog box

Figure 3-10 shows the Advanced Settings tab.



Select a parameter to change its value.

CLI Parameter	Value	Pendin...	Type	Description
Hostname	kodiak			
Port	9089			
Protocol	TCPIP		File DSN	Protocol for file DSN
Database	stores_demo			

See the DB2 Release Notes or Information Center for information on this keyword.

Figure 3-10 DSN Advanced Settings dialog box

Table 3-8 lists the minimum required parameters to connect to an Informix server.

Table 3-8 CLI parameters

CLI parameter	Description
Hostname	Name of the system where the Informix server is running
Port	TCP Port used by the DRDA Informix alias
Protocol	Transport protocol (TCP/IP)
Database	Database server

The configuration settings are stored as a File-DSN (a text file that contains all the information that is required to connect to the database) in the `%USERPROFILE%\db2cli.ini` file. Example 3-4 shows a `db2cli.ini` file.

Example 3-4 The db2cli.ini file

```
[test]
Database=stores_demo
Protocol=TCPIP
Port=9089
Hostname=kodiak
```

For a complete description of all the CLI parameters, refer to the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.apdv.cli.doc/doc/r0007964.html

UNIX configuration

On UNIX platforms, the default installation directory for IBM Data Server Client package is `/opt/IBM/db2/V9.7`. The library for the ODBC driver is `libdb2.a`, which is located in the `lib` directory of your Data Server installation.

Similar to the installation on a Windows system, the ODBC configuration is stored in the `db2cli.ini` file, which by default is included in the `cfg` directory of your Data Server installation or is defined by the DB2CLIINI environment variable

Example 3-5 shows a simple `db2cli.ini` file.

Example 3-5 The db2cli.ini file

```
[test]
Database=stores_demo
Protocol=TCPIP
Port=9089
Hostname=kodiak
```

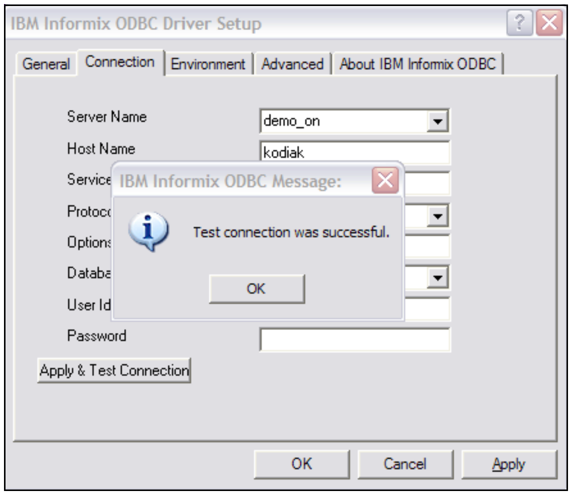
For more information about the `db2cli.ini` file, refer to:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.apdv.cli.doc/doc/c0007882.html

3.2.3 Verifying connectivity

On a Windows system, you can use the ODBC Data Source Administrator to verify the connection with the Informix database server.

When using the IBM Informix ODBC driver, you can verify the connection using the Apply & Test Connection option, as shown in Figure 3-11.



Server Name: demo_on
Host Name: kodiak

Test connection was successful.

Figure 3-11 Informix ODBC connection test

With the IBM Data Server Driver for ODBC, you can perform the connection test using the Connect option, as shown in Figure 3-12.

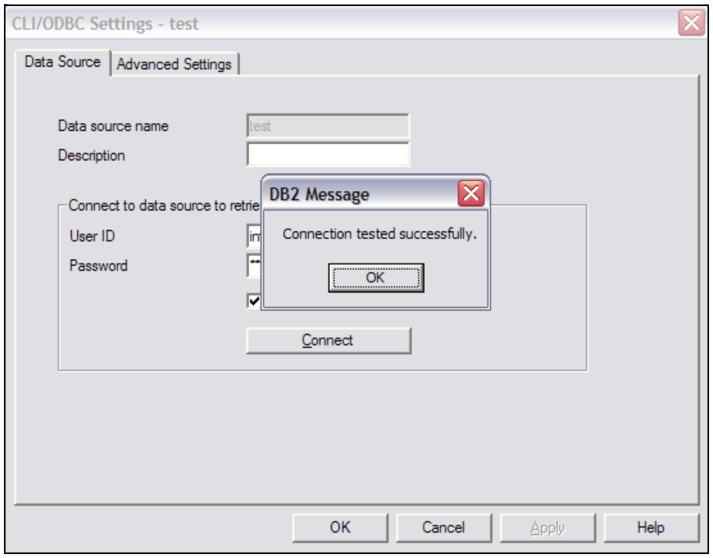


Figure 3-12 IBM Data Server Driver connection test

Client SDK on UNIX does not include any tool to test an ODBC DSN. You can use the C samples in `$INFORMIXDIR\demo\cli` to check whether the ODBC DSN is working.

3.3 Developing an ODBC application

In this section, we describe how to connect to a database server, type mapping between standard ODBC and IBM Informix, and how to perform basic operations with the ODBC driver.

3.3.1 Connecting to the database

Typically, an ODBC application performs the following steps:

1. Connects to the database.

   To connect to the database, the application must pass the connection details to the ODBC driver. These details can be included directly in the connection string or stored as a Data Source Name (DSN).

2. Processes SQL statements.

The application sends SQL request to the ODBC driver to perform database operations (insert, select, delete, update, and so on).

3. Resolves any open transaction.

   If the application is using a transaction, it resolves (commits or rolls back) any open transactions.

4. Terminates the connection, and frees the allocated resources.

These steps are done using the ODBC API functions. Figure 3-13 shows a typical sequence of calls. The labels inside the gray boxes correspond with the ODBC API functions that are used.

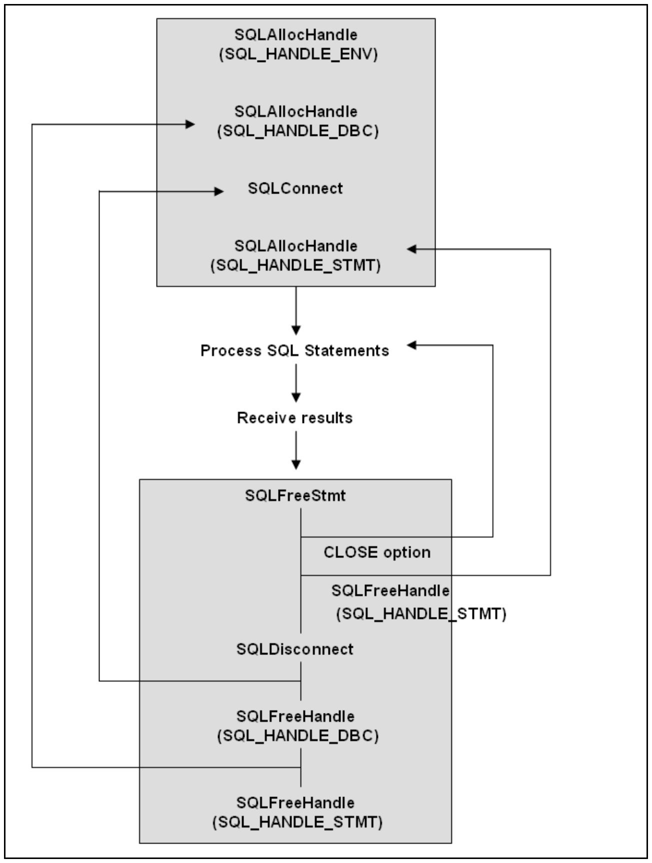

SQLAllocHandle (SQL_HANDLE_ENV)
SQLAllocHandle (SQL_HANDLE_DBC)
SQLConnect
SQLAllocHandle (SQL_HANDLE_STMT)
Process SQL Statements
Receive results
SQLFreeStmt
CLOSE option
SQLFreeHandle (SQL_HANDLE_STMT)
SQLDisconnect
SQLFreeHandle (SQL_HANDLE_DBC)
SQLFreeHandle (SQL_HANDLE_STMT)

Figure 3-13 Typical ODBC calls used by applications

Example 3-6 shows a simple C program that connects to an ODBC DSN.

Example 3-6 A test_connect.c sample

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;

    if (argc != 2)
    {
        fprintf (stdout, "Please specify the name of a DSN!\n");
    return(1);
    }
    else
    {
    if (strstr (argv[1],"DRIVER")==NULL)
       sprintf((char *) connStrIn, "DSN=%s;", (char *)argv[1]);
    else
       sprintf((char *) connStrIn, "%s;", (char *)argv[1]);
    }

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

    rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "SQLSetEnvAttr failed!\n");
        return (1);
    }

    rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection Handle Allocation failed!\n");
        return (1);
    }

    rc = SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, 1000,
                           &connStrOutLen, SQL_DRIVER_NOPROMPT);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connectedion failed!\n");
        return (1);
    }

    fprintf (stdout, "Connected\n");

    SQLDisconnect (hdbc);

    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

You can compile this code on a Windows system using the following command:

```
cl /DWIN32 -I%INFORMIXDIR%\incl\cli odbc32.lib test_connect.c
```

Example 3-7 shows how to compile and run the sample.

Example 3-7 The test_connec.c sample output

```
c:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -I%INFORMIXDIR%\incl\cli
odbc32.lib /nologo test_connect.c
test_connect.c

D:\work>test_connect
Please specify the name of a DSN!

D:\workt>test_connect dummy_dsn
Connection failed!

D:\work>test_connect test
Connected

D:\work>
```

You can use the sample C program to perform a DSN-less connection. Example 3-8 shows the sample C program with a DSN-less connection string using the Informix ODBC driver and the Data Server driver for ODBC.

Example 3-8 A DSN-less connection

```
C:\work>test_connect "DRIVER={IBM INFORMIX ODBC
DRIVER};SERVER=demo_on;DATABASE=stores_demo;HOST=kodiak;PROTOCOL=onsoctcp;SERVI
CE=9088;UID=informix;PWD=password;";
Connected

C:\work>test_connect "driver={IBM DB2 ODBC
DRIVER};Database=stores_demo;hostname=kodiak;port=9089;protocol=TCPIP;
uid=informix; pwd=password"
Connected
C:\work>
```

3.3.2 Type mapping

The data types that are used on the database differ from the data types that are used by your application. This section shows the data type mapping that is needed when working with specific Informix data types with the Informix ODBC driver.

When a query is executed by the application, the data returned by the Informix server might be in a different format than what the application uses. The ODBC driver converts the data that is passed between the application and the database server. This process is invisible to the application. The only requirement for the application is to specify the correct data types when calling the ODBC driver functions.

The application sets ValueType and ParameterType when calling the bind functions.

Example 3-9 shows the definition for one of the bind functions, `SQLBindParameter()`. The ODBC driver converts the data type specified in ValueType to the data type specified in ParameterType.

Example 3-9 SQLBindParameter definition

```
SQLBindParameter(
   SQLHSTMT         StatementHandle,     /* hstmt */
   SQLUSMALLINT     ParameterNumber,     /* ipar */
   SQLSMALLINT      InputOutputType,     /* fParamType */
   SQLSMALLINT      ValueType,           /* fCType */
   SQLSMALLINT      ParameterType,       /* fSqlType */
   SQLUINTEGER      ColumnSize,          /* cbColDef */
   SQLSMALLINT       DecimalDigits,       /* ibScale */
   SQLPOINTER        ParameterValuePtr,   /* rgbValue */
   SQLINTEGER        BufferLength,        /* cbValueMax */
   SQLINTEGER        *StrLen_or_IndPtr);  /* pcbValue */
```

Example 3-10shows an `SQLBindParameter()` call where the application is binding an SQL_C_BINARY parameter with an SQL_INFX_UDT_FIXED value.

Example 3-10 Using an SQLBindParameter() call

```
SQLBindParameter (
    hstmt,
    1,
    SQL_PARAM_INPUT,
    SQL_C_BINARY,
    SQL_INFX_UDT_FIXED,
    loptr_size,
    0,
    loptr_buffer,
    loptr_size,
    &loptr_valsize);
```

Table 3-9 shows the Informix-specific data type mapping that is used with the Informix ODBC driver.

Table 3-9 Mapping for specific Informix data types

Informix SQL	Informix ODBC driver
BIGINT	SQL_INFX_BIGINT
BIGSERIAL	SQL_INFX_BIGINT
BLOB	SQL_IFMX_UDT_BLOB
BOOLEAN	SQL_BIT
BYTE	SQL_LONGVARBINARY
CLOB	SQL_IFMX_UDT_CLOB
DATETIME	SQL_TIMESTAMP
DISTINCT	Any
IDSSECURITYLABEL	Built-in DISTINCT OF VARCHAR(128)
INT8	SQL_BIGINT
INTERVAL DAY	SQL_INTERVAL_DAY

Informix SQL	Informix ODBC driver
INTERVAL DAY TO HOUR	SQL_INTERVAL_DAY_TO_HOUR
INTERVAL DAY TO MINUTE	SQL_INTERVAL_DAY_TO_MINUTE
INTERVAL DAY TO SECOND	SQL_INTERVAL_DAY_TO_SECOND
INTERVAL HOUR	SQL_INTERVAL_HOUR
INTERVAL HOUR TO MINUTE	SQL_INTERVAL_HOUR_TO_MINUTE
INTERVAL HOUR TO SECOND	SQL_INTERVAL_HOUR_TO_SECOND
INTERVAL MINUTE	SQL_INTERVAL_MINUTE
INTERVAL MINUTE TO SECOND	SQL_INTERVAL_MINUTE _TO_SECOND
INTERVAL MONTH	SQL_INTERVAL_MONTH
INTERVAL SECOND	SQL_INTERVAL_SECOND
INTERVAL YEAR	SQL_INTERVAL_YEAR
INTERVAL YEAR TO MONTH	SQL_INTERVAL_YEAR_TO_MONTH
LIST, MULTISET, SET	Any
LVARCHAR	SQL_VARCHAR
MONEY	SQL_DECIMAL
NCHAR	SQL_CHAR
NVARCHAR	SQL_VARCHAR
OPAQUE (fixed)	SQL_INFX_UDT_FIXED
OPAQUE (varying)	SQL_INFX_UDT_VARYING
ROW	Any
SERIAL	SQL_INTEGER
SERIAL8	SQL_BIGINT
TEXT	SQL_LONGVARCHAR

For a complete list of Data Type mapping, refer to the *IBM Informix ODBC Driver Programmer's Manual*, which is available at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.odbc.doc/sii04979050.htm#sii04979050

If your application requires only Standard ODBC data types, you can enable the Report Standard ODBC Types option in your DSN or connection string. When this option is enabled, the ODBC driver handles smart large objects (BLOB and CLOB) as though they were simple large objects (byte and text). The driver generates the smart-large-object calls (`ifx_lo_open`, `ifx_lo_write`, and so on) automatically.

This option changes the mapping for user define types (including MULTISET, SET, ROW, and LIST) to SQL_C_CHAR.

These options are also available as the following ODBC attributes:

- SQL_INFX_ATTR_ODBC_TYPES_ONLY
- SQL_INFX_ATTR_LO_AUTOMATIC
- SQL_INFX_ATTR_DEFAULT_UDT_FETCH_TYPE

3.3.3 Performing database operations

In this section, we show samples of how to use the ODBC driver to perform basic operations with an Informix database. Most of these tasks are common to any ODBC application, so we do not explain them in detail.

This section includes the following topics:

- Simple SQL statements
- Fetching data
- Using parameters
- Calling SQL routines
- Local transactions
- Distributed transactions

Simple SQL statements

You can run SQL statements directly using the `SQLExecDirect()` function. This function is commonly used when there is no need to run the SQL statement more than once. The ODBC driver prepares the statement, executes it, and frees the resources in one operation.

Example 3-11 shows a sample of using the `SQLExecDirect()` function to set the LOCK WAIT time out.

Example 3-11 SQLExecDirect() sample, Simple_sql.c

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

#ifdef WIN32
```

```
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;

    int             sqllen;
    SQLCHAR         *sqlstmt;

    if (argc != 3)
    {
        fprintf (stdout, "Please specify the name of a DSN and the SQL Statement to
run!\n");
    return(1);
    }
    else
    {
    if (strstr (argv[1],"DRIVER")==NULL)
       sprintf((char *) connStrIn, "DSN=%s;", (char *)argv[1]);
    else
       sprintf((char *) connStrIn, "%s;", (char *)argv[1]);


        sqllen  = strlen((char *)argv[2]);
        sqlstmt = (SQLCHAR *) malloc (sqllen + sizeof(char));
        strcpy((char *)sqlstmt, (char *)argv[2]);

    }

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

    rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "SQLSetEnvAttr failed!\n");
        return (1);
    }
    rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection Handle Allocation failed!\n");
        return (1);
    }

    rc = SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, 1000,
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection failed!\n");
        return (1);
    }

    fprintf (stdout, "Connected\n");

    rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Statement Handle Allocation failed!\n");
        return (1);
    }

    rc = SQLExecDirect (hstmt, sqlstmt, SQL_NTS);
    if (rc != SQL_SUCCESS &&  rc != SQL_SUCCESS_WITH_INFO)
    {
        fprintf (stdout, "SQLExecDirectW() failed!\n");
        return (1);
    }

    fprintf (stdout, "Executed SQL Statement:\n%s\n",sqlstmt);

    SQLDisconnect (hdbc);

    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-12 shows the result of running the sample program with different SQL statements. In this example, we input the SQL statement directly from the command line. In a real application, the SQL is normally constructed based on customer input.

In this simple example, we also did not provide error information if the execution fails. You can obtain error information using the `SQLGetDiagRec()` function, which we discuss in 3.3.5, “Error handling” on page 112.

Example 3-12 Output of SQLDirectExec sample

```
c:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -I%INFORMIXDIR%\incl\cli
odbc32.lib /nologo simple_sql.c
simplesql.c

c:\work>simples_sql.exe test "set lock mode to wait 10"
Connected
Executed SQL Statement:
set lock mode to wait 10

c:\work>simple_sql.exe test "create temp table temp1(c1 int)"
Connected
Executed SQL Statement:
create temp table temp1(c1 int)

c:\work>simple_sql.exe test "invalid_SQL"
Connected
SQLExecDirectW() failed!

c:\work>
```

Fetching data

Example 3-13 demonstrates how to run a SELECT statement to retrieve data from the server. To make the code clear, we have removed all the error handling.

Example 3-13 Select.c sample

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;
    SQLCHAR         sqlstmt[100];
    SQLCHAR         code[2+1];
    SQLCHAR         sname[15+1];
    SQLLEN          lcode=0;
    SQLLEN          lsname=0;


    sprintf((char *) connStrIn, "DSN=demo_on");
    sprintf((char *) sqlstmt, "SELECT code, sname FROM state where code<'CA'");

    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, 1000, &connStrOutLen,
                      SQL_DRIVER_NOPROMPT);
    fprintf (stdout, "Connected\n");

    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
    SQLExecDirect (hstmt, sqlstmt, SQL_NTS);

    fprintf (stdout, "Executed SQL Statement:\n%s\n",sqlstmt);
    SQLBindCol (hstmt, 1, SQL_C_CHAR, &code, 3, &lcode);
    SQLBindCol (hstmt, 2, SQL_C_CHAR, &sname, 16, &lsname);

    while (SQLFetch(hstmt)!=SQL_NO_DATA_FOUND)
    fprintf (stdout, "Fetched row: %s, %s\n",code, sname);

    SQLCloseCursor(hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-14 shows the output of this sample.

Example 3-14 Output of select.c

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -I%INFORMIXDIR%\incl\cli
odbc32.lib /nologo select.c
select.c

C:\work>select
Connected
Executed SQL Statement:
SELECT code, sname FROM state where code<'CA'
Fetched row: AK, Alaska
Fetched row: AL, Alabama
Fetched row: AR, Arkansas
Fetched row: AZ, Arizona

C:\work>
```

Using parameters

In most of the cases, an application runs the same SQL statement several times, so it makes sense if the SQL statement is prepared and then used with different values.

Example 3-15 demonstrates how to run a simple SQL SELECT statement using an input parameter. The SQL statement is prepared and then executed with one parameter.

Example 3-15 Example of a parametrized query, Select_param.c

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;
    SQLCHAR         sqlstmt[100];
    SQLCHAR         inputcode[2+1];
    SQLCHAR         code[2+1];
    SQLCHAR         sname[15+1];
    SQLLEN          lcode = 0;
    SQLLEN          lsname = 0;
    SQLSMALLINT     datatype, decimaldigits, nullable;
    SQLUINTEGER     paramsize;


    sprintf((char *) connStrIn, "DSN=demo_on");
    sprintf((char *) sqlstmt, "SELECT code, sname FROM state where code < ?");
    sprintf((char *) inputcode, "CA");

    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, sizeof(connStrOut),
&connStrOutLen, SQL_DRIVER_NOPROMPT);
```

```
    fprintf (stdout, "Connected\n");

    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

    SQLPrepare (hstmt, sqlstmt, strlen(sqlstmt));
    SQLDescribeParam(hstmt, 1, &datatype, &paramsize, &decimaldigits, &nullable);
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, datatype, paramsize,
decimaldigits, inputcode, 0, NULL);

    SQLExecute (hstmt);
    fprintf (stdout, "Executed SQL Statement:\n%s\nUsing: '%s'\n",sqlstmt,inputcode);

    SQLBindCol (hstmt, 1, SQL_C_CHAR, &code, sizeof(code), &lcode);
    SQLBindCol (hstmt, 2, SQL_C_CHAR, &sname, sizeof(sname), &lsname);

    while ((rc = SQLFetch(hstmt))!=SQL_NO_DATA_FOUND)
    fprintf (stdout, "Fetched: %s, %s\n",code, sname);

    SQLCloseCursor(hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-16 shows the SQL statement that the program prepares and the parameter that is used for the placeholder.

Example 3-16 Output for Select_param.c

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -ID:\infx\csdk350tc7\incl\cli
odbc32.lib /nologo select_param.c
select_param.c

C:\work>select_param
Connected
Executed SQL Statement:
SELECT code, sname FROM state where code < ?
Using: 'CA'
Fetched: AK, Alaska
Fetched: AL, Alabama
Fetched: AR, Arkansas
Fetched: AZ, Arizona

C:\work>
```

Calling SQL routines

In this section, we demonstrate how to call an SQL routine from an ODBC application. The following example is a simple stored procedure that returns the user name and session ID:

```
create procedure get_sid(user char(20)) returning char(100);
return 'user= '||trim(user)|| ' session= '||dbinfo('sessionid');
end procedure;
```

We use the following ODBC standard syntax for calling a database routine:

```
{? = call client_routine(?, ?,...)}
```

The first placeholder (?) is used only when the first parameter of the routine is an output parameter. If the first parameter is not an output parameter, you can run the routine (stored procedure or SQL function) using the following syntax:

```
{call client_routine(?, ?, ?, ?)}
```

Example 3-17 illustrates how to call the `get_sid()` SQL function.

Example 3-17 SPL function sample, Function.c

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;
    SQLCHAR         sqlstmt[100];
    SQLCHAR         errorn[20+1];
    SQLCHAR         result[100+1];
    SQLLEN          lresult = 0;

    sprintf((char *) connStrIn, "DSN=demo_on");
    sprintf((char *) sqlstmt, "{? = call get_sid(?)}");
    sprintf((char *) errorn, "informix");
    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, sizeof(connStrOut),
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    fprintf (stdout, "Connected\n");

    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

    SQLBindParameter(hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR,sizeof(result),0,
result, sizeof(result), &lresult);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, strlen(errorn),0,
errorn, 0, NULL);

    SQLExecDirect (hstmt, sqlstmt, SQL_NTS);
    SQLFetch(hstmt);

    fprintf (stdout, "Executed SQL Statement:\n%s\nUsing: '%s'\n",sqlstmt,errorn);
    fprintf (stdout, "Value returned: %s\n",result);

    SQLCloseCursor(hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-18 shows the input parameter passed to the `get_sid()` procedure and the return values.

Example 3-18 Output of Function.c

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -ID:\infx\csdk350tc7\incl\cli
odbc32.lib /nologo function.c
function.c

C:\work>function
Connected
Executed SQL Statement:
{? = call get_sid(?)}
Using: 'informix'
Value returned: user: informix session: 213

C:\work>
```

Local transactions

The default transaction mode for the Informix ODBC driver is auto-commit. The transaction is committed automatically if the SQL statement runs successfully.

You can switch to manual-commit by setting the SQL_AUTOCOMMIT_OFF attribute using the `SQLSetConnectAttr()` function. When an application sets SQL_AUTOCOMMIT_OFF, the next SQL statement automatically starts a transaction that remains open until the application calls `SQLEndTran()`.

In manual mode, all the statements executed by the application are committed or rolled back when the application calls `SQLEndTran()`.

Having auto-commit set might be more convenient when running simple SQL statements because there are less tasks to care about in the application code. However, auto-commit gives less control to the developer than using the manual-commit mode. When using auto-commit, there is no option to roll back a particular change in the database or to insert a multiple rows as a batch operation, which might improve the performance of the application.

Example 3-19 illustrates how to run a local transaction in manual mode.

Example 3-19 Transaction.c

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;
    SQLCHAR         sqlstmt[100];
    SQLINTEGER      cnum = 101;
    SQLSMALLINT     datatype, decimaldigits, nullable;
    SQLUINTEGER     paramsize;


    sprintf((char *) connStrIn, "DSN=demo_on");
    sprintf((char *) sqlstmt, "INSERT INTO orders(order_num,order_date,customer_num)
VALUES (0,current,?)");
```

```
    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, sizeof(connStrOut),
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    fprintf (stdout, "Connected\n");

    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

    SQLSetConnectAttr (hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, (SQLINTEGER)
NULL);

    SQLPrepare (hstmt, sqlstmt, strlen(sqlstmt));
    SQLDescribeParam(hstmt, 1, &datatype, &paramsize, &decimaldigits, &nullable);
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT, datatype, paramsize,
decimaldigits, &cnum, 0, NULL);

    while (cnum<105)
    {
       SQLExecute (hstmt);
       fprintf (stdout, "Executed SQL Statement:\n%s\nUsing: '%d'\n",sqlstmt,cnum);
       cnum++;
    }
    rc = SQLEndTran (SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK);
    fprintf (stdout, "Transaction Rolled back\n");

    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-20 shows the output of the previous example. A local transaction is created, five rows are inserted in the `orders` table, and then the transaction is rolled back using `SQLEndTran()`.

Example 3-20 Output of Trasaction.C

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -ID:\infx\csdk350tc7\incl\cli
odbc32.lib /nologo transact.c
transact.c

C:\work>del pp*

C:\work>transact.exe
Connected
Executed SQL Statement:
INSERT INTO orders(order_num,order_date,customer_num) VALUES (0,current,?)
Using: '101'
Executed SQL Statement:
```

```
INSERT INTO orders(order_num,order_date,customer_num) VALUES (0,current,?)
Using: '102'
Executed SQL Statement:
INSERT INTO orders(order_num,order_date,customer_num) VALUES (0,current,?)
Using: '103'
Executed SQL Statement:
INSERT INTO orders(order_num,order_date,customer_num) VALUES (0,current,?)
Using: '104'
Transaction Rolled back

C:\work>
```

Distributed transactions

On a Windows system, you can run a distributed transaction using the Microsoft Distributed Transaction Coordinator (MSDTC) service.

Example 3-21 demonstrates how to run a distributed transaction. The example opens two connections to two different database servers and enlists both connections into the same distributed transaction.

Example 3-21 Transaction_dtc.cpp

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#include <txdtc.h>
#include <xolehlp.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc, hdbc2;
    SQLHENV         henv;
    SQLHSTMT        hstmt, hstmt2;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[100],connStrIn2[100];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;
    SQLCHAR         sqlstmt[100];
    SQLINTEGER      cnum = 101;
    SQLSMALLINT     datatype, decimaldigits, nullable;
    SQLUINTEGER     paramsize;
    ITransactionDispenser *pTransactionDispenser = NULL;
    ITransaction * pITransaction;

    sprintf((char *) connStrIn , "DSN=server_1");
    sprintf((char *) connStrIn2, "DSN=server_2");
    sprintf((char *) sqlstmt, "INSERT INTO orders(order_num,order_date,customer_num)
VALUES (0,current,101)");

    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc2);

    // Get DTC
    DtcGetTransactionManager(NULL, NULL,IID_ITransactionDispenser, 0, 0, NULL, (void**)
&pTransactionDispenser);

    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, sizeof(connStrOut),
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    fprintf (stdout, "Connected to %s\n",connStrIn);

    SQLDriverConnect (hdbc2, NULL, connStrIn2, SQL_NTS, connStrOut, sizeof(connStrOut),
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    fprintf (stdout, "Connected to %s\n",connStrIn2);

    //Start the transaction on DTC
    pTransactionDispenser->BeginTransaction
(NULL,ISOLATIONLEVEL_READCOMMITTED,0,NULL,&pITransaction);

    //Enlist the connections in distributed transaction
    SQLSetConnectAttr(hdbc, SQL_ATTR_ENLIST_IN_DTC,
(SQLPOINTER)pITransaction,SQL_IS_INTEGER);
    SQLSetConnectAttr(hdbc2, SQL_ATTR_ENLIST_IN_DTC,
(SQLPOINTER)pITransaction,SQL_IS_INTEGER);

    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
    SQLExecDirect (hstmt, sqlstmt, SQL_NTS);
    fprintf (stdout, "Executed SQL Statement:\n%s\nUsing: '%d'\n",sqlstmt,cnum);

    SQLAllocHandle (SQL_HANDLE_STMT, hdbc2, &hstmt2 );
    if (SQLExecDirect (hstmt2, sqlstmt, SQL_NTS)!=SQL_SUCCESS)
    {
        fprintf(stdout, "Rolling back global transaction\n");
        pITransaction->Abort(NULL,FALSE,FALSE);
    }
    else
    {
        pITransaction->Commit( 0, XACTTC_SYNC_PHASEONE, 0 );
        pITransaction->Release();
        fprintf(stdout, "Transaction Committed\n");
    }
    SQLFreeHandle(SQL_HANDLE_STMT,hstmt2);
    SQLFreeHandle(SQL_HANDLE_STMT,hstmt);
    SQLDisconnect (hdbc2);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc2);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-22 shows how to compile the sample and the output of the program.

Example 3-22 Output of Transact_dtc.cpp

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -I%INFORMIXDIR%\incl\cli
odbc32.lib xoleHlp.lib /nologo transaction_dtc.cpp
transaction_dtc.cpp

C:\work>transaction_dtc
Connected to DSN=server_1
Connected to DSN=server_2
Executed SQL Statement:
INSERT INTO orders(order_num,order_date,customer_num) VALUES (0,current,101)
Using: '101'
Transaction Committed

C:\work>
```

Note: Remember that the DSN must be accessible to the MSDTC service. In Example 3-21 on page 93, `test_1` and `test_2` are both created as System-DSN.

3.3.4 Handling special data types

This section describes how to work with the Informix specific data types, such as smart large objects and complex data types.

Smart large objects

Smart large objects are a type of large objects supported by Informix. Smart large objects are logically stored in a table column but physically stored in a specific type of dbspaces called smart blob space (sbspace).

The information stored in the table column is structure that contains information about the large object, such as the pointer to the location in the sbspace that contains the data or special attributes. Informix has two types of smart large objects.

- BLOB: Stores binary data
- CLOB: Stores character data

A client application uses these data structures to perform random I/O operations on the large data, such as open, read, or write operations. There are two options for accessing smart large objects from an ODBC application:

- Use the smart-large-object ODBC API.

  If the application requires random access to the large data, it must use the smart large object functions. These functions give the application a greater control over the smart-large-object data in terms of object properties, concurrency access, and logging.

- Use smart-large-object automation.

  The ODBC driver automatically uses the ODBC API for handling large objects. The application can access smart large objects as standard ODBC data types (SQL_LONGVARBINARY and SQL_LONGVARCHAR).

  To enable smart-large-object automation use the “Report Standard ODBC Types Only DSN” option under the DSN Advanced tab or set the SQL_INFX_ATTR_LO_AUTOMATIC connection attribute.

Data structures for smart large objects

Table 3-10 describes the data structures used by the smart-large-object functions.

Table 3-10 Smart large object data structures

Data structure	Name	Description
lofd	file descriptor	Defines the file descriptor to access smart-large-object data.
loptr	pointer structure	Contains the security information and pointer for a smart large object. This is the data stored with the table columns.
lospec	specification structure	Contains the storage characteristics for a smart large object.
lostat	status structure	Contains the status information for a smart large object.

Client functions to access smart large objects

Table 3-11 lists the SQL functions that you can use in your ODBC program to access the Informix smart large objects.

Table 3-11 Smart large object client functions

Function	Description
ifx_lo_alter(loptr, lospec)	Alters the storage attributes like Logging or Last-access
ifx_lo_close(lofd)	Closes a smart large object
ifx_lo_col_info(colname, lospec)	Updates column-level storage characteristics
ifx_lo_create(lospec, flags, loptr, lofd)	Creates and opens a new smart large object
ifx_lo_def_create_spec(lospec)	Creates a smart-large-object specification structure
ifx_lo_open(lofd, loptr, flags)	Opens a smart large object
ifx_lo_read(lofd, buf)	Reads data from a smart large object
ifx_lo_readwithseek(lofd, buf, offset, whence)	Reads starting at a specific location
ifx_lo_seek(), ifx_lo_seek(lofd, offset, whence, seek_pos)	Sets the file position for the next operation
ifx_lo_specget_estbytes(lospec, estbytes)	Returns the estimated size
ifx_lo_specget_extsz(lospec, extsz)	Returns the allocation extent size
ifx_lo_specget_flags(lospec, flags)	Returns create-time flags
ifx_lo_specget_maxbytes(lospec, maxbytes)	Returns the maximum number of bytes
ifx_lo_specget_sbspace(lospec, sbspace)	Returns the sbspace name
ifx_lo_specset_estbytes(lospec, estbytes)	Sets the estimated number of bytes
ifx_lo_specset_extsz(lospec, extsz)	Sets the allocation extent size
ifx_lo_specset_flags(lospec, flags)	Sets the create-time flags
ifx_lo_specset_maxbytes(lospec, maxbytes)	Sets the maximum number of bytes

Function	Description
ifx_lo_specset_sbspace(lospec, sbspace)	Sets the sbspace name
ifx_lo_stat(lofd, lostat)	Initializes a smart-large-object status structure
ifx_lo_stat_atime(lostat, atime)	Returns the last-access time
ifx_lo_stat_cspec(lostat, lospec)	Returns the specification structure
ifx_lo_stat_ctime(lostat, ctime)	Returns the last-time change
ifx_lo_stat_refcnt(lostat, refcount)	Returns the number of references
ifx_lo_stat_size(lostat, size)	Returns the size
ifx_lo_tell(lofd, seek_pos)	Returns the current file position
ifx_lo_truncate(lofd, offset)	Truncates a smart large object at a given position
ifx_lo_write(lofd, buf)	Writes data
ifx_lo_writewithseek(lofd, buf, offset, whence)	Writes data at a specific location

Calling client functions from ODBC

An application must use the following standard ODBC syntax to call the smart-large-object functions:

```
{? = call function_name (?, ?,...)}
```

The following example calls the `ifx_lo_open()` function:

```
{? = call ifx_lo_open(?, ?, ?)}
```

Creating a smart large object

To create a smart large object, you must create a BLOB descriptor and then insert the BLOB into the database.

Use the following steps to create a new smart large object:

1. Allocate memory for the specification structure using `lospec`.
2. Create a `lospec` using `ifx_lo_def_create_spec()`.
3. Initialize the specification structure using `lospec`.
4. Allocate memory for the pointer structure using `loptr`.
5. Create the object using `ifx_lo_create()`.

6. Write the data into the object using `ifx_lo_write()`.
7. Perform an SQL operation (INSERT, UPDATE, and so on).
8. Close the smart large object using `ifx_lo_close()`.

Example 3-23 shows how to insert a BLOB into the database.

Example 3-23 Create_lo.c

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#include <fcntl.h>
#include <sys/stat.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000], connStrOut[1000], sqlstmt[100];
    SQLSMALLINT     connStrOutLen;
    SQLINTEGER      cnum = 101;
    SQLSMALLINT     datatype, decimaldigits, nullable;
    SQLUINTEGER     paramsize;
    SQLCHAR    colname[25] = "catalog.advert_descr";
    SQLINTEGER      colname_size = SQL_NTS;
    SQLCHAR*        file_name = (SQLCHAR *) "create_lo.c";
    SQLCHAR*        blob_data;
    SQLSMALLINT     blob_size;
    SQLINTEGER      blob_wsize, mode = LO_RDWR, cbMode = 0;
    int     fd=0;
    struct stat     statbuf;

/* BLOB file descriptor */
    SQLINTEGER      lofd;
    SQLINTEGER      lofd_valsize = 0;
/* BLOB pointer structure */
    SQLCHAR*        loptr_buffer;
    SQLSMALLINT     loptr_size;
    SQLINTEGER      loptr_valsize = 0;
/* BLOB specification structure */
    SQLCHAR*        lospec_buffer;
    SQLSMALLINT     lospec_size;
    SQLINTEGER      lospec_valsize = 0;


    sprintf((char *) connStrIn, "DSN=demo_on");
    sprintf((char *) sqlstmt, "INSERT INTO catalog(catalog_num,advert_descr) VALUES
(0,?)");

 /* Read the file to insert in the BLOB */
    stat(file_name,&statbuf);
    blob_size = statbuf.st_size;
    blob_data = malloc (blob_size + 1);
    fd = _open (file_name, O_RDONLY);
    _read (fd, blob_data, blob_size);
    _close(fd);
    blob_data[blob_size] = '\0';
    blob_wsize = blob_size;

    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, sizeof(connStrOut),
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

/* Obtain the size of the lospec and allocate memory */

    SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size, sizeof(lospec_size), NULL);
    lospec_buffer = malloc (lospec_size);

/* Call ifx_lo_def_create_spec() to create a lospec */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer, lospec_size,
&lospec_valsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_def_create_spec(?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* Call ifx_lo_col_info() to initialise the lospec */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, sizeof(colname),
0, colname, sizeof(colname), &colname_size);
    lospec_valsize = lospec_size;
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer, lospec_size,
&lospec_valsize);
    rc=SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_col_info(?, ?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* Obtain the size of the smart loptr and allocate memory */
    SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size, sizeof(loptr_size), NULL);
    loptr_buffer = malloc (loptr_size);

/* Call ifx_lo_create() to create the BLOB */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_UDT_FIXED,
(UDWORD)lospec_size, 0, lospec_buffer, lospec_size, &lospec_valsize);
```

```
    rc=SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, (UDWORD)0,
0, &mode, sizeof(mode), &cbMode);
    loptr_valsize = loptr_size;

    SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer, loptr_size, &loptr_valsize);
    SQLBindParameter (hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, (UDWORD)0,
0, &lofd, sizeof(lofd), &lofd_valsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_create(?, ?, ?, ?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* Call ifx_lo_write to write the BLOB data */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, (UDWORD)0, 0,
&lofd, sizeof(lofd), &lofd_valsize);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
(UDWORD)blob_size, 0, blob_data, blob_size, &blob_wsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_write(?, ?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    loptr_valsize = loptr_size;

/* Call SQLExecDirect to do the INSERT  */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_UDT_FIXED,
(UDWORD)loptr_size, 0, loptr_buffer, loptr_size, &loptr_valsize);
    SQLExecDirect (hstmt, sqlstmt, SQL_NTS);

/* Call ifx_lo_close() to close the BLOB */
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
(UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_close(?)}", SQL_NTS);

    free (lospec_buffer);
    free (loptr_buffer);
    free (blob_data);

    SQLFreeStmt (hstmt, SQL_CLOSE);
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Accessing a smart large object

Reading a smart large object from the database involves the following steps:

1. Allocate memory for the smart-large-object pointer structure (`loptr`).
2. Perform the SELECT statement.
3. Bind the smart-large-object pointer structure variable with `loptr` retrieved from the database.

4. Call `ifx_lo_open()` to open the smart large object with the `loptr` fetched from the database. This step opens the smart large object on the database.
5. Obtain the size of the smart large object status structure (`lostat`) and allocate enough memory.
6. Call `ifx_lo_stat()` to obtain the BLOB status structure, so that we can retrieve the estimated size of the BLOB in the next step.
7. Call `ifx_lo_stat_size()` to retrieve the size of the BLOB data and to allocate enough memory.
8. Call `ifx_lo_read()` to retrieve the BLOB data.
9. Perform any operation with the BLOB data.
10. Call `ifx_lo_close()` to close the BLOB.

Example 3-24 shows a simple application that selects a CLOB column from the database.

Example 3-24 The select_lo.c application

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#include <fcntl.h>
#include <sys/stat.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBChdbc;
    SQLHENVhenv;
    SQLHSTMThstmt;
/* BLOB file descriptor */
    SQLLENlofd;
    SQLLENlofd_valsize = 0;
/* BLOB pointer structure */
    SQLCHAR*loptr_buffer;
    SQLSMALLINTloptr_size;
    SQLLENloptr_valsize = 0;
/* BLOB status structure */
    SQLCHAR*lostat_buffer;
    SQLSMALLINTlostat_size;
    SQLLENlostat_valsize = 0;
/* BLOB data */
```

```
    SQLCHAR*lo_data;
    SQLLENlo_data_valsize = 0;

    SQLRETURNrc = 0;
    SQLCHAR*dsn = "demo_on";
    SQLCHAR*selectStmt = (SQLCHAR *) "SELECT advert_descr FROM catalog where catalog_num 
= 10075";

    SQLINTEGERmode = LO_RDONLY;
    SQLLENlo_size;
    SQLLENcbMode = 0, cbLoSize = 0;


    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLConnect (hdbc, dsn, SQL_NTS, (SQLCHAR *) "", SQL_NTS, (SQLCHAR *) "", SQL_NTS);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

/* Get the size of the loptr */
    SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size, sizeof(loptr_size), NULL);
    loptr_buffer = malloc (loptr_size);

    SQLExecDirect (hstmt, selectStmt, SQL_NTS);

/* Bind the loptr with the resulset column */
    SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size, &loptr_valsize);
    SQLFetch (hstmt);
    SQLCloseCursor (hstmt);

/* Call ifx_lo_open() using the loptr fetched from the database */
    SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG, SQL_INTEGER, (UDWORD)0, 0, 
&lofd, sizeof(lofd), &lofd_valsize);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_UDT_FIXED, 
(UDWORD)loptr_size, 0, loptr_buffer, loptr_size, &loptr_valsize);
    SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, (UDWORD)0, 0, 
&mode, sizeof(mode), &cbMode);
    SQLExecDirect (hstmt, (SQLCHAR *) "{? = call  ifx_lo_open(?, ?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* Get the size of the lostat and allocate enough memory */
    SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size, sizeof(lostat_size), NULL);
    lostat_buffer = malloc(lostat_size);

/* Call ifx_lo_stat() to get the BLOB lostat */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, (UDWORD)0, 0, 
&lofd, sizeof(lofd), &lofd_valsize);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY, 
SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer, lostat_size, 
&lostat_valsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_stat(?, ?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* Call ifx_lo_stat_size to get the size of the BLOB data and allocate enough memory*/
```

```
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_UDT_FIXED,
(UDWORD)lostat_size, 0, lostat_buffer, lostat_size, &lostat_valsize);
    SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG, SQL_BIGINT, (UDWORD)0, 0,
&lo_size, sizeof(lo_size), &cbLoSize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
    lo_data = malloc (lo_size + 1);

/* Call ifx_lo_read() to get the BLOB data */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, (UDWORD)0, 0,
&lofd, sizeof(lofd), &lofd_valsize);
    SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR, lo_size, 0,
lo_data, lo_size, &lo_data_valsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_read(?, ?)}", SQL_NTS);
    lo_data[lo_size] = '\0';
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* Call ifx_lo_close() to close the BLOB */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, (UDWORD)0, 0,
&lofd, sizeof(lofd), &lofd_valsize);
    SQLExecDirect (hstmt, (SQLCHAR *) "{call ifx_lo_close(?)}",SQL_NTS);

    fprintf(stdout,"%s",lo_data);

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

    SQLFreeStmt (hstmt, SQL_CLOSE);
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Smart large object automation

Developing an application to handle smart large objects using the smart large object automation method does not require any special attention. Use SQL_LONGBINARY and SQL_LONGVARCHAR as data type.

Example 3-25 Illustrates how to insert a smart large object with the SQL_INFX_ATTR_LO_AUTOMATIC enabled.

Example 3-25 Lo_auto.c

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

```
#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#include <fcntl.h>
#include <sys/stat.h>
#endif

#include "infxcli.h"

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

    SQLHDBC     hdbc;
    SQLHENV     henv;
    SQLHSTMT    hstmt;

    SQLCHAR*      dsn="demo_on";
    SQLCHAR       sqlstmt[128];
    SQLCHAR       str[128];
    SQLLEN        str_len = SQL_NTS;
    SQLSMALLINT   BufferLength = 128;
    SQLINTEGER    int_val;
    SQLRETURN     rc = 0;


    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLConnect (hdbc, dsn, SQL_NTS, (SQLCHAR *) "", SQL_NTS, (SQLCHAR *) "", SQL_NTS);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

/* Set SQL_INFX_ATTR_ODBC_TYPES_ONLY connection attribute to FALSE*/
    SQLSetConnectAttr(hdbc, SQL_INFX_ATTR_ODBC_TYPES_ONLY, (SQLPOINTER) SQL_FALSE,
SQL_IS_UINTEGER);

/* Set SQL_INFX_ATTR_LO_AUTOMATIC connection attribute to TRUE*/
    SQLSetConnectAttr(hdbc, SQL_INFX_ATTR_LO_AUTOMATIC, (SQLPOINTER) SQL_TRUE,
SQL_IS_UINTEGER);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

/* Preare the SQL statement */
    sprintf(sqlstmt, "INSERT INTO catalog(catalog_num,advert_descr) VALUES (0,?)");
    SQLPrepare (hstmt, sqlstmt, SQL_NTS);

/* Bind the input parameter and Execute the SQL */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR,
BufferLength, 0, &str, 0, &str_len);
    SQLExecute (hstmt);

    SQLFreeStmt (hstmt, SQL_CLOSE);
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Complex data types

Rows and Collections are composite values that consist of one or more elements. You can use the execute SQL statements to access an entire row or collection. However, you cannot access individual elements within a row or collection.

To access composite values from the ODBC driver you must use the Client functions for Rows and Collections.

Client functions for Rows and Collections

The ODBC functions for handling Rows and Collections start with the prefix *ifx_rc_*. Table 3-12 lists the available functions.

Table 3-12 Row and Collection Client functions

Function	Description
ifx_rc_count()	Returns the number of elements or fields that are in a row or collection
ifx_rc_create()	Creates a buffer for a row or collection
ifx_rc_delete()	Deletes an element from a collection
ifx_rc_describe()	Returns information about the complex data type or and individual element
ifx_rc_fetch()	Returns the value of an element that is in a row or collection
ifx_rc_free()	Frees a row or collection handle
ifx_rc_insert()	Inserts a new element into a collection
ifx_rc_isnull()	Evaluate if a complex type is NULL or not
ifx_rc_setnull()	Set a complex type to NULL
ifx_rc_typespec()	Returns the type specification for a row or collection
ifx_rc_update()	Updates the value for an element that is in a row or collection

Creating a complex data type

Example 3-26 shows how to perform an INSERT operation using the Informix ODBC functions `ifx_rc_xxx`. We use a simple table `customer_rc` defined as follows:

```
CREATE ROW TYPE name_t (
   fname   VARCHAR(15),
   lname   VARCHAR(15)
);

CREATE TABLE customer_rc (
   customer_num    SERIAL,
   customer_name   name_t,
   contact_dates   LIST(DATETIME YEAR TO DAY NOT NULL)
);
```

Example 3-26 Create_rc.c

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#include <fcntl.h>
#include <sys/stat.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBChdbc;
    SQLHENVhenv;
    SQLHSTMThstmt;

/* Row variables */
    HINFX_RChrow;
    HINFX_RChlist;

    SQLCHAR*dsn = "demo_on";
    SQLRETURNrc = 0;
    SQLINTEGERi, in;
    SQLLENdata_size = SQL_NTS;
    SQLSMALLINTposition = SQL_INFX_RC_ABSOLUTE;
    SQLSMALLINTjump;
    SQLCHARrow_data[2][15] = {"Ludwig", "Pauli"};
    SQLLENrow_data_size = SQL_NTS;
    SQLCHARlist_data[2][25] = {"2008-08-16","2008-08-16"};
    SQLLENlist_data_size = SQL_NTS;
    SQLCHAR*insertStmt = (SQLCHAR *) "INSERT INTO customer_rc VALUES (0, ?, ?)";
    SQLLENcbHrow = 0, cbHlist = 0, cbPosition = 0, cbJump = 0;

/* Connection */
    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLConnect (hdbc, dsn, SQL_NTS, (SQLCHAR *) "", SQL_NTS, (SQLCHAR *) "", SQL_NTS);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

/* Call ifx_rc_create() to allocate a row handle */
    SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY, SQL_INFX_RC_ROW,
sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC), &cbHrow);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0, (SQLCHAR *)
"ROW(fname VARCHAR(15), lname VARCHAR(15))", 0, &data_size);
    SQLExecDirect (hstmt, (SQLCHAR *) "{? = call ifx_rc_create(?)}", SQL_NTS);

/* Call ifx_rc_create() to allocate a list handle */
    SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY, SQL_INFX_RC_LIST,
sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC), &cbHlist);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0, (SQLCHAR *)
"LIST (DATETIME YEAR TO DAY NOT NULL)", 0, &data_size);
    SQLExecDirect (hstmt, (SQLCHAR *) "{? = call ifx_rc_create(?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

/* call ifx_rc_update() to insert elements into the row */
    for (i=0; i<2; i++)
    {
        SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_ROW,
sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC), &cbHrow);
        SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 25, 0,
row_data[i], 0, &row_data_size);
        SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
&position, 0, &cbPosition);
        jump = i + 1;
        SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
&jump, 0, &cbJump);
        SQLExecDirect (hstmt, (SQLCHAR *)"{call ifx_rc_update(?, ?, ?, ?)}", SQL_NTS);
    }

/* Call ifx_rc_insert() to insert elements into the list */
    for (i=0; i<2; i++)
    {
        SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_LIST,
sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC), &cbHlist);
        SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,SQL_DATE, 25, 0,
list_data[i], 0, &list_data_size);
        SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
&position, 0, &cbPosition);
        jump = i + 1;
        SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
&jump, 0, &cbJump);
        SQLExecDirect (hstmt, (SQLCHAR *)"{call ifx_rc_insert( ?, ?, ?, ? )}", SQL_NTS);
    }
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_COLLECTION,
sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC), &cbHrow);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_COLLECTION,
sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC), &cbHlist);
    SQLExecDirect (hstmt, insertStmt, SQL_NTS);

    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_ROW,
sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC), &cbHrow);
    SQLExecDirect(hstmt, (SQLCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

/* Free the list handle */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_LIST,
sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC), &cbHlist);
    SQLExecDirect(hstmt, (SQLCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

    SQLFreeStmt (hstmt, SQL_CLOSE);
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);
    return (rc);
}
```

Example 3-27 shows the information inserted into the `customer_rc` table after the execution of the example.

Example 3-27 The customer_rc table

```
> select * from customer_rc
> ;


customer_num   1
customer_name  ROW('Ludwig','Pauli')
contact_dates  LIST{'2008-08-16','2008-08-16'}

1 row(s) retrieved.

>
```

Accessing a complex data type

Example 3-28 demonstrates how to select a Row type with the ODBC driver.

Example 3-28 The select_row.c application

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#include <fcntl.h>
#include <sys/stat.h>
#endif

#include "infxcli.h"


int main (long argc, char* argv[])
{
    SQLHDBC          hdbc;
    SQLHENV          henv;
    SQLHSTMT         hstmt, hstmt_row;

    HINFX_RC    hrow;
    SQLCHAR          *dsn="demo_on";
    SQLRETURN        rc = 0;
    SQLINTEGER       i;
    SQLSMALLINT      position = SQL_INFX_RC_ABSOLUTE, jump = 0;
    SQLLEN           cbHrow = 0, cbPosition = 0, cbJump = 0, cbRCData = 0;
    SQLLEN     data_size = SQL_NTS, cbrow = 0;
    SQLCHAR*    sqlstmt = (SQLCHAR *) "SELECT customer_name FROM customer_rc WHERE 
customer_num = 1";
    SQLCHAR          row_data[200];

/* Connection */
    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLConnect (hdbc, dsn, SQL_NTS, (SQLCHAR *) "", SQL_NTS, (SQLCHAR *) "", SQL_NTS);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt_row);

/* Call ifx_rc_create() to allocate a row handle */
    SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY, SQL_INFX_RC_ROW, 
sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC), &cbHrow);
    SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0, (SQLCHAR *) 
"row", 0, &data_size);
    SQLExecDirect (hstmt, (SQLCHAR *) "{? = call ifx_rc_create(?)}", SQL_NTS);
    SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
/* Bind the row handle with the resulset column */
    SQLExecDirect (hstmt, sqlstmt, SQL_NTS);
    SQLBindCol (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) hrow, sizeof(HINFX_RC), &cbHrow);
    SQLFetch (hstmt);

/* Get the row data */
    fprintf(stdout, "Row data:\n");
    for (i=0; i<2; i++)
    {
       strcpy((char *) row_data, "<null>");

       SQLBindParameter (hstmt_row, 1, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR, 0, 0,
row_data, 200, &cbRCData);
       SQLBindParameter (hstmt_row, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC), &cbHrow);
       SQLBindParameter (hstmt_row, 3, SQL_PARAM_INPUT, SQL_C_SHORT,  SQL_SMALLINT, 0,
0, &position, 0, &cbPosition);
       jump = i + 1;
       SQLBindParameter (hstmt_row, 4, SQL_PARAM_INPUT, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
&jump, 0, &cbJump);

/* Call ifx_rc_fetch() to fetch individual elements from the row */
       SQLExecDirect (hstmt_row, (SQLCHAR *) "{ ? = call ifx_rc_fetch( ?, ?, ? ) }",
SQL_NTS);
       SQLFreeStmt (hstmt_row, SQL_RESET_PARAMS);

       fprintf(stdout, "\t\t%s\n", row_data);
    }

/* Call ifx_rc_free() to free the row handle */
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_INFX_RC_ROW,
sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC), &cbrow);
    SQLExecDirect(hstmt, (SQLCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

    SQLFreeStmt (hstmt, SQL_CLOSE);
    SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-29 shows the output of Example 3-28 on page 110.

Example 3-29 Output of select_row.c

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -ID:\infx\csdk350tc7\incl\cli
odbc32.lib /nologo select_row.c
select_row.c

C:\work>select_row
Row data:
```

```
                 Ludwig
                 Pauli

C:\work>
```

3.3.5 Error handling

In this section, we show how to obtain additional information when an ODBC function fails.

Error handling sample

When an ODBC function fails, it stores information in the diagnostic record. This record contains information such as error messages, warning, and status information about the success or failure of the ODBC call.

An ODBC application can retrieve the diagnostic record or individual fields using the `SQLGetDiagRec()` and `SQLGetDiagField()` ODBC functions.

Example 3-30 shows a typical usage of `SQLGetDiagRec()`.

Example 3-30 A sample connect_error.c application

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;
    UCHAR           SqlState[200] = "", ErrorMsg[200] = "";
    SQLINTEGER      IsamError = 0;
    SDWORD          NativeError = 0L;
    SWORD           ErrorMsgp = 0;
    SQLSMALLINT     recnum = 1;

    if (argc != 2)
    {
        fprintf (stdout, "Please specify the name of a DSN!\n");
    return(1);
    }
    else
    {
    if (strstr (argv[1],"DRIVER")==NULL)
       sprintf((char *) connStrIn, "DSN=%s;", (char *)argv[1]);
    else
       sprintf((char *) connStrIn, "%s;", (char *)argv[1]);
    }

    rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    rc = SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, 1000,
&connStrOutLen, SQL_DRIVER_NOPROMPT);
    if (rc != SQL_SUCCESS)
    {
        SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, recnum, SqlState, &NativeError, ErrorMsg,
199, &ErrorMsgp);
        fprintf(stdout, "SqlState = %s\n Native Error = %d\n Error Message = %s\n",
SqlState, NativeError, ErrorMsg);
        fprintf (stdout, "Connection failed!\n");
        return (1);
    }

    fprintf (stdout, "Connection Successful\n");
    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

Example 3-31 shows the output of Example 3-30 on page 112. In addition to the ODBC SQLSTATE error, it returns the Informix native error and the description for the error.

Example 3-31 Output of the connect_error.c application

```
C:\work>cl /DWIN32 /D_CRT_SECURE_NO_DEPRECATE -I%INFORMIXDIR%\incl\cli
odbc32.lib /nologo connect_error.c
connect_error.c

C:\work>connect_error wrongDNS
SqlState = IM002
 Native Error = 0
 Error Message = [Microsoft][ODBC Driver Manager] Data source name not found
and no default driver specified
Connection failed!
```

```
C:\work>connect_error demo_on;UID=wronguser
SqlState = 28000
 Native Error = -951
 Error Message = [Informix][Informix ODBC Driver][Informix]Incorrect password
or user wronguser@localhost is not known on the database server.
Connection failed!

C:\work>
```

Because this function is called quite often, it make sense to have a function to display the error message. Example 3-32 illustrates a typical error handling function.

Example 3-32 A sample simple_select_werr.c application

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

#ifdef WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif

#include "infxcli.h"

void CheckDiag (SQLSMALLINT handle_type, SQLHANDLE handle, char *text)
{
    RETCODErc = SQL_SUCCESS;
    UCHARSqlState[200] = "", ErrorMsg[200] = "";
    SQLINTEGERIsamError = 0;
    SDWORDNativeError = 0L;
    SWORDErrorMsgp = 0;
    SQLSMALLINTrecnum = 1;

    fprintf (stdout,"Error in %s \n",text);
    while (rc != SQL_NO_DATA_FOUND) {
        rc = SQLGetDiagRec(handle_type, handle, recnum, SqlState, &NativeError,
ErrorMsg, 199, &ErrorMsgp);
        if (rc != SQL_NO_DATA_FOUND) {
            SQLGetDiagField(handle_type, handle, recnum, SQL_DIAG_ISAM_ERROR,
&IsamError, SQL_IS_INTEGER, NULL);
             fprintf (stdout," SqlState = %s\n Native Error = %d\n Error Message = %s\n
ISAM Error = %d\n", SqlState, NativeError, ErrorMsg, IsamError);
            }
        recnum++;
    }
}
int main (long argc, char* argv[])
{
    SQLHDBC         hdbc;
    SQLHENV         henv;
    SQLHSTMT        hstmt;

    SQLRETURN       rc = 0;
    SQLCHAR         connStrIn[1000];
    SQLCHAR         connStrOut[1000];
    SQLSMALLINT     connStrOutLen;

    int             sqllen;
    SQLCHAR         *sqlstmt;

    if (argc != 3)
    {
        fprintf (stdout, "Please specify the name of a DSN and the SQL Statement to
run!\n");
    return(1);
    }
    else
    {
    if (strstr (argv[1],"DRIVER")==NULL)
       sprintf((char *) connStrIn, "DSN=%s;", (char *)argv[1]);
    else
       sprintf((char *) connStrIn, "%s;", (char *)argv[1]);

        sqllen  = strlen((char *)argv[2]);
        sqlstmt = (SQLCHAR *) malloc (sqllen + sizeof(char));
        strcpy((char *)sqlstmt, (char *)argv[2]);

    }

    SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
    SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    SQLDriverConnect (hdbc, NULL, connStrIn, SQL_NTS, connStrOut, 1000, &connStrOutLen,
SQL_DRIVER_NOPROMPT);
    fprintf (stdout, "Connected\n");
    SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );

    rc = SQLExecDirect (hstmt, sqlstmt, SQL_NTS);
    if (rc != SQL_SUCCESS &&  rc != SQL_SUCCESS_WITH_INFO)
    {
    CheckDiag(SQL_HANDLE_STMT, hstmt,"SQLExecDirect()");
        fprintf (stdout, "SQLExecDirectW() failed!\n");
        return (1);
    }
    fprintf (stdout, "Executed SQL Statement:\n%s\n",sqlstmt);

    SQLDisconnect (hdbc);
    SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
    SQLFreeHandle (SQL_HANDLE_ENV, henv);

    return (rc);
}
```

We create a function called `CheckDiag()` to retrieve the diagnostic information:

```
void CheckDiag (SQLSMALLINT handle_type, SQLHANDLE handle, char *text)
```

Example 3-33 shows the output of the previous sample.

Example 3-33 Output of the simple_select_werr.c application

```
C:\work>simple_sql_werror demo_on "wrong sql"
Connected
Error in SQLExecDirect()
 SqlState = 42000
 Native Error = -201
 Error Message = [Informix][Informix ODBC Driver][Informix]A syntax error has
occurred.
 ISAM Error = 0
SQLExecDirectW() failed!

C:\work>simple_sql_werror demo_on "DELETE from wrongtable"
Connected
Error in SQLExecDirect()
 SqlState = 42S02
 Native Error = -206
 Error Message = [Informix][Informix ODBC Driver][Informix]The specified table
(wrongtable) is not in the database.
 ISAM Error = -111
SQLExecDirectW() failed!

C:\work>
```

Note: If an error is generated by the database server, the error message contains the string `[Informix]` before the description of the error. For example:

```
Error in SQLExecDirect()
SqlState = 42S02
 Native Error = -206
 Error Message = [Informix][Informix ODBC Driver][Informix]The specified
table (test) is not in the database.
 ISAM Error = -111
SQLExecDirectW() failed!
```

ODBC SQLSTATE errors with Informix ODBC Driver

The error that is returned by the ODBC function is based on the X/Open standard SQLSTATE errors. For the SQLSTATE codes, refer to the *Informix ODBC Driver Guide* at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.odbc.doc/sii091033222.htm#sii091033222

3.3.6 Troubleshooting

In this section, we discuss typical problems when using the Informix ODBC drivers and available traces.

Environment

Make sure the setup of Informix Client is correct. If your application fails to load the ODBC driver, verify that the Informix Client or Data Server Client libraries are accessible to the application.

- The environment variable PATH should contain the `bin` directory of your Client package.
- The shared library PATH variable on a UNIX system should contain the directory where the shared libraries are located.
- Make sure that the INFORMIXDIR variable is set correctly (as an environment variable or stored in the registry using the `setnet32.exe` utility).
- Any 32-bit applications require 32-bit drivers to work. Make sure that the drivers that you have installed are of the same type as your application.

If your application fails to connect, make sure the connection details are valid.

Client SDK on Windows contains a simple connection tool called `iLogin`, which you can use to check whether you have connection with the database server. If `iLogin` fails to connect, all the other drivers will fail also.

- Check that the values stored in the registry with the `setnet32.exe` utility are valid.
- If you use 32-bit and 64-bit drivers on the same system, remember that on a Windows system, there are two registry hives to store the connectivity information. Make sure that both are correct.

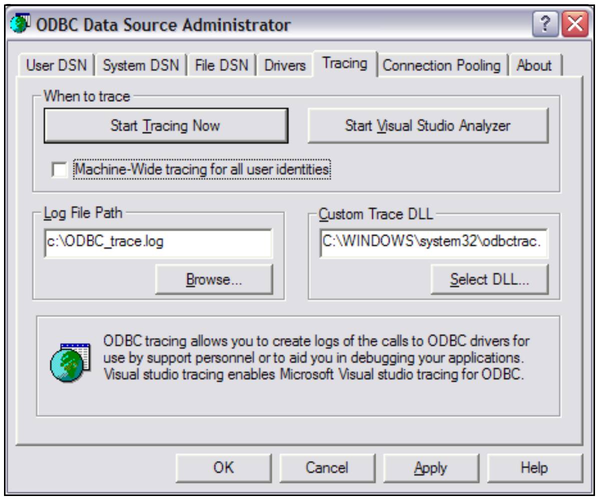
Tracing

When developing an application using the ODBC driver, most problems are caused by passing incorrect parameters to the ODBC functions.

ODBC Trace

You can generate a trace file of all the calls to the ODBC driver using the ODBC Trace facility. On a Windows system, you can enable ODBC Trace using the ODBC Data Source Administrator.

Figure 3-14 shows the Tracing tab.


ODBC Data Source Administrator

User DSN | System DSN | File DSN | Drivers | Tracing | Connection Pooling | About

When to trace

Start Tracing Now | Start Visual Studio Analyzer

☐ Machine-Wide tracing for all user identities

Log File Path: c:\ODBC_trace.log

Browse...

Custom Trace DLL: C:\WINDOWS\system32\odbctrac.

Select DLL...

ODBC tracing allows you to create logs of the calls to ODBC drivers for use by support personnel or to aid you in debugging your applications. Visual studio tracing enables Microsoft Visual studio tracing for ODBC.

OK | Cancel | Apply | Help

Figure 3-14 Enabling trace

Note: Remember to select **Machine-Wide tracing for all user identities** if your application runs as a service or with a different user.

Example 3-34 shows some of the entries in an ODBC trace file.

Example 3-34 ODBC trace

```
...
rccreate        99c-db8 EXIT  SQLBindParameter  with return code 0
(SQL_SUCCESS)
                HSTMT               01D51DC0
                UWORD                        2
                SWORD                        1 <SQL_PARAM_INPUT>
                SWORD                        1 <SQL_C_CHAR>
                SWORD                        1 <SQL_CHAR>
                SQLULEN                   25
                SWORD                        0
                PTR                0x002CFEC8
                SQLLEN                     0
                SQLLEN *            0x002CFF50 (-3)

rccreate        99c-db8 ENTER SQLBindParameter
                HSTMT               01D51DC0
                UWORD                        3
                SWORD                        1 <SQL_PARAM_INPUT>
                SWORD                        5 <SQL_C_SHORT>
                SWORD                        5 <SQL_SMALLINT>
                SQLULEN                    0
                SWORD                        0
                PTR                0x002CFF68
                SQLLEN                     0
                SQLLEN *            0x002CFF54
...
```

On a UNIX system, ODBC trace is active if the Trace parameter in the `[ODBC]` section of the `odbc.ini` configuration file is set to `1`.

Example 3-35 shows the Trace settings for the `odbc.ini` file.

Example 3-35 Trace in the odbc.ini file

```
;
; Trace file Section
;
Trace=1
TraceFile=/tmp/odbctrace.out
InstallDir=/opt/IBM/informix
TRACEDLL=idmrs09a.so
```

SQLIDEBUG

All the Informix clients use the SQLI protocol to communicate with the database server.

In addition to the ODBC trace, you can generate an SQLIDEBUG trace that contains all the message between the client application and the database server.

You can enable SQLIDEBUG trace at the Informix Client side by defining the environment variable as follows:

```
SQLIDEBUG=2:path_to_trace_files
```

Alternatively, you can enable trace at the server side by running the following command:

```
onmode -p 1 sqli_dbg
```

Note: On a Windows system, when using the server side tracing, the SQLIDEBUG files are created in the `C:\temp\sqli` directory. This directory must exist before you can enable the trace.

On a UNIX system, the trace files are created in the `/tmp/sqli` directory.

Example 3-36 demonstrates how to set SQLIDEBUG trace on the client.

Example 3-36 SQLIDEBUG sample

```
C:\work>set SQLIDEBUG=2:sqli_trace

C:\work>simple_sql_werror demo_on "wrong_sql"
Connected
Error in SQLExecDirect()
 SqlState = 42000
 Native Error = -201
 Error Message = [Informix][Informix ODBC Driver][Informix]A syntax error has
occurred.
 ISAM Error = 0
SQLExecDirectW() failed!

C:\work>dir sqli_trace_2940_1016_1f428b8
 Volume in drive C is W2003
 Volume Serial Number is 50DA-70D7

 Directory of C:\work

22/06/2010  18:39               348 sqli_trace_2940_1016_1f428b8
               1 File(s)            348 bytes
               0 Dir(s)  76,628,099,072 bytes free

C:\work>
```

The SQLIDEBUG trace files contains the SQLI packages between the server and the client. To obtain an readable output, you must run the `sqliprt` tool.

Example 3-37 demonstrates how to run `sqliprt`.

Example 3-37 The sqliprt output

```
C:\work>sqliprt -o trace.txt sqli_trace_2940_1016_1f428b8

C:\work>type trace.txt
SQLIDBG Version 1

C->S (4)                                Time: 2010-06-22 18:39:05.43700
        SQ_INTERNALVER
                Internal Version Number: 316

C->S (14)                               Time: 2010-06-22 18:39:05.43700
        SQ_PROTOCOLS
        SQ_EOT

S->C (14)                               Time: 2010-06-22 18:39:05.43700
        SQ_PROTOCOLS
        SQ_EOT

C->S (90)                               Time: 2010-06-22 18:39:05.43700
        SQ_INFO
                INFO_ENV
                        Name Length = 12
                        Value Length = 34
                        "DBTEMP"="C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp"
                        "SUBQCACHESZ"="10"
                        "OPTOFC"="0"
                INFO_DONE
        SQ_EOT

S->C (2)                                Time: 2010-06-22 18:39:05.43700
        SQ_EOT

C->S (4)                                Time: 2010-06-22 18:39:05.43700
        SQ_BEGIN
        SQ_EOT

S->C (10)                               Time: 2010-06-22 18:39:05.43700
        SQ_XACTSTAT
        SQ_EOT

C->S (4)                                Time: 2010-06-22 18:39:05.43700
        SQ_CMMTWORK
        SQ_EOT

S->C (10)                               Time: 2010-06-22 18:39:05.43700
        SQ_XACTSTAT
        SQ_EOT

C->S (22)                               Time: 2010-06-22 18:39:05.43700
        SQ_PREPARE
                # values: 0
                CMD.....: "wrong_sql" [9]
        SQ_NDESCRIBE
        SQ_WANTDONE
        SQ_EOT

S->C (12)                               Time: 2010-06-22 18:39:05.43700
        SQ_ERR
                SQL error..........: -201
                ISAM/RSAM error....: 0
                Offset in statement: 1
                Error message......: "" [0]
        SQ_EOT

C:\work>
```

Note: On a UNIX system, the equivalent of the `sqliprt` tool is `sqliprint`.

DRDADEBUG

All IBM Data Server drivers use the DRDA protocol to communicate with the database server.

Similar to SQLIDEBUG, you can enable DRDADEBUG trace at the server side by running the following `onmode` command:

```
onmode -p 1 drda_dbg
```

The DRDA trace files are created in the `/tmp/drda` directory for a UNIX system and the `C:\temp\drda` directory for a Windows system. You need to use the `drdaprint` tool to convert the trace files to a readable format.

Example 3-38 shows the use of `drdaprint`.

Example 3-38 Using the drdaprint tool

```
C:\temp\drda>dir
 Volume in drive C is W2003
 Volume Serial Number is 50DA-70D7

 Directory of C:\temp\drda

22/06/2010  18:50    <DIR>          .
22/06/2010  18:50    <DIR>          ..
22/06/2010  18:50             2,269 drda.47
               1 File(s)          2,269 bytes
               2 Dir(s)  76,628,135,936 bytes free

C:\temp\drda>drdaprint
Usage: drdaprint [-f] [-o outfile] inpfile
        -f: format hex dump

C:\temp\drda>drdaprint -o trace.txt drda.47

C:\temp\drda>
```

Example 3-39 shows a section of the DRDA trace file.

Example 3-39 DRDATrace file

```
C:\temp\drda>type trace.txt
DRDADBG Version 1

1       data IDS DRDA Communication Manager
        function  sqljcIntReceive()
        bytes 270
        time  2010-06-22 18:50:13.31200

        RECEIVE BUFFER(AS):
            EXCSAT RQSDSS                      (ASCII)           (EBCDIC)
         0 1 2 3 4 5 6 7  8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
  0000  00C3D041000100BD 10410080115EA289   ...A.....A...^..  .C}..........;si
  0010  949793856DA29893 6DA685999996994B   ....m...m......K  mple_sql_werror.
  0020  85A7F0F6F1F4F0C4 F9F4F0F0F0000000   ................  ex06140D94000...
  0030  0000000000000000 0000000000000000   ................  ................
  0040  0000000000000000 000000000060F0F0   .............`..  .............-00
  0050  F0F1C1C4D4C9D5C9 E2E3D9C1E3D6D940   ...............@  01ADMINISTRATOR
  0060  4040404040404040 4040404040404040   @@@@@@@@@@@@@@@@
  0070  E2E3D6D9C5E26DC4 F0C4C2F240404040   ......m.....@@@@  STORES_D0DB2
  0080  4040404040404040 40F0001814041403   @@@@@@@@@.......           0......
  0090  000A2407000A1474 0005240F00081440   ..$....t..$....@  ...............
  00A0  0009000B1147D8C4 C2F261D5E3000A11   .....G....a.....  ......QDB2/NT...
  00B0  6DC4E4C2C9E3D600 0C115AE2D8D3F0F9   m.........Z.....  _DUBITO...]SQL09
  00C0  F0F7F0                              ...               070
```

4

Working with ESQL/C

This chapter discusses ESQL/C, an SQL application programming interface (API) that enables you to embed Structured Query Language (SQL) statements directly into a C program. ESQL/C is bundled with the Informix Client Software Development Kit (Client SDK). The ESQL/C preprocessor, `esql`, converts each SQL statement and all IBM Informix-specific code to C-language source code and invokes the C compiler to compile it.

The advantage of using ESQL/C is that it supports all the data types as well as extended data types of Informix and is optimized for an IBM Informix database. If you are using only Informix as your database server, then using ESQL/C might be your best choice.

In this chapter, we discuss the basic elements of an Informix ESQL/C application and show how to perform database operations within an ESQL/C program. We end the chapter by looking at various methods in handling exceptions and troubleshooting.

This chapter includes the following topics:

- Informix ESQL/C
- Setup and configuration
- Windows system configuration
- Developing an ESQL/C application

4.1 Informix ESQL/C

Informix ESQL/C includes the following software components:

- The Informix ESQL/C libraries of C functions, which provide access to the database server
- The Informix ESQL/C header files, which provide definitions for the data structures, constants, and macros that are useful to the Informix ESQL/C program
- The **`esql`** command, which processes the Informix ESQL/C source code to create a C source file that it then passes to the C compiler
- The `finderr` utility on the UNIX system and the Informix Error Messages Windows-based Informix error messages utility that enable you to obtain information about IBM Informix-specific error messages

On Windows platforms, Informix provides the following additional utilities:

- The `setnet32.exe` utility, which is a Windows-based utility that enables you to set configuration information
- The `iLogin` utility, which is a demonstration program that displays a dialog box with fields for the connection parameters and for testing a connection to the database server (uses the `stores7` database)

Figure 4-1 gives an overview of the relationship between ESQL/C and the native C and illustrates the process of the transfer of control.

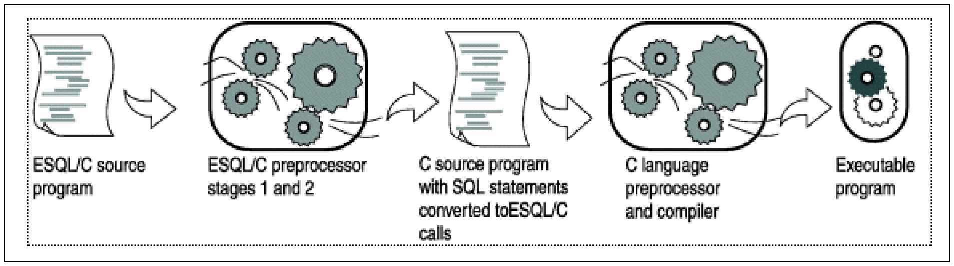

ESQL/C source program
ESQL/C preprocessor stages 1 and 2
C source program with SQL statements converted toESQL/C calls
C language preprocessor and compiler
Executable program

Figure 4-1 Relationship between Informix ESQL/C and C

When you have created an Informix ESQL/C program file, you run the **`esql`** command on that file. By default, the Informix ESQL/C preprocessor runs first and translates the embedded SQL statements in the program into Informix ESQL/C function calls that communicate with the database server. The Informix ESQL/C preprocessor produces a C source file and calls the C compiler. The C compiler then compiles your source file and links any other C source file, object file, or library file the same way as any other C program.

If the `esql` command does not encounter errors in one of these steps, it generates an executable file. You can run the compiled Informix ESQL/C program as you would run any C program. When the program runs, it calls the Informix ESQL/C library procedures. The library procedures set up communications with the database server to carry out the SQL operations.

4.2 Setup and configuration

Informix ESQL/C is installed, by default, as a part of Client SDK. The release notes for Client SDK contain the information about supported versions of the IBM Informix database server. Make sure that you install the version of Client SDK that is supported by the database with which you are working.

4.3 Windows system configuration

On a Windows system, the default Client SDK installation directory is `C:\Program Files\IBM\Informix\Client-SDK`. Make sure that the INFORMIXDIR environment variable is pointing to the directory where the product was installed. The PATH environment variable needs to contain the following directories:

- The path to the `bin` directory that is under the installation directory needs to be defined (that is, the PATH variable needs the `$INFORMIXDIR/bin` directory defined).
- The path where native C compiler is located. Informix Client products are certified with the Microsoft Visual C++ 2005 SP1.

UNIX configuration

On UNIX platforms, the default installation directory for Client SDK is `/opt/IBM/informix`. The INFORMIXDIR environment variable needs to point to the directory where the product was installed. Add the following directories to the PATH environment variable:

- The `bin` directory under the installation directory, `$INFORMIXDIR/bin`
- The path to the native C compiler

The `sqlhosts` file contains the information that is required to connect to an IBM Informix database server. You must set this file to include an entry for your Informix database server name, ESQL/C required protocol, host name, and port number. By default, the `sqlhosts` file is under the `$INFORMIXDIR/etc/` directory. You can use the INFORMIXSQLHOSTS environment variable to point to a different location.

Example 4-1 shows a simple `sqlhosts` file.

Example 4-1 The sqlhosts file

```
#server protocol hostname service/port
demo_on onsoctcp kodiak 9088
```

For more information about the `sqlhosts` file, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.admin.doc/ids_admin_0158.htm

4.4 Developing an ESQL/C application

In this section, we describe how to connect to a database, how to perform basic database operations such as delete and insert, and how to update using ESQL/C. We also describe the handling of extended data types by ESQL/C.

We use the sample instance and database that Informix Server provides to demonstrate ESQL/C application development. The sample instance, `demo_on`, is created when the Informix server is installed. On a Windows operating system, the sample instance is `ol_svr_custom`. You can create a sample database, `stores_demo`, under the `demo_on` instance and populate that sample database with tables and data by running the `dbaccessdemo` utility that resides in the `$INFORMIXDIR/bin/` directory.

4.4.1 Creating an ESQL/C application

To build an Informix ESQL/C application:

1. Develop a C program with the embedded Informix SQL statements, and name the program with a `.ec` or `.ecp` extension. The SQL statements are qualified with an EXEC SQL keyword or the dollar sign ($) symbol as shown in the following example:

   ```
   EXEC SQL include sqlca;
   EXEC SQL SELECT fname into :c1 FROM customer WHERE customer_num=:i1
   $SET ISOLATION DIRTY READ;
   ```

2. Preprocess the Informix ESQL/C source file with the **`esql`** command. The **`esql`** command also invokes the C compiler to compile the program into object code.

   As necessary, correct errors that are reported by the preprocessor and the compiler and then repeat this step.

3. Link the object code into one or more executable files using the **esql** command. The executable files have a .exe extension.
4. Run the application.

   You run the compiled Informix ESQL/C program as you would any C program. When the program runs, it calls the Informix ESQL/C library procedures. The library procedures set up communications with the database server to carry out the SQL operations.

Example 4-2 shows a simple program, customer.ec, that connects to the database and retrieves data. All the SQL-related statements are embedded in the C program with the EXEC SQL keyword.

Example 4-2 Simple ESQL/C program

```
#include <stdio.h>
EXEC SQL include sqlca;
EXEC SQL include sqltypes;

int main()
{
EXEC SQL BEGIN DECLARE SECTION;
int i1=101;
char c1[50];
char c2[50];
EXEC SQL END DECLARE SECTION;
int i2;

EXEC SQL connect to 'stores_demo';

EXEC SQL select fname,lname  into :c1, :c2  from customer where
customer_num=:i1
;
if (SQLCODE == 100)
  {
  printf("SQLCODE=%d\n",SQLCODE);
  printf("Data not found\n");
  }
else
  {
  printf("SQLCODE=%d\n",SQLCODE);
  printf("Data found \n");
  printf("Last Name  \t%s\n",c1);
  printf("First Name  \t%s\n",c2);
  }
}
```

Compiling ESQL/C programs

The **esql** command translates Informix ESQL/C code to C code and then calls the C compiler to compile and link the C code.

The C compiler takes the following actions:

1. Compiles the C language statements to object code.
2. Links to Informix ESQL/C libraries and to any other files or libraries that you specify.
3. Creates an executable file.

The general syntax for an **esql** command to compile is as follows:

```
esql <options> <source.ec> <options> <outfile>
```

By default, the **esql** command creates an executable called `a.out` in the current directory. You can explicitly specify the name of the executable file with the **-o** option. For example, the following command compiles the `customer.ec` executable file shown in Example 4-2 on page 129 and produces the executable file `customer.exe`:

```
esql -o customer.ec customer.exe
```

If you run the **esql** command on a Windows operating system, the name of the target file defaults to the name of the first Informix ESQL/C source file on the **esql** command line. The extension is changed to either `.exe` or `.dll`, depending on the type of target that is generated.

You can use a compiler other than the native C compiler by setting the INFORMIXC environment variable. Table 4-1 lists the native C compilers on various platforms.

Table 4-1 Native C compiler

Platforms	Native compiler
Solaris	CC
HP	aC++
Windows	VC++ / VS2008/ VS2005
AIX	xlc
Open source	gcc/g++

If you want to pass C compiler options that have the same names as Informix ESQL/C processor options, precede them with the **-cc** option. For example, the following **esql** command passes the **-od** and **-g** options to the C compiler but uses the **-db2** option itself:

```
esql -cc -od -g demo1.ec -db2
```

Shared libraries

IBM Informix products use the Informix general libraries for interactions between the client SQL API products (IBM Informix ESQL/C and IBM Informix ESQL/COBOL) and the database server. You can choose between the following types of Informix general libraries to link with your Informix ESQL/C application:

- Static Informix general libraries

  To link a static library, the linker copies the library functions to the executable file of your Informix ESQL/C program. The static Informix general libraries allow an Informix ESQL/C program on computers that do not support shared memory to access the Informix general library functions.

  To link static libraries use the **-static** option, for example:

  ```
  esql -static file.ec -o file.exe
  ```

- Shared Informix general libraries

  To link a shared library, the linker copies information about the location of the library to the executable file of your Informix ESQL/C program. The shared Informix libraries allow several applications to share a single copy of these libraries, which the operating system loads just once into shared memory.

- Thread-safe versions of static and shared Informix general libraries

  The thread-safe versions of Informix general libraries allow an Informix ESQL/C application that has several threads to call these library functions simultaneously. The thread-safe versions of Informix libraries are available as both static libraries and shared libraries.

There are some platform specific considerations when you link shared Informix general libraries to an ESQL/C module. The environment variable that specifies the search path at run time is different depending on the platform, as listed in Table 4-2.

Table 4-2 Environment variable for shared libraries

Platform	**Environment variable**
AIX	LIBPATH
Solaris	LD_LIBRARY_PATH
HP-UX	SHLIB_PATH
Mac OS X	DYLD_LIBRARY_PATH

Platform	Environment variable
Linux	LD_LIBRARY_PATH
Windows	LIB

Set the `$INFORMIXDIR/lib` directory and any of its subdirectories to specify the shared-library path. For example, on Linux, set the LD_LIBRARY_PATH environment variable as follows:

- Bourne shell

```
LD_LIBRARY_PATH=$INFORMIXDIR/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

- C shell

```
setenv LD_LIBRARY_PATH $INFORMIXDIR/lib:$LD_LIBRARY_PATH
```

On a Windows system, use the following command:

```
set LIB = %INFORMIXDIR%\lib\;%LIB%
```

To link shared Informix general libraries with an Informix ESQL/C module, you do not need to specify a command-line option. Informix ESQL/C links shared libraries by default. The following command compiles the `file.ec` source file with shared Informix libraries:

```
esql myfile.ec -o myfile.exe
```

Choosing between shared and static library versions

Shared libraries are most useful in multiuser environments where only one copy of the library is required for all applications. Shared libraries bring the following benefits to your Informix ESQL/C application:

- Shared libraries reduce the size of executable files because these library functions are linked dynamically on an as-needed basis.
- At run time, a single copy of a shared library can be linked to several programs, which results in less memory use.
- The effects of shared libraries in an Informix ESQL/C executable are transparent to the user.

Although shared libraries save both disk and memory space, when an Informix ESQL/C application uses them, it must perform the following overhead tasks:

- Dynamically load the shared library into memory for the first time
- Perform link-editing operations
- Execute library position-independent code

These overhead tasks can incur runtime penalties and are not necessary when you use static libraries.

4.4.2 Performing database operations

In this section, we discuss how to perform basic database operations in an ESQL/C application:

- Database connections
- Simple SQL statements (SELECT, INSERT, UPDATE, and DELETE)
- Static and dynamic SQL
- Calling SQL routines (stored procedures)
- Using transactions

In the program, we try to make use of most commonly used data types. For information about other data types, refer to:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.esqlc.doc/sii-03-sourceforchaptitle.htm`

Database connections

When an Informix ESQL/C application executes, it has no connections to any database server. For SQL statements to run, however, such a connection must exist. To establish a connection to a database server, the Informix ESQL/C program must take the following actions:

1. Use an SQL statement to establish a connection to the database server.
2. Specify, in the SQL statement, the name of the database server to which to connect.

The client application connects to the default database server when the application does not explicitly specify a database server for the connection. You must set the INFORMIXSERVER environment variable even if the application does not establish a connection to the default database server.

If user name and password are not explicitly specified using the `InetLogin` structure or the USER clause, the default user ID is used to attempt the connection. The default user ID is the login name of the user who is running the application.

Example 4-3 shows a statement that connects to the `stores_demo` database under the `demo_on` instance.

Example 4-3 Database connection statement

```
EXEC SQL connect to 'stores_demo@demo_on';
EXEC SQL connect to stores_demo user :username using :password;
```

For more information about database connections in ESQL/C, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.esqlc.doc/sii12164286.htm#sii12164286

Simple SQL statements

There are several way to execute an SQL statement from an ESQL/C application. The application must choose an appropriate method based on the nature of the SQL statement.

- SQL statements that do not return a result set of data such as INSERT, DELETE, UPDATE, or Data Definition Language (DDL) can be executed using the EXEC SQL command (same as the `$` prefix) or the `EXEC SQL execute immediate` statement. Both methods produce the same C code.

  ```
  EXEC SQL execute immediate :cmdstring;
  $CREATE TABLE my_customer (fname char(20));
  ```

  If the SQL statement is going to be executed more than one time during the life of the application, it can be prepared with the EXEC SQL PREPARE command and then executed when required using the EXEC SQL EXECUTE command:

  ```
  EXEC SQL prepare d_id from :stmt_buf;
  EXEC SQL execute d_id;
  ```

- SQL statements that return one row can be executed using the EXEC SQL EXECUTE INTO command. These types of statement are normally referred as *singleton* statements.

  ```
  EXEC SQL SELECT specs into :mspecs FROM my_customer WHERE customer_num=2;
  ```

- SQL statements that return more than one row are executed using a select cursor:

  ```
  EXEC SQL declare cursor1 cursor for SELECT fname into :var1 from customer;
  EXEC SQL open cursor1;
  EXEC SQL fetch cursor1;
  ```

Example 4-4 demonstrates how to execute an SQL statement using some of these previously described methods.

Example 4-4 Simple SQL statements (INSERT, UPDATE, and DELETE)

```
#include <stdio.h>
#include <string.h>
EXEC SQL include sqlca;
EXEC SQL include sqltypes;

int main()
{
EXEC SQL BEGIN DECLARE SECTION;
char cmdstring[4096];
string  *fname[5]       = { "Ludwig","Carole", "Philip", "Anthony", "Raymond" };
string  *lname[5]       = { "Pauli", "Sadler", "Currie", "Higgins", "Vector" };
string  *company[5]     = {"All Sports Supplies","Sports Spot", "Phil's Sports", "Play
Ball!", "Los Altos Sports" };
char pref='t';
char  *specs[5]         = { "This is just any string ", "This is just another string ",
"This is the third string", "This is one more string", "This is the last string"};
lvarchar  mspecs[250];

char *order_date[5]     = {"08/01/77","08/02/77","08/03/77","08/04/77","08/05/77"};
float ship_charge[5]    = {10000.59,590000.32,345577.12,987098.32,876893.22};
char *ship_duration[5]  = {"10:10:10","11:11:11","22:22:22","33:33:33","44:44:44"};

EXEC SQL END DECLARE SECTION;
$WHENEVER ERROR STOP;
int i;

EXEC SQL connect to 'stores_demo';

$ CREATE TABLE my_customer (
customer_num        bigserial,
fname               char(15),
lname               char(15),
company             char(30),
preferred           boolean,
specs               lvarchar
);

$ CREATE TABLE my_orders (
order_num           bigserial,
order_date          date ,
customer_num        bigint,
ship_charge         money(10,2),
ship_duration       INTERVAL hour to second
);

for (i = 0; i < 5 ; i++)
  {
  $INSERT INTO my_customer VALUES (
  0,
  :fname[i],
  :lname[i],
  :company[i],
  :pref,
  :specs[i]
  );
  }
for (i = 0; i < 5 ; i++)
  {
  sprintf(cmdstring, "INSERT INTO my_orders VALUES (10000001,'%s',%d,%f,'%s' );",
order_date[i], i,ship_charge[i], ship_duration[i]);
  EXEC SQL execute immediate :cmdstring;
  }

sprintf(cmdstring, "UPDATE my_customer SET specs='This is a new spec' WHERE customer_num
= 2");
EXEC SQL execute immediate :cmdstring;
if (SQLCODE != 0)
  printf("SQLCODE=%d\n",SQLCODE);

sprintf(cmdstring, "DELETE  FROM my_customer where customer_num=3");
EXEC SQL execute immediate :cmdstring;
if (SQLCODE != 0)
  printf("SQLCODE=%d\n",SQLCODE);

EXEC SQL SELECT specs into :mspecs FROM my_customer WHERE customer_num=2;
if (SQLCODE != 0)
  printf("SQLCODE=%d\n",SQLCODE);
else
 printf("The spec for the customer is :%s\n", mspecs);
}
```

For additional information regarding the execution of SQL statements, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.esqlc.doc/esqlc298.htm

Static and dynamic SQL

The SQL statements shown in Example 4-4 on page 135 are all *static* SQL statements. With a static SQL statement, all the information that is needed is known at compile time. However, in some applications the programmer does not know the contents or possibly even the types of SQL statements that the program needs to execute. For example, a program might prompt the user to enter a SELECT statement, so that the programmer has no idea what columns are accessed when the program runs.

Such applications require *dynamic* SQL statements. Dynamic SQL statements allow an IBM Informix ESQL/C program to build an SQL statement at run time, so that the contents of the statement can be determined by user input.

Example 4-5 shows how to use dynamic SQL in a program.

Example 4-5 Dynamic SQL statements

```
#include <stdio.h>
EXEC SQL include sqlca;
EXEC SQL include sqltypes;

int main()
{
EXEC SQL BEGIN DECLARE SECTION;
int i1;
char cmdstring[2048];
char c1[50];
char c2[50];
EXEC SQL END DECLARE SECTION;
int i2;

EXEC SQL connect to 'stores_demo';

printf("Enter the customer number [101-128] to see the names:");
scanf("%d", &i1);
sprintf (cmdstring, "SELECT fname,lname from customer where customer_num = ?");
EXEC SQL prepare ex_id  from  :cmdstring;
EXEC SQL execute ex_id into :c1,:c2 using  :i1;
;
if (SQLCODE == 100)
{
printf("SQLCODE=%d\n",SQLCODE);
printf("Data not found\n");
}
else
{
printf("SQLCODE=%d\n",SQLCODE);
printf("Data found \n");
printf("Last Name  \t%s\n",c1);
printf("First Name  \t%s\n",c2);
}
}

OUTPUT:
Enter the customer number [101-128] to see the names:102
SQLCODE=0
Data found
Last Name       Carole
First Name      Sadle
```

The SELECT statement shown in Example 4-5 on page 137 is called a *singleton* SELECT statement because it returns only one row. For SELECT statements that return multiple rows, you must use a SELECT cursor statement. Example 4-6 shows how to use cursor for SELECT statements that return multiple rows.

Example 4-6 Using cursor for SELECT statements that return multiple rows

```
#include <stdio.h>
EXEC SQL include sqlca;
EXEC SQL include sqltypes;

int main()
{
EXEC SQL BEGIN DECLARE SECTION;
int  i;
char cmdstring[2048];
char c1[50];
char c2[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL connect to 'stores_demo';

printf("Enter customer number [103 - 128] :");
scanf("%d", &i);
sprintf (cmdstring, "SELECT fname,lname from customer where customer_num < ?;");
EXEC SQL prepare ex_id  from  :cmdstring;
EXEC SQL declare ex_cursor cursor for ex_id;
EXEC SQL open ex_cursor using :i;

/* Print out what DESCRIBE returns*/
for (;;)
{
EXEC SQL fetch ex_cursor into :c1,:c2;

if (strncmp(SQLSTATE, "00", 2) != 0)
break;
if (SQLCODE == 100)
{
printf("SQLCODE=%d\n",SQLCODE);
printf("Data not found\n");
}
else
{
printf("%s\t%s\n",c1,c2);
}
}

OUTPUT:
Enter customer number [103 - 128] : 108
Ludwig Pauli
Carole Sadler
Philip Currie
```

```
Anthony Higgins
Raymond Vector
George Watson
Charles Ream
```

For more details about dynamic SQL and the cursors, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.esqlc.doc/sii-sc3dysq-35492.htm#sii-sc3dysq-35492

Calling SQL routines

In this section, we demonstrate how to call an SQL routine from an ESQL/C application. You can accomplish a wide range of objectives with SQL routines, including improving database performance, simplifying writing applications, and limiting or monitoring access to data. Because an SQL routine is stored in an executable format, you can use it for repeated tasks to improve performance. When you invoke an SPL routine, rather than straight SQL code, you can bypass repeated parsing, validity checking, and query optimization.

Example 4-7 shows a simple program that contains an SPL routine that receives arguments price and percent tax and that returns price after adding the tax.

Example 4-7 Calling SQL routines

```
#include <stdio.h>
$ INCLUDE sqlca.h;
$ INCLUDE sqltypes.h;

int main()
{
EXEC SQL BEGIN DECLARE SECTION;
char cmdstring[2048];
float n1,n2;
int i=2;
EXEC SQL END DECLARE SECTION;

EXEC SQL whenever sqlerror stop;
EXEC SQL CONNECT TO 'stores_demo';

sprintf(cmdstring, "CREATE FUNCTION inc_price(n1 money(8,2), n2 int) RETURNING
money(8,2) RETURN(n1 + (n1 *n2)/ 100); END FUNCTION;");
EXEC SQL execute immediate :cmdstring;

sprintf(cmdstring, "SELECT FIRST 5 total_price, inc_price(total_price,?) from items ;");
EXEC SQL prepare ex_id  from  :cmdstring;
EXEC SQL declare ex_cursor cursor for ex_id;
EXEC SQL open ex_cursor using :i;

/* Print out what DESCRIBE returns*/
for (;;)
{
```

```
EXEC SQL fetch ex_cursor into :n1,:n2;

if (strncmp(SQLSTATE, "00", 2) != 0)
break;
if (SQLCODE == 100)
{
printf("SQLCODE=%d\n",SQLCODE);
printf("Data not found\n");
}
else
{
printf("%f\t%f\n",n1,n2);
}
}
}

OUTPUT:
250.000000      255.000000
960.000000      979.200012
240.000000      244.800003
20.000000       20.400000
840.000000      856.799988
```

Using transactions

A *transaction* is a collection of SQL statements that are treated as a single unit of work. All the SQL statements that you issue in an ANSI-compliant database are contained in transactions automatically. With a database that is not ANSI compliant, transaction processing is an option.

In a database that is not ANSI compliant, a transaction is enclosed by a BEGIN WORK statement and a COMMIT WORK or a ROLLBACK WORK statement. In an ANSI-compliant database, the BEGIN WORK statement is unnecessary, because all statements are contained in a transaction automatically. You need to indicate only the end of a transaction with a COMMIT WORK or ROLLBACK WORK statement.

Example 4-8 shows an example using transactions.

Example 4-8 Using transactions

```
#include <stdio.h>
$include "sqlca.h";
$include "sqlhdr.h";
$include "sqltypes.h";

main()
{

EXEC SQL BEGIN DECLARE SECTION;
int n1;
char c1[20];
EXEC SQL END DECLARE SECTION;

EXEC SQL whenever sqlerror stop;

EXEC SQL CREATE DATABASE itso WITH LOG;

EXEC SQL CREATE TABLE t1(num serial, name char(20));
EXEC SQL BEGIN WORK;
EXEC SQL INSERT INTO t1 VALUES (0,'name1');
EXEC SQL INSERT INTO t1 VALUES (0,'name2');
EXEC SQL INSERT INTO t1 VALUES (0,'name3');
EXEC SQL ROLLBACK;
EXEC SQL BEGIN WORK;
EXEC SQL INSERT INTO t1 VALUES (0,'name4');
EXEC SQL INSERT INTO t1 VALUES (0,'name5');
EXEC SQL COMMIT;

EXEC SQL declare ifx_cursor cursor for select num,name into :n1,:c1 from t1;
EXEC SQL open ifx_cursor ;
for (;;)
{
EXEC SQL fetch ifx_cursor;
if (sqlca.sqlcode!=0)
break;
printf("%d\t%s\n",n1,c1);
}
EXEC SQL close ifx_cursor;
EXEC SQL free ifx_cursor;
}

OUTPUT:
4       name4
5       name5
```

4.4.3 Data types mapping

This section contains information about the correspondence data types between SQL and C and how to handle data types in an IBM Informix ESQL/C program.

When a query is executed by the application, the data that the Informix server returns might be in a different format than the format that the application uses. The ESQL/C converts the data passed between the application and the database server. This process is transparent to the application. The only requirement for the application is to specify the correct data types in ESQL/C.

Table 4-3 lists a few ESQL/C data type mapping as examples.

Table 4-3 Informix data type mapping

Informix SQL	ESQL/C
BOOLEAN	boolean
BYTE	loc_t
LVARCHAR	lvarchar
NCHAR(n)	fixchar [n] or string [n+1]
NVARCHAR(m)	varchar[m+1] or string [m+1]
SERIAL8	int8 or ifx_int8_t
TEXT	loc_t
BLOB	ifx_lo_t
CLOB	ifx_lo_t
LIST(e)	collection
MULTISET(e)	collection
Opaque data type	lvarchar, fixed binary, or var binary
ROW(...)	row
SET(e)	collection

For a complete list of all the Informix SQL to ESQL/C data type mapping, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.esqlc.doc/sii03147680.htm

4.4.4 Handling special data types

This section describe how to work with Informix specific data types. We discuss the following data types:

- Smart large objects (BLOB and CLOB)
- Collection data types (LIST, MULTISET, and SET)
- ROW data type

To store large objects inside database, you can use data types such as TEXT, BYTE, BLOB, and CLOBS. Informix supports simple large objects and smart large objects. Simple large objects are the TEXT and BYTE types that exist primarily for compatibility with earlier versions of Informix applications. The smart large objects are the BLOB and CLOBs. When you write new applications that need to access large objects, use smart large objects to hold character (CLOB) and binary (BLOB) data.

Table 4-4 summarizes the advantages that smart large objects present over simple large objects.

Table 4-4 Advantages of smart blobs over simple blobs

Large object feature	Simple large objects	Smart large objects
Maximum size of data	2 gigabytes	4 terabytes
Data accessibility	No random access to data	Random access to data
Reading the large object	The database server reads a simple large object on an all or nothing basis.	Library functions provide access that is similar to accessing an operating-system file. You can access specified portions of the smart large object.
Writing the large object	The database server updates a simple large object on an all or nothing basis.	The database server can rewrite only a portion of a smart large object.
Data logging	Data logging is always on.	Data logging can be turned on and off.

Example using CLOB and BLOBs

Informix ESQL/C supports the SQL data types CLOB and BLOB with the `ifx_lo_t` data type. Because of the potentially huge size of smart large object data, the Informix ESQL/C program does not store the data directly in a host variable. Instead, the client application accesses the data as a file-like structure. To use smart large object variables in an Informix ESQL/C program, take the following actions:

1. Declare a host variable with the `ifx_lo_t` data type.
2. Access the smart large object with a combination of the following three data structures:
   - The LO-specification structure, `ifx_lo_create_spec_t`
   - The LO-pointer structure, `ifx_lo_t`
   - An integer LO file descriptor

Smart large objects are stored logically in a table column but stored physically in a specific type of dbspaces called smart blob space (sbspace). You must create the sbspace in the server before you can run this example. For details about creating the sbspace, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.admin.doc/ids_admin_0491.htm

Example 4-9 illustrates how to work with BLOB and CLOB data types using ESQL/C functions. The `filetoclob` and `filetoblob` functions are used to enter the CLOB and BLOB data respectively. If your files are located on the client system, then use the client name as the second argument to the `filetoblob` function. If your files are located on the server system, then use the server name as the second argument.

The output is a part of the file that you insert as a BLOB. In this example, we used a `readme.txt` file.

Example 4-9 The ifx_lo_sample.ec file

```
#include <stdio.h>
$include "sqlca.h";
$include "sqlhdr.h";
$include "sqltypes.h";

main()
{
int error, ic1, oflags, cflags, extsz, imsize, isize, iebytes;
time_t time;
struct tm *date_time;
char col_name[300]="test", sbspc[129];

EXEC SQL BEGIN DECLARE SECTION;
 fixed binary 'blob' ifx_lo_t c2;
 char srvr_name[256];
 ifx_lo_create_spec_t *cspec;
 ifx_lo_stat_t *stats;
 ifx_int8_t size, c1, estbytes, maxsize;
 int lofd;
 long atime, ctime, mtime, refcnt;
EXEC SQL END DECLARE SECTION;


EXEC SQL whenever sqlerror stop;
EXEC SQL connect to 'stores_demo';

EXEC SQL create table t2  (c1 int, c2 blob);
EXEC SQL insert into t2 values (1,filetoblob ('/tmp/README.txt', 'server','t2','c2'));

EXEC SQL declare ifxcursor cursor for select c1,c2 into :c1,:c2  from t2 for update;

EXEC SQL open ifxcursor;
for (;;)
 {
```

```
 EXEC SQL fetch ifxcursor;
 if (sqlca.sqlcode!=0)
   break;
 lofd = ifx_lo_open(&c2, LO_RDWR, &error);
 ifx_lo_read(lofd, srvr_name, 256, &error);
 printf("Value: %s\n",srvr_name);
 ifx_lo_write(lofd,col_name,5,&error);

 ifx_lo_close(lofd);
}

EXEC SQL close ifxcursor;
EXEC SQL free ifxcursor;

ifx_lo_close(lofd);
}
```

For more information about the ESQL/C functions to create, alter, and access BLOB and CLOB data, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.esqlc.doc/sii-03-sourceforchaptitle.htm

Example using collection data types

Collection data types (SET, LIST, and MULTISET) enable you to store and manipulate collections of data within a single row of a table. A collection data type has two components:

- A type constructor, which determines whether the collection type is a SET, MULTISET, or LIST
- An element type, which specifies the type of data that the collection can contain

The elements of a collection can be of any data type. The elements of a collection are the values that the collection contains. In a collection that contains the values `{'blue', 'green', 'yellow', and 'red'}`, *blue* represents a single element in the collection. Every element in a collection must be of the same type. For example, a collection whose element type is INTEGER can contain only integer values.

Informix ESQL/C uses collection variables to access collection data types. A collection variable stores the elements from a collection column as though they were rows in a table. You can use this *virtual table* as part of a cursor declaration to fetch the individual elements of the collection column.

Example 4-10 shows the ESQL/C commands that are used to retrieve an element from a collection column. The `parents` column is a collection of INT values. The collection is stored into the host variable `hv1` and is used to open a cursor using the variable as a virtual table, `table(:hv1)`.

Example 4-10 The collection_sample

```
EXEC SQL client collection hv1;
EXEC SQL int parent_id;
EXEC SQL select parents into :hv1 from grade12_parents where class_id = 1;
EXEC SQL declare cur1 cursor for select id from table(:hv1);
EXEC SQL open cur1;
EXEC SQL fetch cur1 into :parent_id;
```

The INT elements of the collection are retrieved from the host variable using the **`EXEC SQL fetch cur1`** command.

Example 4-11 shows a complete ESQL/C program that illustrates how to insert and select into tables that contain the collection data types.

Example 4-11 Example using collection types

```
#include <stdio.h>

static void print_collection(
const char *tag,
EXEC SQL BEGIN DECLARE SECTION;
parameter client collection c
EXEC SQL END DECLARE SECTION;
)
{
        EXEC SQL BEGIN DECLARE SECTION;
        int4    value;
        EXEC SQL END DECLARE SECTION;
        mint            item = 0;

        EXEC SQL WHENEVER ERROR STOP;
        printf("COLLECTION: %s\n", tag);
        EXEC SQL DECLARE c_collection CURSOR FOR
                SELECT * FROM TABLE(:c);
        EXEC SQL OPEN c_collection;
        while (sqlca.sqlcode == 0)
        {
                EXEC SQL FETCH c_collection INTO :value;
                if (sqlca.sqlcode != 0) break;
                printf("\tItem %d, value = %d\n", ++item, value);
        }
        EXEC SQL CLOSE c_collection;
        EXEC SQL FREE c_collection;
}
```

```
mint main(int argc, char **argv)
{
        EXEC SQL BEGIN DECLARE SECTION;
        client collection list     (integer not null) lc1;
        client collection set      (integer not null) sc1;
        client collection multiset (integer not null) mc1;
        char *dbase = "stores_demo";
        mint seq;
        char *stmt1 =
                "INSERT INTO t_collections VALUES(0, "
                "'LIST{-1,0,-2,3,0,0,32767,249}', 'SET{-1,0,-2,3}', "
                "'MULTISET{-1,0,0,-2,3,0}') ";
        EXEC SQL END DECLARE SECTION;
if (argc > 1)
                dbase = argv[1];
        EXEC SQL WHENEVER ERROR STOP;
        printf("Connect to %s\n", dbase);
        EXEC SQL connect to :dbase;

        EXEC SQL CREATE TEMP TABLE t_collections
        (
                seq     serial not null,
                l1 list    (integer  not null),
                s1 set     (integer  not null),
                m1 multiset(integer  not null)
        );


        EXEC SQL EXECUTE IMMEDIATE :stmt1;

        EXEC SQL ALLOCATE COLLECTION :lc1;
        EXEC SQL ALLOCATE COLLECTION :mc1;
 EXEC SQL ALLOCATE COLLECTION :sc1;

        EXEC SQL DECLARE c_collect CURSOR FOR
                SELECT seq, l1, s1, m1 FROM t_collections;
        EXEC SQL OPEN c_collect;

        EXEC SQL FETCH c_collect INTO :seq, :lc1, :sc1, :mc1;
        EXEC SQL CLOSE c_collect;
        EXEC SQL FREE c_collect;

        print_collection("list/integer", lc1);
        print_collection("set/integer", sc1);
        print_collection("multiset/integer", mc1);

        EXEC SQL DEALLOCATE COLLECTION :lc1;
        EXEC SQL DEALLOCATE COLLECTION :mc1;
        EXEC SQL DEALLOCATE COLLECTION :sc1;

        puts("OK");
        return 0;
}
```

Example 4-12 shows the output of Example 4-11 on page 146.

Example 4-12 Collection example output

```
Connect to stores_demo
COLLECTION: list/integer
        Item 1, value = -1
        Item 2, value = 0
        Item 3, value = -2
        Item 4, value = 3
        Item 5, value = 0
        Item 6, value = 0
        Item 7, value = 32767
        Item 8, value = 249
COLLECTION: set/integer
        Item 1, value = -1
        Item 2, value = 0
        Item 3, value = -2
        Item 4, value = 3
COLLECTION: multiset/integer
        Item 1, value = -1
        Item 2, value = 0
        Item 3, value = 0
        Item 4, value = -2
        Item 5, value = 3
        Item 6, value = 0
OK
```

4.4.5 Exception handling

The applications that you write require that the database server processes your SQL statements successfully as you intend. If a query fails and you do not know about the failure, you might display meaningless data to the user.

To handle such errors, an Informix ESQL/C program must check that every SQL statement executes as you intend.

This section discusses the two widely used exception handling methods in Informix applications development:

- Use the SQLSTATE variable and the GET DIAGNOSTICS statement to check for runtime errors and warnings that your Informix ESQL/C program might generate.
- Use the SQLCODE variable and the SQL Communications Area (sqlca) to check for runtime errors and warnings that your Informix ESQL/C program might generate.

For more information about ESQL/C Exception handling, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.esqlc.doc/sii-11-40709.htm

Exception handling with the SQLSTATE variable

After the database server executes an SQL statement, it sets the SQLSTATE variable with a value that indicates the success or failure of the statement. From this value, your program can determine if it needs to perform further diagnostics. If the SQLSTATE variable indicates a problem, you can use the GET DIAGNOSTICS statement to obtain more information.

Example 4-13 shows how to handle exceptions using the SQLSTATE variable and the GET DIAGNOSTICS statement. The GET DIAGNOSTICS statement returns information that is held in the fields of the diagnostics area. The diagnostics area is an internal structure that the database server updates after it executes an SQL statement. Each application has one diagnostics area.

Example 4-13 Error handling in ESQL/C

```
#include <stdio.h>
#include <stdio.h>

void error_chk()
{
EXEC SQL BEGIN DECLARE SECTION;
    mint excp_count;
    char overflow[2];
    mint excp_num=1;
    char message[255];
    mint mlen;
    char sql_state_code[6];
    mint i=1;
EXEC SQL END DECLARE SECTION;
 printf("SQLSTATE: %s\n",SQLSTATE);
    printf("SQLCODE: %d\n", SQLCODE);
    printf("\n");

    EXEC SQL get diagnostics :excp_count = NUMBER, :overflow = MORE;

EXEC SQL get diagnostics  exception :i :sql_state_code = RETURNED_SQLSTATE, :message =
MESSAGE_TEXT, :mlen = MESSAGE_LENGTH;
        printf("EXCEPTION %d: SQLSTATE=%s\n", i, sql_state_code);
        message[mlen-1] = '\0';
        printf("MESSAGE TEXT: %s\n", message);
}

mint main(int argc, char **argv)
{
        EXEC SQL BEGIN DECLARE SECTION;
        char *dbase = "stores_demo";
        EXEC SQL END DECLARE SECTION;

        if (argc > 1)
                dbase = argv[1];
        EXEC SQL WHENEVER sqlerror CALL error_chk ;

        printf("Connect to %s\n", dbase);
        EXEC SQL connect to :dbase;

        EXEC SQL SELECT province FROM customer WHERE customer_num=1;
        return 0;
}

OUTPUT:
Connect to stores_demo
SQLSTATE: IX000
SQLCODE: -217

EXCEPTION 1: SQLSTATE=IX000
MESSAGE TEXT: Column (province) not found in any table in the query (or SLV is
undefined).
```

4.4.6 Troubleshooting

In this section, we list frequent ESQL/C errors and discuss how to diagnose a problem using the trace facility. The following typical errors can occur in ESQL/C applications:

- Locale mismatch

  The most common error you might encounter when writing ESQL/C application is a locale mismatch error:

  ```
  -23197  Database locale information mismatch
  ```

  The straight forward solution is to match all the locale-related environment variables, namely CLIENT_LOCALE, DB_LOCALE, and SERVER_LOCALE. The default values for CLIENT_LOCALE are `en_us.8859-1` for UNIX and `en_us.1252` for Windows systems. You can set these variables using the **`export`** (ksh) or **`setenv`** (csh) commands, depending on which shell you are using. On Windows systems, you can set the locale using the `setnet32.exe` utility.

  For more information about these locales and related material, refer to:

  http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.glsug.doc/ids_gug_035.htm

- Compile and linking errors

  If there is a problem with the compile or linking, the following error message is returned:

  ```
  error while loading shared libraries: libifsql.so: cannot open shared
  object file: No such file or directory
  ```

  This error is due generally to a wrong setting of the shared library path. Refer to 4.4.1, “Creating an ESQL/C application” on page 128 for information about the correct settings.

- Database connection errors

  if there is an issue with the database connection, the following error message is returned:

  ```
  -25596 The INFORMIXSERVER value is not listed in the sqlhosts file or the
  Registry.
  ```

  This error is due to incorrect setting of INFORMIXSERVER variable. Verify that it is set to the server instance to which you want to connect.

The SQLIDEBUG trace

ESQL/C uses the SQLI protocol to communicate with the Informix database server. The SQLIDEBUG trace allows you to trace all the messages between the client and server. You can use it to diagnose problems such as SQL errors, unexpected return values, or performance issues.

You can enable the SQLIDEBUG trace on both the client and the server side.

On the server side, you can enable SQLIDEBUG using the **`onmode -p 1 sqli_dbg`** parameter.

To set SQLIDEBUG on the client side, create the environment variable SQLIDEBUG with the following format:

```
sqlidebug=2:path_trace_file
```

where *`path_trace_file`* corresponds to the location and name of the trace file.

Example 4-14 shows how to create the variable on a UNIX platform and how to decode the SQLI trace file using the `sqliprint` utility, which is included as part of Client SDK.

Example 4-14 The SQLIDEBUG trace

```
informix@irk:/work$ export SQLIDEBUG=2:/tmp/sqlitrace
informix@irk:/work$
informix@irk:/work$ sqliprint /tmp/sqlitrace_17008_0_8c819d0 | more
...
```

```
C->S (16)                               Time: 2010-07-06 20:21:58.05185
        SQ_DBOPEN
                 "stores7" [7]
                NOT EXCLUSIVE
        SQ_EOT

S->C (28)                               Time: 2010-07-06 20:21:58.05487
        SQ_DONE
                Warning..: 0x15
                # rows...: 0
                rowid....: 0
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_EOT

C->S (56)                               Time: 2010-07-06 20:21:58.05595
        SQ_PREPARE
                # values: 1
                CMD.....: "SELECT code, sname FROM state WHERE code = ?" [44]
        SQ_NDESCRIBE
        SQ_WANTDONE
...
```

5 Working with the JDBC drivers

In this chapter, we describe how to develop a Java application using the two available Java Database Connectivity (JDBC) drivers. This chapter includes the following topics:

- ► JDBC drivers for an Informix database
- ► Setup and configuration
- ► JDBC type mapping
- ► Performing database operations
- ► Informix additional features

5.1 JDBC drivers for an Informix database

Java Database Connectivity (JDBC) is the JavaSoft specification of a standard application programming interface (API) that allows Java programs to access database management systems.

The JDBC API consists of a set of interfaces and classes written in the Java programming language. Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results.

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

For more information about the JDBC API, refer to JDBC documentation at

http://java.sun.com/javase/6/docs/technotes/guides/jdbc/

You can use the following JDBC drivers from a Java application to process data in an IBM Informix database:

- IBM Informix JDBC Driver
- IBM Data Server Driver for JDBC and SQLJ

Both JDBC drivers are developed as pure-Java drivers (Type 4). Thus, when you use a Type 4 driver from a Java application, your session connects directly to the database or database server without a middle layer.

5.1.1 IBM Informix JDBC Driver

IBM Informix JDBC Driver is a platform-independent, industry-standard Type 4 driver that provides enhanced support for distributed transactions.

Informix JDBC Driver follows the JDBC 3.0 specifications, providing support for the following IBM Informix database engines:

- IBM Informix 7.x, 9.4x, 10.0, 11.10 and 11.50
- IBM Informix Extended Parallel Server (XPS) 8.5x
- IBM Informix Standard Engine (SE) 7.x
- IBM Informix OnLine Version 5.x

To use IBM Informix JDBC Driver Version 3.50.JC6, you must use a JDK 1.4.2 or later package on your platform.

You can download the Informix JDBC Driver from:

http://www14.software.ibm.com/webapp/download/search.jsp?go=y&rs=ifxjdbc

Example 5-1 shows the contents of the Informix JDBC Driver package.

Example 5-1 Informix JDBC directory

```
demo
doc
    javadoc
    release
lib
    ifxjdbc.jar
    ifxjdbcx.jar
    ifxlang.jar
    ifxlsupp.jar
    ifxsqlj.jar
    ifxtools.jar
license
proxy
```

This is a brief explanation of the `.jar` files in the driver:

- `ifxjdbc.jar`

  The optimized implementations of the JDBC API interfaces, classes, and methods.

- `ifxjdbcx.jar`

  The implementation of data source, connection pooling, and XA-related class files.

- `ifxlang.jar`

  The localized versions of all message text supported by the driver.

- `ifxlsupp.jar`

  Support functions for the `ifxlang.jar` package.

- `ifxsqlj.jar`

  The classes for runtime support of SQLJ programs.

- `ifxtools.jar`

  The ClassGenerator, lightweight directory access protocol (LDAP) loader, and other utilities.

5.1.2 IBM Data Server Driver for JDBC and SQLJ

IBM Data Server Driver for JDBC and SQLJ, formerly known as *IBM Driver for JDBC and SQLJ*, is a JDBC driver that uses the DRDA protocol to communicate with IBM database servers. The use of a common communication protocol such as DRDA means that the IBM Data Server Driver for JDBC and SQLJ allows you to write client applications that can use both DB2 and Informix database servers.

The IBM Data Server Driver for JDBC and SQLJ is compliant with the JDBC 3.0 and JDBC 4.0 specifications. This driver is included as part of the IBM Data Server Driver Package, which is bundled with Informix Client Software Development Kit (Client SDK). IBM Data Server Driver for JDBC and SQLJ is also available as a separate download at:

http://www-01.ibm.com/support/docview.wss?rs=71&uid=swg24026929

Example 5-2 shows a listing of the files that are included in the IBM Data Server Driver for JDBC and SQLJ installation package and highlights important files.

Example 5-2 Data Server Driver for JDBC directory (snippet)

```
db2jcc.jar
db2jcc4.jar
sqlj.zip
sqlj4.zip
jdbc4_LI_en
jdbc4_LI_en.rtf
jdbc_LI_en
```

The Data Server Driver for JDBC and SQLJ includes the following JDBC drivers:

- `db2jcc.jar`

  Use the `db2jcc.jar` file in the CLASSPATH if you plan to use the version of the IBM Data Server Driver for JDBC and SQLJ that includes only JDBC 3.0 and earlier functions.

- `db2jcc4.jar`

  Use the `db2jcc4.jar` file in the CLASSPATH if you plan to use the version of the IBM Data Server Driver for JDBC and SQLJ that includes only JDBC 4.0 and earlier functions.

- `sqlj.zip`

  Provides support for SQLJ Java applications. SQLJ is used to embed SQL statements inside Java applications.

- `sqlj4.zip`

  Type 4 driver for SQLJ applications.

5.2 Setup and configuration

In this section, we discuss the setup and configuration parameters of Informix JDBC drivers. We also discuss how to verify the connection with the Informix database server.

5.2.1 Configuration

The Data Server Client Driver for JDBC requires the use of a DRDA connection. Thus, the Informix database server must have an alias configured for DRDA communications. Informix JDBC Driver uses *native* Informix protocol or SQLI and requires no specific alias on the Informix database server when using the Informix JDBC Driver.

On the database server, verify that the `sqlhosts` file contains the correct listeners for the Informix database server.

Example 5-3 shows the `sqlhosts` file of the Informix server that we used in our examples.

Example 5-3 Our sqlhosts file

```
demo_on        onipcshm       kefka   demo_on
demo_on_tcp    onsoctcp       kefka   9088
demo_on_drda   drsoctcp       kefka   9089
```

For information regarding how to set up and configure a DRDA alias on the Informix server, refer to 2.1.3, “Configuring Informix Server” on page 27.

On the client system, you must have one of the two JDBC drivers installed. For information about the installation and configuration, refer to 2.2.3, “Setting up IBM Data Server drivers” on page 43 and 2.2.4, “Setting up Informix JDBC” on page 53.

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

Example 5-4 shows the CLASSPATH environment variable, which contains the default location of both JDBC drivers.

Example 5-4 CLASSPATH example

```
set CLASSPATH=C:\Program Files\IBM\IBM DATA SERVER
DRIVER\java\db2jcc.jar;C:\Program
Files\IBM\Informix_JDBC_Driver\lib\ifxjdbc.jar;C:\Program
Files\IBM\Informix_JDBC_Driver\lib\ifxjdbcx.jar;%CLASSPATH%;.;
```

Products such as IBM Data Studio or IBM WebSphere® Application Server have specific location and configuration files for the JDBC drivers. Always refer to each individual documentation for setup details.

You can find additional information regarding the configuration of the CLASSPATH environment variable in refer to 2.2.3, “Setting up IBM Data Server drivers” on page 43 and 2.2.4, “Setting up Informix JDBC” on page 53.

5.2.2 Verify connectivity with Informix JDBC Driver

In this section, we demonstrate how to verify a correct setup and configuration of the Informix JDBC Driver.

To load IBM Informix JDBC Driver, the application can use the following `Class.forName()` Java method, passing the name of the Informix JDBC Driver as argument:

```
Class.forName("com.informix.jdbc.IfxDriver");
```

Example 5-5 contains a basic Java program that can be used to verify the connectivity with the Informix JDBC Driver. The connection string uses the following parameters:

- The IBM Informix JDBC Driver identifier, `jdbc:informix-sqli`
- The database server host name, `kefka.lenexa.ibm.com`

  You can also specify the IP address.
- The port number of the SQLI listener of the Informix server, `9088`
- The database name, `stores_demo`
- Informix instance identified by the INFORMIXSERVER variable
- User and password for connecting to the database server

Example 5-5 SimpleConnection.java

```
import java.sql.*;

public class SimpleConnection {
  public static void main(String[] args) {

    String url =
"jdbc:informix-sqli://kefka.lenexa.ibm.com:9088/stores_demo:INFORMIXSERVER=demo
_on;user=informix;password=Ifmx4you";
    Connection conn = null;

    System.out.println("URL = \"" + url + "\"");

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

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

    try {
      System.out.println("Connected ...");
      DatabaseMetaData md = conn.getMetaData();
      System.out.println("Driver name: " + md.getDriverName());
      System.out.println("Driver version: " + md.getDriverVersion());
      System.out.println("Database product name: "
          + md.getDatabaseProductName());
      System.out.println("Database product version: "
          + md.getDatabaseProductVersion());

    } catch (SQLException e) {
      System.out.println("FAILED: failed to connect!");
    }

    try {
      conn.close();
    } catch (SQLException e) {
      System.out.println("FAILED: failed to close the connection!");
    }
System.out.println("Done!");
  }
}
```

Example 5-6 shows the compile line and output of the `SimpleConnect.java` program.

Example 5-6 Running SimpleConnection.java

```
C:\RedBook>javac SimpleConnection.java

C:\RedBook>java SimpleConnection
URL =
"jdbc:informix-sqli://kefka:9088/stores_demo:INFORMIXSERVER=demo_on;user=i
nformix;password=Ifmx4you"
Connected ...
Driver name: IBM Informix JDBC Driver for IBM Informix Dynamic Server
Driver version: 3.50.JC6W1
Database product name: Informix Dynamic Server
Database product version: 11.50.FC7
Done!
```

5.2.3 Verify connectivity with the Data Server Driver

You can use the same Java program (Example 5-5 on page 159) to verify the connection with the Data Server Driver for JDBC with minor changes. The following changes are required:

- Load the Data Server JDBC Driver class using the `Class.forName()` method:

  ```
  Class.forName("com.ibm.db2.jcc.DB2Driver");
  ```

- The name of the Data Server JDBC Driver in the connection string should be `jdbc:ids`.
- The port number to use the Data Server Driver client is different from the Informix JDBC client. A common mistake is to specify the SQLI port in the URL that leads to an error such as the following error:

  ```
  com.ibm.db2.jcc.am.io: [jcc][t4][2030][11211][3.58.82] A communication
  error occurred during operations on the connection's underlying socket,
  socket input stream,or socket output stream.  Error location: Reply.fill().
  Message: Insufficient data. ERRORCODE=-4499, SQLSTATE=08001
          at com.ibm.db2.jcc.am.ed.a(ed.java:319)
          at com.ibm.db2.jcc.t4.a.a(a.java:416)
  ```

Example 5-7 shows the changes to the `SimpleConnection.java` source code. You do not need to specify the INFORMIXSERVER variable when using the IBM Data Server Driver for JDBC.

Example 5-7 SimpleConnection.java with Data Server JDBC Driver

```
...

      Class.forName("com.ibm.db2.jcc.DB2Driver");
    } catch (ClassNotFoundException cnfe) {
      System.out.println("No such class available.");
      return;
    }
    try {
      con = DriverManager
          .getConnection("jdbc:ids://kefka:9089/stores_demo",
              "informix", "Ifmx4you");
      System.out.println("connected");
...
```

Example 5-8 shows how to compile and run the program. Notice that the metadata information that the JDBC driver retrieves is slightly different from the metadata retrieved with IBM Informix JDBC Driver.

Example 5-8 Output of SimpleConnection.java using the Data Server JDBC Driver

```
C:\RedBook>javac SimpleConnection.java

C:\RedBook>java SimpleConnection
connected
Driver name: IBM DB2 JDBC Universal Driver Architecture
Driver version: 3.58.82
Database product name: IDS/UNIX64
Database product version: IFX11500
connected
Done
```

5.3 JDBC type mapping

Mapping is a way of specifying data type equivalents. Because there are variations between the SQL data types that are supported by each database vendor, the JDBC API defines a set of generic SQL data types that to use on the Java application. In addition to these data types, which are defined by the JDBC driver, the Java language itself has its own data types that might differ from the SQL types that the database vendor uses. When writing a Java application, the programmer uses Java data types to manipulate the data. The application developers needs to understand the equivalent data type in JDBC and the equivalent data types on the database server.

Table 5-1 lists a few of the data type mapping that is required when working with an IBM Informix database as examples.

Table 5-1 Data Type Mapping from JDBC to Informix basic data types

Java Type	**JDBC Type**	**Informix Type**
long	BIGINT	INT8, BIGINT, BIGSERIAL
byte[]	BINARY, VARBINARY	BYTE
boolean	BIT	BOOLEAN
java.sql.Date	DATE	DATE
java.math.BigDecimal	DECIMAL	DECIMAL
byte[]	LONGVARBINARY	BYTE or BLOB
java.lang.String	LONGVARCHAR	TEXT or CLOB
java.math.BigDecimal	NUMERIC	MONEY
float, java.lang.Float	REAL	SMALLFLOAT
java.sql.Time	TIME	DATETIME HOUR TO SECOND
java.sql.Timestamp	TIMESTAMP	DATETIME YEAR TO FRACTION(5)

For the complete list of data type mappings between the Informix SQL data types and the Informix JDBC Driver, refer to the *JDBC Driver Programmer's Guide* at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.jdbc_pg.doc/sii-xc-21122.htm#sii-xc-21122

For more information regarding the data types that the IBM Data Server Driver for JDBC and SQLJ use, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.jccids.doc/com.ibm.db2.luw.apdv.java.doc/doc/rjvjdata.htm

5.4 Performing database operations

In this section, we provide examples of how to use IBM Informix JDBC Driver and IBM Data Server Driver for JDBC for typical database operations, such as connection to the Informix database, and to manipulate data.

5.4.1 Connection to the database

A Java application can obtain a connection with an Informix database using one of the following methods:

- Use the DriverManager class. This method involves loading the JDBC driver using the `Class.forName()` method and obtaining a connection by calling the `getConnection()` method:

  ```
  Class.forName("com.ibm.db2.jcc.DB2Driver");
  con = DriverManager.getConnection("jdbc:ids://...");
  ```

- Use a `DataSource` object through the `javax.sql` extensions.

  ```
  DB2ConnectionPoolDataSource ds = new DB2ConnectionPoolDataSource();
  PooledConnection poolconn = ds.getPooledConnection();
  con = poolconn.getConnection();
  ```

The DriverManager class was implemented in the original JDBC 1.0 API. This class provides direct access to all the JDBC driver features; however, it requires the application to load the JDBC driver manually and to use a hard-coded connection string with the connection details.

The `DataSource` interface was introduced in the JDBC 2.0 API as the preferred method to obtain a JDBC connection. The application has no need to load a JDBC driver class or provide hard-coded connection details. The interface needs to know only the name of the `DataSource` object that it wants to use. The details for the database connection are stored within the `DataSource` object definition outside the application code.

Connecting using the DriverManager

When using the DriverManager class to connect to an IBM Informix database, a connection string must be passed to the `getConnection()` method with the details for the database server. These details include information such as the system where the database server is running and the port number needed for the connection.

Example 5-9 shows the syntax of the connection string used with Informix JDBC Driver and IBM Data Server Driver for JDBC.

Example 5-9 DriverManager connection string syntax

```
jdbc:[jdbc-driver]://[{ip-address|host-name}:{port-number|service-name}][/dbnam
e]:INFORMIXSERVER=servername[{;user=user;password=password]
```

Connecting to an Informix database uses the following parameters:

- `jdbc-driver`

  This parameter identifies a particular JDBC driver. Informix JDBC Driver uses `informix-sqli`. IBM Data Server Driver for JDBC uses `ids` or `db2`.

- `ip-address` or `host-name`

  This parameter specifies the system that is running the database server.

- `port-number|service-name`

  This parameter specifies the communication port that the database server uses.

- `dbname`

  This parameter names the database to open after a successful connection with the server.

- INFORMIXSERVER

  This parameter identifies the name of the Informix database server to which to connect. This parameter is not required when using Data Server JDBC Driver.

- User and password

  These parameters provide the authentication credentials for the connection.

In addition to these standard details, which are common to all JDBC drivers, the connection string is also used to specify properties that are relevant only to a specific JDBC driver or database server. Table 5-2 lists some of the properties specific to Informix JDBC Driver.

Table 5-2 Informix JDBC Driver connection string

Parameter	**Description**
DB_LOCALE	Locale of the database in the database server.
CLIENT_LOCALE	Locale used by the Client application
INFORMIXCONTIME	Time-out value for a connection with the server
SQLIDEBUG	Enable the trace of the SQLI protocol

Parameter	Description
STMT_CACHE	Enables SQL statement cache on the server
IFX_ISOLATION_LEVEL	Specify the default Isolation Level for the session

IBM Data Server Driver for JDBC supports most of the Informix JDBC Driver properties. For a complete list of the properties that are supported when connecting to an Informix database server, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.jdbc_pg.doc/sii-02conect-66368.htm

Example 5-10 shows how to compile a basic Java code that connects to the Informix database. The driver name and connection string are passed as parameters for the program.

Example 5-10 A connect.java file

```
C:\work>set CLASSPATH=ifxjdbc.jar:db2jcc.jar:.
C:\work>type connect.java
import java.sql.*;

public class connect{
   public static void main( String [] args ) {

   Connection conn = null;
   Statement is = null;

   if (args.length<2) {
      System.out.println("Need the JDBC driver and Connection_String as parameters");
      System.out.println("   java connect \"com.informix.jdbc.IfxDriver\"
\"jdbc:informix-sqli://kodiak:9088/stores_demo
:INFORMIXSERVER=demo_on;user=informix;password=password;\" ");
      return;
   }

   try {
       Class.forName(args[0]);
       conn = DriverManager.getConnection(args[1]);
      System.out.println("Connected to "+conn.getMetaData().getDatabaseProductName());
      conn.close();
   }
   catch ( Exception e ) {
      System.err.println(e);
   }
 }
}

C:\work>javac connect.java
C:\work>java connect "com.informix.jdbc.IfxDriver"
"jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on;
```

```
"
Connected to Informix Dynamic Server

C:\work>java connect "com.ibm.db2.jcc.DB2Driver"
"jdbc:ids://kodiak:9089/stores_demo:user=informix;password=password";
Connected to IDS/NT64

C:\work>
```

Connecting using the DataSource object extensions

When using a `DataSource` object, the information for the database connection is stored as properties of the `DataSource` object. These properties are usually stored in a Java Naming and Directory Interface (JNDI) service that is managed by the Java runtime or the application server.

The application requires no knowledge about the specific connection details. It makes use of the `DataSource` object to get a connection with the database server.

In addition to the method implemented by the JDBC driver to set standard properties, both Informix JDBC Driver and Data Server Driver for JDBC provide extra methods to set and retrieve properties that are specific to the Informix database. These properties have the same effect on the database session as the Informix environment variables that are used by other Informix APIs, such as ODBC or .NET.

Example 5-11 shows a `DataSource` object that is created using the Informix JDBC Driver and how to set some of the `DataSource` object properties.

Example 5-11 DataSource object sample

```
IfxDataSource ds = new IfxDataSource();

ds.setServerName("demo_on");
ds.setDatabaseName("stores_demo");
ds.setPortNumber(9089);
ds.setIfxIFXHOST("kodiak");
ds.setIfxCLIENT_LOCALE("en_US.CP1252");
ds.getConnection("informix", "password");
```

You can use these Informix specific properties to modify the behavior of the JDBC driver or to control particular features of the database server. Table 5-3 lists a portion of the `DataSource` object properties that are implemented in the Informix JDBC Driver.

Table 5-3 DataSource object properties

Informix variable	Method

`CLIENT_LOCALE`	`public String getIfxCLIENT_LOCALE()`
	`public void setIfxCLIENT_LOCALE()`

`DB_LOCALE`	`public String getIfxDB_LOCALE()`
	`public void setIfxDB_LOCALE()`

`IFX_ISOLATION_LEVEL`	`public String getIfxIFX_ISOLATION_LEVEL()`
	`public void setIfxIFX_ISOLATION_LEVEL (1)`

`IFX_LOCK_MODE_WAIT`	`public int getIfxIFX_LOCK_MODE_WAIT()`
	`public void setIfxIFX_LOCK_MODE_WAIT ()`

`SQLIDEBUG`	`public String getIfxSQLIDEBUG ()`
	`public void setIfxSQLIDEBUG ()`

`STMT_CACHE`	`public String getIfxSTMT_CACHE()`
	`public void setIfxSTMT_CACHE()`

`USEV5SERVER`	`public boolean isIfxUSEV5SERVER()`
	`public void setIfxUSEV5SERVER()`

For a complete list of all the available properties, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.jdbc_pg.doc/sii-xb-13590.htm

Because IBM Data Server Driver for JDBC supports both Informix and DB2 database servers, the `DataSource` object properties that it provides are different from those provided by Informix JDBC Driver. For a complete list, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.jccids.doc/com.ibm.db2.luw.apdv.java.doc/doc/rjvdsprp.htm

5.4.2 Manipulating data

In this section, we provide basic examples of how to perform typical data manipulation tasks using a JDBC driver with an IBM Informix database.

Simple SQL statements

A Java application can use a `Statement` object to run basic SQL statements against the database server. The `Statement` interface provides two methods:

- `executeQuery()` returns a `ResultSet` value

  Use this method to run SQL statements that return data (for example, a SELECT SQL statement).

- `executeUpdate()` returns the number of rows effected by the SQL statement

  Use this method to perform INSERT, DELETE, and UPDATE operations or to run Data Definition Language (DDL) statements.

Example 5-12 demonstrates how to perform a DELETE statement using the `executeUpdate()` method.

Example 5-12 An executeUpdate sample

```
C:\work>type delete.java
import java.sql.*;

public class delete{
   public static void main( String [] args ) {

   Connection conn = null;
   Statement is = null;

   try {
      Class.forName("com.informix.jdbc.IfxDriver");
      conn =
DriverManager.getConnection("jdbc:informix-sqli://kodiak:9088/stores_demo:INFOR
MIXSERVER=demo_on");

      is = conn.createStatement();
      int rc=is.executeUpdate("DELETE FROM state WHERE code='"+args[0]+"'");

      System.out.format("Deleted %d rows",rc);

      conn.close();
   }
   catch ( Exception e ) {
      System.err.println(e);
   }
 }
}

C:\work>
C:\work>java delete AZ
Deleted 1 rows
C:\work>
```

Prepare SQL statement with parameters

You can use the PreparedStatement object to run the prepared SQL statement against the database server. Use the PreparedStatement object when the application uses an SQL statement repeatedly. The application can run the SQL statement multiple times with different values, which saves the time that the application takes to process and optimize an SQL statement.

Example 5-13 demonstrates how to use a PreparedStatement object. The example inserts a row in the state table using the parameters that are supplied through the command line.

Example 5-13 An insert.java sample

```
C:\work>cat insert.java
import java.sql.*;
import java.util.*;
import java.text.*;


public class insert{
   public static void main( String [] args ) {

   Connection conn = null;

   try {
      Class.forName("com.informix.jdbc.IfxDriver");
      conn =
DriverManager.getConnection("jdbc:informix-sqli://kodiak:9088/stores_demo:INFOR
MIXSERVER=demo_on");

      PreparedStatement pstmt=conn.prepareStatement("INSERT INTO state
VALUES(?,?,?)");
      pstmt.setString(1, args[0]);
      pstmt.setString(2, args[1]);
      pstmt.setString(3, args[2]);

      int rc=pstmt.executeUpdate();

      System.out.format("Inserted %d rows",rc);
      conn.close();
   }
   catch ( Exception e ) {
      System.err.println(e);
   }
 }
}

C:\work>java insert AZ Arizona 0
Inserted 1 rows
C:\work>
```

Retrieve data from a database table

To retrieve data from a database table or a function that returns more than one row, a JDBC application must use the `ResultSet` object. When the `executeQuery()` method from the `Statement` or `PreparedStatement` object is call, the data is returned in the form of a `ResultSet` object. This object allows the fetching of rows from the database.

Example 5-14 demonstrates how to run an SQL SELECT statement to retrieve data using the `PreparedStatement` and `ResultSet` objects. The result of the prepare statement is stored in the `ResultSet` object, `dbRes`. After that, the `dbRes.next()` method is used to scroll through the `ResultSet` data.

Example 5-14 A select.java sample

```
C:\work>cat select.java
import java.sql.*;

public class select {
 public static void main( String [] args ) {

 Connection conn = null;
 ResultSet dbRes = null;
 Statement is = null;

 try {
    Class.forName("com.informix.jdbc.IfxDriver");
    conn =
DriverManager.getConnection("jdbc:informix-sqli://kodiak:9088/stores_demo:INFOR
MIXSERVER=demo_on;');

    PreparedStatement pstmt=conn.prepareStatement("SELECT * FROM state where
code<?");
    pstmt.setString(1, args[0]);

    pstmt.executeQuery();
    dbRes = pstmt.getResultSet();

    while (dbRes.next()) {
      System.out.format("%s,",dbRes.getString(1));
      System.out.format("%s,",dbRes.getString(2));
      System.out.format("%f\n",dbRes.getDouble(3));
    }

    dbRes.close();
    conn.close();
  }
  catch ( Exception e ) {
    System.err.println(e);
   }
 }
}

C:\work>java select CA
AL,Alabama        ,0.040000
AR,Arizona        ,0.000000
AZ,Arizona        ,0.055000

C:\work>
```

Transactions

Local transaction are controlled directly with the connection object. Methods such as `Connection.commit()` and `Connection.rollback()` are used to resolved a transaction.

By default, all the connections that are created by the Informix JDBC Driver Connection object are in AutoCommit mode. Thus, every SQL statement sent to the Informix database server is committed automatically. If control over the transaction is required, you can turn off the AutoCommit mode using the `Connection.setAutoCommit()` method.

When using Informix JDBC Driver in an XA environment (XA is a standard specification for distributed transactions), the AutoCommit feature is always disabled. The transaction manager is the only component that has control over the transaction.

Example 5-15 demonstrates how to disable AutoCommit mode using the `setAutoCommit(false)` method.

Example 5-15 A sample localtrans.java file

```
C:\work>cat localtrans.java
import java.sql.*;
import java.util.*;
import java.text.*;


public class localtrans{
   public static void main( String [] args ) {

   Connection conn = null;
   Statement is = null;

   try {
      Class.forName("com.informix.jdbc.IfxDriver");
      conn =
DriverManager.getConnection("jdbc:informix-sqli://kodiak:9088/stores_demo:INFOR
MIXSERVER=demo_on");

      conn.setAutoCommit(false);

      is = conn.createStatement();
      int rc = is.executeUpdate("DELETE FROM state WHERE code='UK'");
      System.out.format("Deleted %d rows\n",rc);

      System.out.format("Aborting transaction\n");
      conn.rollback();

      conn.close();
   }
   catch ( Exception e ) {
      e.printStackTrace();
   }
 }
}


C:\work>java localtrans
Deleted 1 rows
Aborting transaction

C:\work>
```

Run a user-defined routine

The CallableStatement object provides a way to run a user-defined routine (UDR) using a standard method that is common to all the IBM database servers.

The results from the execution of a UDR is returned as a result set or as an OUT parameter. Use the following SQL syntax to call a UDR using the CallableStatement interface:

```
"{? = call function_name (?, ?,...)}";
```

The placeholders identify the IN, OUT, and INOUT parameters for the UDR.

The application can set and retrieve the value for the routine parameters using the registerOutParameter() and getxxx() methods.

Example 5-16 shows a basic UDR that is defined with two OUT parameters.

Example 5-16 A state_tax SQL routine

```
CREATE FUNCTION state_tax(OUT vtax percent,
                              vcode CHAR(2),
                          OUT vsname CHAR(20))
                      RETURNS BOOLEAN;
  SELECT sales_tax,upper(sname)
     INTO vtax, vsname FROM state
     WHERE code=vcode;

  RETURN 't';
END FUNCTION;
```

We use CallableStatement to invoke this UDR from a JDBC application. Example 5-17 demonstrates how to set the IN and OUT for the UDR. The BOOLEAN value is returned directly by the SQL function and retrieved using a ResultSet object. To retrieve the OUT parameters, the example uses the getDouble() and GetString() methods.

Example 5-17 A callable.java sample file

```
C:\work>cat callable.java
import java.sql.*;

public class callable {
 public static void main( String [] args ) {

 Connection conn = null;

 try {
    Class.forName("com.informix.jdbc.IfxDriver");
    conn =
DriverManager.getConnection("jdbc:informix-sqli://kodiak:9088/stores_demo:INFOR
MIXSERVER=demo_on;");

    CallableStatement cstmt = conn.prepareCall ("{? = call state_tax(?, ?,
?)}");

    cstmt.registerOutParameter(1, Types.DOUBLE);
    cstmt.registerOutParameter(3, Types.CHAR);
    cstmt.setString(2, "CA");

    ResultSet dbRes = cstmt.executeQuery();
 // Retrieve OUT parameters from the function
    while (dbRes.next())
        System.out.format("UDR returns = %s\n", dbRes.getBoolean(1));

 // Retrieve OUT parameters from the function
    System.out.format("OUT tax   %s\n",cstmt.getDouble(1));
    System.out.format("OUT sname %s\n",cstmt.getString(3));

    dbRes.close();
    cstmt.close();
    conn.close();
  }
  catch ( Exception e ) {
    System.err.println(e);
   }
 }
}
C:\work>java callable
UDR returns = true
OUT tax   0.0825
OUT sname CALIFORNIA
C:\work>
```

5.5 Informix additional features

This section demonstrates the use of the following additional IBM Informix features:

- Batch inserts or updates and using ResultSet metadata
- BIGSERIAL data type
- Informix smart large objects
- Secure Socket Layer

5.5.1 Batch inserts or updates and using ResultSet metadata

In this section, we demonstrate the following concepts:

- Batch inserts

  When performing multiple inserts (or updates) using a prepared statement, it is more efficient to do all the inserts in *bulk*. The method adds all the inserts in a batch and then runs the batch.

- Using `ResultSet` metadata

  When you run a query from a Java program that generates a result set, there is additional information available that is separate from the result set itself. This additional information about the data is returned by the `ResultSet` metadata. For example, you might use `ResultSet` metadata to obtain table definition information from one database to another database. You can dynamically obtain all the details about each column that returns data.

Example 5-18 shows the batch insert and how to obtain the column name and column type from the `ResultSet` metadata. The schema for the `gentemp` table is as follows:

```
CREATE TABLE gentemp (id INT, name VARCHAR(15) NOT NULL)
```

We use the `con.prepareStatement()` method to prepare the SQL INSERT statement. Then, we execute the statement using the `pstmt.addBatch()` method, which makes the INSERT statement a part of a batch operation. After adding 10 rows to the table, we execute the batch operation using the `pstmt.executeBatch()` method.

The metadata information for the table is retrieved using the `rs.getMetaData()` method.

Example 5-18 Listing for the IfxBatchDemo.java file

```
import java.sql.*;
import java.io.*;

public class IfxBatchDemo {
  public static void main(String args[]) throws SQLException, IOException,Exception {

    String IfxURL =
"jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on";
    Connection con = null;
    int cnt = 0;

    System.out.println("Start");
    Class.forName("com.informix.jdbc.IfxDriver");
    con = DriverManager.getConnection(IfxURL);
    con.setAutoCommit(false);

    Statement stmt = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    ResultSetMetaData rsmd = null;

    pstmt = con.prepareStatement("INSERT INTO gentemp (id,name) VALUES (?,?)");
    // Fill in 10 rows
    for (int i = 0; i < 10; i++) {
      pstmt.setInt(1, i + 1000);
      pstmt.setString(2, "String #" + i);
      pstmt.addBatch();
    }
    int[] rows = pstmt.executeBatch();
    System.out.println("  Inserted data. Rows = " + rows.length);
    con.commit();
    pstmt.close();

    pstmt = con.prepareStatement("SELECT * FROM gentemp");
    rs = pstmt.executeQuery();
    rsmd = rs.getMetaData();

    System.out.println("    " + "Col Names " + rsmd.getColumnName(1) + ","
        + rsmd.getColumnName(2));
    System.out.println("    " + "Col Types " + rsmd.getColumnTypeName(1)
        + "," + rsmd.getColumnTypeName(2));
    int row = 1;
    while (rs.next()) {
      System.out.println("    Row  " + row + " = "
          + rs.getInt(rsmd.getColumnName(1)) + ","
          + rs.getString(rsmd.getColumnName(2)));
      row++;
    }

    pstmt.close();
    con.commit();
    con.close();
  }
}
```

5.5.2 BIGSERIAL data type

IBM Informix version 11.50 implements the ANSI standard SQL data type BIGINT and the BIGSERIAL data types:

- BIGINT is mapped to the JDBC standard BIGINT data type. It can be used from a Java application as any other Informix data type.
- BIGSERIAL (similar to SERIAL and SERIAL8) does not have an obvious mapping to any JDBC data type. Thus, the application must use Informix-specific JDBC methods to retrieve the value of these columns.

The methods to access Informix serial data types are included in the Informix JDBC implementation of the JAVA `java.sql.Statement` interface, `IfxStatement`. You can use the `getSerial()`, `getSerial8()`, and `getBigSerial()` methods to retrieve the last value that was inserted on a serial column.

Example 5-19 shows how to retrieve a BIGSERIAL value using the `IfxStatement.getBigSerial()` method. The example uses the following schema for the table:

```
CREATE TABLE tempbs(id BIGSERIAL, name CHAR(10));
```

Example 5-19 The bigserial.java sample

```
import java.sql.*;
import com.informix.jdbc.*;

public class bigserial {
 public static void main( String [] args ) {

 Connection conn = null;
 ResultSet dbRes = null;
 long insertedserial = 0;
 String
url="jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on;";

 try {
    Class.forName("com.informix.jdbc.IfxDriver");
    conn = DriverManager.getConnection(url);

    IfxStatement stmt= (IfxStatement) conn.createStatement();
    stmt.executeUpdate("INSERT INTO tempbs VALUES (0,'test');");
    insertedserial =stmt.getBigSerial();
    System.out.println("Last serial: \t"+insertedserial);

    stmt.executeQuery("SELECT FIRST 1 * FROM tempbs ORDER BY rowid DESC");
    dbRes = stmt.getResultSet();

    while (dbRes.next()) {
      System.out.format("Last row: \t%d,",dbRes.getLong(1));
      System.out.format("%s\n",dbRes.getString(2));
    }

    dbRes.close();
    conn.close();
  }
  catch ( Exception e ) {
    System.err.println(e);
   }
 }
}
```

Example 5-19 on page 177 inserts a row in the `tempbs` table and stores the last BIGSERIAL value in the `insertedSerial` variable. Example 5-20 shows how to compile the example and its output.

Example 5-20 The output of the bigserial.java file

```
C:\work>javac bigserial.java

C:\work>java bigserial
Last serial:    14
Last row:       14,test

C:\work>
```

5.5.3 Informix smart large objects

Smart large objects are a type of large object that Informix supports. Smart large objects are stored logically in a table column but stored physically in a specific type of dbspace called *smart blob space*.

The data that is stored in the table column is a structure that contains information about the large object, such as special attributes or pointers to the location in the smart blob space that contains the data. Informix has the following types of smart large objects:

- A *BLOB* stores binary data.
- A *CLOB* stores character data.

You can manipulate smart large objects with the Informix JDBC Driver using the following methods:

- Standard JDBC 3.0 API

  Using standard JDBC methods such as `getString()`, `setAsciiStream()`, or `getBinaryStream()` allows an application to handle smart lager objects as standard Java data types.

- Informix smart-large-object extensions

  If an application requires random access to the large data, it must use Informix smart-large-object extensions. These extensions to the JDBC API give the application greater control over the smart-large-object data in terms of object properties, concurrency access, and logging.

  The `IfxSmartBlob` interface implements most of the smart-large-object extensions, methods such as `IfxLoCreate()`, `IfxLoOpen()`, `IfxLoRead()`, and `IfxLoWrite()` can be used to access the smart large object structures and to manipulate the data of the object.

  For a complete description of all the Informix smart-large-objects extensions, refer to the *JDBC Driver Programmer's Guide*, which is available at:

  http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.jdbc_pg.doc/sii-04data-36421.htm

Note: IBM Data Server Driver for JDBC does not support the Informix smart large object extensions. Only the standard JDBC API is available to work with smart large objects.

Using the standard JDBC API

Working with smart large objects using the JDBC API does not require any specific Java code. Example 5-21 demonstrates how to retrieve a CLOB column from the Informix database using the `getString()` method.

Example 5-21 The getclob.java sample

```
C:\work>cat getclob.java
import java.sql.*;
public class getclob {
 public static void main( String [] args ) {
 Connection conn = null;
 ResultSet dbRes = null;
 Statement is = null;

 try {
  Class.forName("com.informix.jdbc.IfxDriver");
  conn = DriverManager.getConnection(
       "jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on;");
  PreparedStatement pstmt=conn.prepareStatement("SELECT * FROM catalog WHERE
                                               catalog_num=?");
  pstmt.setString(1, args[0]);
  pstmt.executeQuery();
  dbRes = pstmt.getResultSet();
  dbRes.next();
  System.out.println("Advert description: "+dbRes.getString("advert_descr"));
  dbRes.close();
  conn.close();
  }
 catch ( Exception e ) {
    System.err.println(e);
  }
 }
}
C:\work>javac getclob.java

C:\work>java getclob 10001
Advert description: Brown leather. Specify first baseman's or infield/outfield
style. Specify right- or left-handed.

C:\work>
```

Using the Informix extensions

The smart-large-object extensions allows a Java application to have direct control over the large object structures and data pointers that are used to describe the large object.

The following interfaces provide the methods that are needed to handle smart large objects:

- `IfxLobDescriptor` stores the internal characteristics for a smart large object. The application must create an `IfxLobDescriptor` object to insert a new large object in the database.
- `IfxLocator` identifies a particular large object. An `IfxLocator` object can be created or retrieved from the database to perform any I/O operations with the large object.
- `IfxSmartBlob` represents a smart large object within the Informix JDBC Driver. This object provides all the methods that are necessary to create, open, read, and write to a large object.
- `IfxloStat` stores statistical information about a smart large object, such as the size, last access time, last modified time, and last status change.
- `IfxBblob` and `IfxCblob` add extended functionality to the standard JDBC 3.0 BLOB and CLOB classes.

Example 5-22 demonstrates how to read data from a smart large object using the `IfxSmartBlob` method. The code performs the following steps to select the CLOB column `advert_descr`:

1. Stores the `advert_descr` column as an `IfxCblob` object using the `getClob()` method.
2. Gets the `IfxLocator` from the `IfxCblob` object through the `cblob.getLocator()` method.
3. Creates an `IfxSmartBlob` object and uses the `IfxLocator` to open the smart large object in `smb.IfxLoOpen()`.
4. Reads the first 200 bytes of the CLOB using the `smb.IfxLoRead()` method.
5. Closes the large object and releases the `IfxLocator`.

Example 5-22 The loext.java file

```
import java.sql.*;
import com.informix.jdbc.*;
public class loext{
 public static void main( String [] args ) {
 Connection conn = null;
 byte[] buffer = new byte[200];

 try {
    Class.forName("com.informix.jdbc.IfxDriver");
    conn = DriverManager.getConnection(
       "jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on;");

    PreparedStatement pstmt=conn.prepareStatement("SELECT * FROM catalog WHERE
                                              catalog_num=?");
    pstmt.setString(1, args[0]);
    pstmt.executeQuery();
    ResultSet dbRes = pstmt.getResultSet();
    dbRes.next();

    IfxCblob cblob = (IfxCblob) dbRes.getClob("advert_descr");
    IfxLocator loPtr = cblob.getLocator();
    IfxSmartBlob smb = new IfxSmartBlob(conn);
    int loFd = smb.IfxLoOpen(loPtr, smb.LO_RDONLY);
    int bytesReaded = smb.IfxLoRead(loFd, buffer, buffer.length);
    smb.IfxLoClose(loFd);
    smb.IfxLoRelease(loPtr);

    System.out.println("Advert description: "+new String(buffer).trim());
    dbRes.close();
    conn.close();
  }
  catch ( Exception e ) {
    System.err.println(e);
   }
 }
}
```

Example 5-23 shows how to compile the `loext.java` sample and the output of this sample.

Example 5-23 The output of the loext.java file

```
C:\work>javac loext.java

C:\work>java loext 10001
Advert description: Brown leather. Specify first baseman's or infield/outfield
style.  Specify right- or left-handed.

C:\work>
```

For more information regarding the use of smart large objects with Informix JDBC Driver, refer to the *JDBC Driver Programer's Guide*, which is available at:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.jdbc_pg.doc/sii-04data-36421.htm#sii-04data-36421`

5.5.4 Secure Socket Layer

In addition to the encrypted communications that are provided by the Communication Support Module (CSM), the Informix database server supports the use of Secure Socket Layer (SSL) communications for the encryption of the packages between client and server.

Informix JDBC Driver supports only CSM encryptions. You must use IBM Data Server Driver for JDBC to connect to an SSL-enabled Informix database server.

Before you can use SSL encryption with an Informix database, you must configure both server and client systems.

Preparing the Informix server for SSL

To enable SSL on the Informix server, create an Informix alias using the `drsocssl` protocol.

Example 5-24 enables SSL communications with an Informix server.

Example 5-24 SSL configuration on the Informix server

```
$gsk7capicmd -keydb -create -db kodiak.kdb -pw password -type cms -stash
$gsk7capicmd -cert -create -db kodiak.kdb  -pw password -label testlabel -dn
"CN=bedfont.uk.ibm.com,O=ibm,C=UK" -size 1024 -default_cert yes
$gsk7capicmd -cert -extract -db kodiak.kdb -format ascii -label testlabel -pw
password -target testlabel.cert

$ pwd
/usr3/11.50/ssl
$ ls -a
.                kodiak.crl  kodiak.rdb  testlabel.cert
..               kodiak.kdb  kodiak.sth
$ onstat -c | grep ssl
DBSERVERALIASES kodiak_shm,kodiak_drda,kodiak_ssl
$ grep kodiak_ssl $INFORMIXSQLHOSTS
kodiak_ssl    drsocssl        kodiak   9191
$
```

For detailed information regarding the configuration of SSL with an Informix server, refer to the *Configuring a Server Instance for Secure Sockets Layer Connections* topic in the information center at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.sec.doc/ids_ssl_002.htm

Preparing client for SSL

To enable SSL encryption, both the client and server systems must use the same certificated file. You can import a certified file into the client system using the `keytool` utility, which is included with the Java SDK framework.

Example 5-25 demonstrates how to import the `testlabel.cert` certificate file that was created in Example 5-24 on page 183 to enable a trusted relationship between client and server. A keystore file is created to store the keys for the SSL encrypted connection.

Example 5-25 SSL Client side configuration

```
C:\work>keytool -importcert -file testlabel.cert -keystore .keystore
Enter keystore password:
Re-enter new password:
Owner: CN=bedfont.uk.ibm.com, O=ibm, C=UK
```

```
Issuer: CN=bedfont.uk.ibm.com, O=ibm, C=UK
Serial number: -14b0e6b89751eab1
Valid from: Thu Jul 15 08:10:31 PDT 2010 until: Sat Jul 16 08:10:31 PDT 2011
Certificate fingerprints:
         MD5:  66:FA:CF:44:9F:F3:38:40:7B:9D:93:D6:D6:1C:DB:C5
         SHA1: CF:38:6F:CA:C9:A1:54:43:FC:64:AD:F6:DF:5F:CA:65:01:58:DE:DE
         Signature algorithm name: SHA1withRSA
         Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\work>dir .key* /b
.keystore

C:\work>
```

Using a Java client

A Java application can connect to the SSL Informix server by performing the following operations:

- Set the `javax.net.ssl.truststore` system property to point to the created keystore (in our example, `C:\work\.keystore`).
- Set the `javax.net.ssl.trustStorePassword` System property to the password that is used for the certificate.
- Get a data source object.
- Set the port number to the SSL port, 9191.
- Set the data source property `setSslConnection` to `true`.

Example 5-26 shows basic Java code that connects to an SSL Informix server.

Example 5-26 Listing conssl.java

```
import java.sql.*;
import javax.sql.*;
import java.io.*;
import java.util.*;
import com.ibm.db2.jcc.*;

public class conssl {
  public static void main(String args[]) throws SQLException, IOException,
      Exception {
      System.setProperty ("javax.net.ssl.trustStore","c:/work/.keystore");
      System.setProperty ("javax.net.ssl.trustStorePassword","password");
      DB2ConnectionPoolDataSource ds = new DB2ConnectionPoolDataSource();

      ds.setUser("informix");
      ds.setPassword("password");
      ds.setServerName("kodiak");
      ds.setDatabaseName("stores_demo");
      ds.setPortNumber(9191);
      ds.setDriverType(4);
      ds.setSslConnection(true); // SSL

      PooledConnection poolconn = ds.getPooledConnection();
      Connection con = poolconn.getConnection();

      DatabaseMetaData md = con.getMetaData();

      System.out.println("Driver name: " + md.getDriverName());
      System.out.println("Connected to "+ md.getDatabaseProductName());
      System.out.println("Database product version: " +
                             md.getDatabaseProductVersion());
      con.close();
  }
}
```

After a successful connection, the code produces metadata information such as the driver name and database version. Example 5-27 shows the output.

Example 5-27 Output of conssl.java

```
C:\work>javac conssl.java

C:\work>java conssl
Driver name: IBM DB2 JDBC Universal Driver Architecture
Connected to IDS/UNIX32
Database product version: IFX11500

C:\work>
```

5.6 Typical errors

Common issues in Java applications for Informix include errors due to an application development environment without proper configuration and SQL syntax errors. In this section, we discuss these typical errors.

5.6.1 Class not found errors

The following code shows a class not found error message:

```
C:\RedBook>java IfxSimpleConnection
Exception in thread "main" java.lang.NoClassDefFoundError:
IfxSimpleConnection
Caused by: java.lang.ClassNotFoundException: IfxSimpleConnection
...
```

This type of error is usually from either failing to load the JDBC driver or failing to load the application class because the environment is not configured properly.

To resolve this problem:

- Make sure that you have the JDBC driver `.jar` files in your CLASSPATH.
- Make sure that you have a "." (dot) to include your current directory in the CLASSPATH.

5.6.2 Connectivity errors

When an application fails to connect to the server, you see an error message that is similar to the following message:

```
Exception in thread "main" java.sql.SQLException:
com.informix.asf.IfxASFException: Attempt to connect to database server
(demo_on) failed.
at com.informix.jdbc.IfxSqliConnect.<init>(IfxSqliConnect.java:1319)
...
```

To resolve this problem:

- Verify that the server is running and that the ports are configured correctly on the server.
- Check the port number specified in the application. Remember, Informix JDBC Driver connects to an SQLI port, and IBM Data Server driver connects to a DRDA port.
- Verify that there is no firewall between the client and server. Use Telnet to connect from the client to the server.

5.6.3 Syntax errors

The exception with SQLCODE -201 is a syntax error in an SQL statement that you are trying to run. Other than examining the SQL manually, it can be helpful to know the location of the SQL about which the server is complaining by obtaining the SQL statement offset as follows:

```
try {
  stmt.execute( SQL );
}
catch(Exception e) {
  System.out.println ("Error Offset :"+((IfmxConnection
conn).getSQLStatementOffset() );
  System.out.println(e.getMessage() );
}
```

5.7 Tracing

Tracing the communication with the Informix database server might be required for further diagnostics. The method that you use to enable tracing depends on the Informix JDBC Driver that you are using.

5.7.1 IBM Informix JDBC Driver

Informix JDBC uses the Informix SQLI protocol for the communication with the database server. A trace file with all the SQLI messages can be generated by enabling the SQLITRACE feature within the JDBC driver.

Depending on the method used for the connection, the SQLIDEBUG can be activated using a connection string keyword or a `DataSource` method:

- Set trace for Informix JDBC Driver using `DriverManager`

```
DriverManager.getConnection("jdbc:informix-sqli://kodiak:9088/sysmaster:INF
ORMIXSERVER=demo_on;user=;password=;SQLIDEBUG=/tmp/jdbctrace");
```

- Set trace for Informix JDBC Driver using a `DataSource` object

```
IfxDataSource ifxds = new IfxDataSource();
ifxds.setIfxSQLIDEBUG("c:/temp/sqli.trc");
```

You can run the trace through the `sqliprint` utility to render it to text form.

5.7.2 IBM Data Server Driver for JDBC

IBM Data Server Driver for JDBC uses DRDA as the network protocol and produces a DRDA trace, which is a straight text output trace that can be examined directly.

You can enable DRDA trace with the JDBC using the following methods:

- Set trace for IBM Data Server Driver using `DriverManager`

  ```
  DriverManager.getConnection("jdbc:ids://kodiak:9089/sysmaster:user=;passwor
  d=;traceFile=/jcc.trc;TraceLevel=TRACE_ALL;");
  ```

- Set trace for IBM Data Server Driver using a `DataSource` object

  ```
  DB2SimpleDataSource ds = new DB2SimpleDataSource();
  ds.setTraceFile("/jcc.trc");
  ds.setTraceLevel(com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);
  ds.setTraceFileAppend(false);
  ```

Both SQLIDEBUG and DRDADEBUG traces can be activated at the server side rather than on the client system using the Informix `onmode` utility that is included with the Informix database server. You can find more information about both traces in “Tracing” on page 117.

6

IBM Informix with Hibernate

This chapter describes how to use the HIbernate Java package with an IBM Informix database.

It includes the following topics:

- Hibernate for Java
- Setup and configuration
- Using Hibernate with an Informix database

6.1 Hibernate for Java

This section describes the concepts of Hibernate and the Java Persistence API (JPA) programming model.

6.1.1 Overview of Hibernate

Hibernate is an open source Java object-relational mapping (ORM) and persistence framework that allows you to map Plain Old Java Objects (POJO) to relational database tables using XML configuration files. Hibernate also provides a data query language, Hibernate Query Language (HQL), and retrieval facilities that help reduce development time spent on manual data handling in JDBC calls or SQL statements.

Persistence data refers to any data that must be stored in the database and that must exist after the application has stopped running. Hibernate provides a way to persist the typical relational data as well as Java objects that the application uses.

A relational database stores the information in a tabular way with tables and columns. A relational database ensures the integrity of the data using SQL objects such as constraints and referential integrity.

An object-orientated programming (OOP) language, such as Java or C++, does not have the same data representation as relational databases. The data is represented by objects with attributes and methods. The relationship between these objects is implemented with concepts such as inheritance or polymorphism that do not exist in a relational database.

ORM technologies such as Hibernate try to solve the limitations of object-orientated languages when using relational databases. ORM allows developers to focus on the business logic of the application, without the need for dealing with the data access layer. A developer needs to load only the customer object and does not need to care about how the information for that particular customer is stored in the database. ORM solutions produce more robust and portable code, which means faster development time.

For more information about Hibernate, refer to:

`http://docs.jboss.org/hibernate/stable/core/reference/en/html/`

6.1.2 Hibernate concepts

The Hibernate framework includes the following basic concepts:

- *Persistence*, in Hibernate terms, refers to the concept of storing the state of a Java object in the database so that it can be restored later.
- *Mapping* is the process to map SQL tables to Java objects. Mapping is accomplished using XML mapping files or using Java annotations (metadata that is added to the application source) in the Java code.
- *Object processing* refers to the use of the mapped objects from the application code. How to save and load the state of an Java object using the Hibernate API or the specific Hibernate Query Language (HQL).

In a simplified words, Hibernate for Java is a set of Java APIs that allows you to store and retrieve the state of Java objects into a database using a JDBC driver.

Figure 6-1 illustrates the typical components of a Hibernate application.

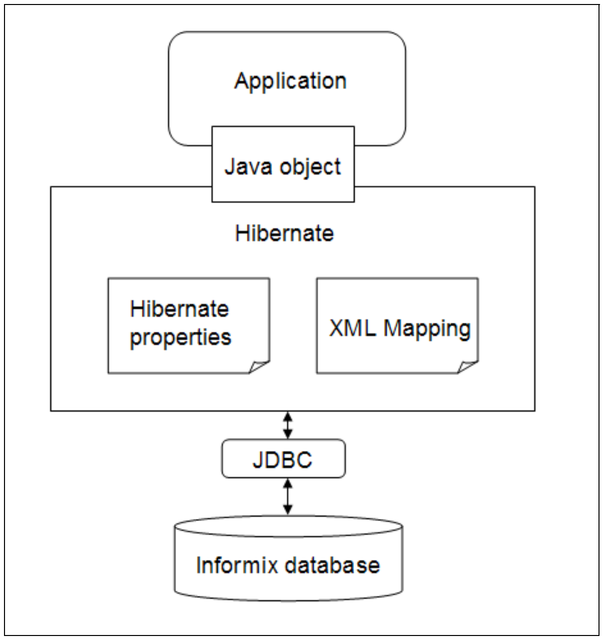


Figure 6-1 Hibernate components

A Hibernate application includes the following components (as shown in Figure 6-1):

- *Application* represents the Java application.
- *Java object* is the object that the application wants to persist (keep in the database).
- *Hibernate properties* is the configuration file for the Hibernate framework, named `hibernate.cfg.xml`, that contains information such as the connection details for the database server or the SQL dialect to use.
- *XML Mapping* is an XML file that contains the mappings between Java objects and database objects.
- *JDBC* is the JDBC driver that is used for the database connection. You can use both IBM Informix JDBC Driver and IBM Data Server Driver for JDBC with Hibernate.
- *Informix database* is the IBM Informix database server.

In addition to the ORM, Hibernate also provides connection management and a transaction management services to be used with a Java application. We do not discuss these topics in this section. For information regarding the use and configuration of the connection and transaction management services, refer to:

`http://docs.jboss.org/hibernate/core/3.3/reference/en/html/transactions.html`

6.2 Setup and configuration

This section describes the installation and configuration of Hibernate to be used with an IBM Informix database.

6.2.1 Installation

Hibernate is an open source project that you can download from:

`http://sourceforge.net/projects/hibernate/files`

The latest version of Hibernate API is V3.5.3, which fully implements the JPA 2.0. The compressed package name is `hibernate-distribution-3.5.3-Final-dist.zip`.

Example 6-1 shows the contents of the Hibernate compressed package.

Example 6-1 Hibernate distribution content

```
changelog.txt
 documentation
        javadocs
        manual
 hibernate-testing.jar
 hibernate3.jar
 hibernate_logo.gif
 lgpl.txt
 lib
     bytecode
     jpa
          hibernate-jpa-2.0-api-1.0.0.Final.jar
     optional
          c3p0
          ehcache
          infinispan
          jbosscache
          oscache
          proxool
          swarmcache
     required
          antlr-2.7.6.jar
          commons-collections-3.1.jar
          dom4j-1.6.1.jar
          javassist-3.9.0.GA.jar
          jta-1.1.jar
          slf4j-api-1.5.8.jar
 project
```

The package contains most of the Java libraries that are required to run a Hibernate application, including the Hibernate documentation and the source code project files for the Hibernate libraries. Table 6-1 describes the Java libraries that are included.

Table 6-1 Included Java libraries

File	Description
`hibernate3.jar`	Hibernate Core library for Relational Persistence
`hibernate-jpa-2.0-api-1.0.0.Final.jar`	Hibernate definition of the Java Persistence 2.0 API
`antlr-2.7.6.jar`	Framework for grammatical descriptions containing Java

File	Description
`commons-collections-3.1.jar`	Types that extend and augment the Java Collections Framework
`dom4j-1.6.1.jar`	XML framework for Java
`javassist-3.9.0.GA.jar`	Java programming Assistant
`jta-1.1.jar`	The javax.transaction package
`slf4j-api-1.5.8.jar`	API for the Simple Logging Facade for Java (SLF4J)

In addition to the libraries that are supplied by the Hibernate package, the `.jar` packages contain the following libraries for the JDBC driver that must be included in the CLASSPATH:

- `ifxjdbc.jar`: IBM Informix JDBC Driver
- `db2jcc.jar`: IBM Data Server Driver for JDBC

Note: The `slf4j-api-1.5.8.jar` package does not contain the complete set of libraries for the Simple Logging Facade for Java (SLF4J). You must use a package that provides an implementation for SLF4J in conjunction with Hibernate API.

Several implementations of SLF4J are available at:

`http://www.slf4j.org/download.html`

In our examples, we use `slf4j-simple-1.6.1.jar` and `slf4j-nop-1.6.1.jar`, which provide simple-logging and discarded-logging.

The Hibernate package is also available as a Maven 2 artifact. Maven is an open source software project management that uses XML files based on the Project Object Model (POM) to manage all the attributes and dependencies files of a Java project. For more information about Maven, refer to the Apache Maven Project documentation at:

`http://maven.apache.org/what-is-maven.html`

6.2.2 Configuration

In this section, we discuss the configuration settings that are required to develop an Informix application with Hibernate.

CLASSPATH

CLASSPATH is an environment variable that tells the Java compiler and the Java virtual machine (JVM) where to look for Java class files and Java libraries.

To use Hibernate with a Java program, you must include the core library, the `hibernate3.jar` file, and all the libraries in the `lib/required` directory in the application CLASSPATH. An implementation of the SLF4J logger and the JDBC driver is also required for Hibernate to work.

Example 6-2 shows a UNIX script that creates the CLASSPATH environment variable. We use `slf4j-nop-1.6.1.jar` with discarded-logging and the Informix JDBC Driver `ifxjdbc.jar` file for our examples.

Example 6-2 Hibernate CLASSPATH script

```
export CLASSPATH=$CLASSPATH:.
export CLASSPATH=$CLASSPATH:/work/lib/commons-collections-3.1.jar
export CLASSPATH=$CLASSPATH:/work/lib/dom4j-1.6.1.jar
export CLASSPATH=$CLASSPATH:/work/lib/hibernate-jpa-2.0-api-1.0.0.Final.jar
export CLASSPATH=$CLASSPATH:/work/lib/hibernate3.jar
export CLASSPATH=$CLASSPATH:/work/lib/javassist-3.9.0.GA.jar
export CLASSPATH=$CLASSPATH:/work/lib/jta-1.1.jar
export CLASSPATH=$CLASSPATH:/work/lib/slf4j-api-1.6.1.jar
export CLASSPATH=$CLASSPATH:/work/lib/antlr-2.7.6.jar

export CLASSPATH=$CLASSPATH:/work/lib/slf4j-nop-1.6.1.jar
export CLASSPATH=$CLASSPATH:/work/lib/ifxjdbc.jar
```

Hibernate configuration file

Hibernate provides the following files to specify configuration parameters:

- `hibernate.properties`: A standard Java properties text file
- `hibernate.cfg.xml`: An XML formatted file

Both files contain the same configuration details for the Hibernate service. If both files exist, the XML configuration file overrides the settings in the properties file.

The configuration file (`hibernate.propertieshibernate.cfg.xml`) contains the information that Hibernate needs to connect to the database server. Details such as name of the JDBC driver class, ConnectionString, and authentication details are kept in this file.

The Hibernate libraries require the configuration file (`hibernate.properties` or `hibernate.cfg.xml`) to be located in the root directory of the CLASSPATH.

Example 6-3 shows a `hibernate.cfg.xml` file using the Informix JDBC Driver.

Example 6-3 A hibernate.cfg.xml sample

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
   <hibernate-configuration>
   <session-factory name="Informix_Session">
<!-- properties -->
   <property name="connection.driver_class">
 com.informix.jdbc.IfxDriver
   </property>
   <property name="connection.url">
 jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on;
   </property>
   <property name="hibernate.connection.username">
 informix
   </property>
   <property name="hibernate.connection.password">
 password
   </property>
   <property name="dialect">
 org.hibernate.dialect.InformixDialect
   </property>

   <property name="hibernate.show_sql">false</property>
   <property name="hbm2ddl.auto">update</property>
<!-- mapping files -->
   <mapping resource="State.hbm.xml"/>

   </session-factory>
</hibernate-configuration>
```

In addition to the JDBC information, the configuration file also contains configuration details for the Hibernate service.

Use the `dialect` property to specify the SQL dialect for the database server. A Hibernate *dialect* is a Java class that contains specific details regarding the SQL syntax that is needed to communicate with a particular database.

When using the IBM Informix JDBC Driver with Hibernate, set the dialect to `orb.hibernate.dialect.InformixDialect`. The dialect for the IBM Data Server Driver for JDBC is `org.hibernate.dialect.DB2Dialect`.

These classes are included in the `hibernate3.jar` package and contain a basic, non-optimized implementation of the Informix dialect.

Note: You can find a patch that adds Informix SQL optimizations to the original Hibernate dialect for Informix databases at:

http://www.iiug.org/opensource/files/hibernate-3.3.2_informix.tar.gz

The XML configuration file allows the inclusion of XML mapping files. Use the `mapping resource` property to specify the XML file that contains the mapping between the SQL table and the Java object.

Example 6-3 on page 196 includes the `State.hbm.xml` mapping file as part of the hibernate configuration, as shown in the following line:

```
<mapping resource="State.hbm.xml"/>
```

Properties, such as `hibernate.show_sql` or `hbm2ddl.auto`, control the behavior of the Hibernate service. Use the `hibernate.show_sql` property to dump all the SQL statements to the console, which might be useful for debugging purposes. Set the `hbm2ddl.auto` property to `update` to specify that the schema for the SQL table is updated automatically if it differs from the definition in the XML definition.

For more information regarding all the supported properties for the Hibernate configuration files, refer to:

http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html

XML mapping file

You can use the XML mapping file to specify the relations between a Java object and an SQL database object. These mapping definitions are used to provide Hibernate with the information that is needed to persist the Java objects into the relational database. They also provide support features, such as creating the database schema and relationship between the objects.

XML mapping files are not required when using Java annotations. The mapping information is added using annotations in the Java source code.

Example 6-4 shows a basic XML file used to map the `States` table to the `State` Java object.

Example 6-4 The State.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
   <class
```

```
      name="State"
      table="States">
      <id
         name="id"
         column="id">
         <generator class="increment"/>
      </id>

      <property
         name="code"
         column="code"/>
      <property
         name="sname"
         column="sname"/>
</class>
</hibernate-mapping>
```

The definition includes information, such as the table name and column to be used as identifier (ID), and specific attributes for the elements, such as the generator class that defines a column as an identifier that is generated automatically by the Java libraries. These details are used by the Hibernate libraries to generate SQL statements automatically, including Data Definition Language (DDL) and Data Manipulation Language (DML).

For a complete list of all the attributes that are available in an XML mapping file, refer to:

http://docs.jboss.org/hibernate/core/3.3/reference/en/html/mapping.html#mapping-declaration

6.3 Using Hibernate with an Informix database

In this section, we demonstrate how to perform basic operations using the Hibernate API with an IBM Informix database server.

6.3.1 Components of a Hibernate application

To develop a Hibernate application, you need to complete the following tasks:

- Create the Java objects.
- Create the XML Mapping files for the Java objects.
- Create the configuration file for the Hibernate framework.

- Create the HibernateUtil helper class that provides access to the Hibernate session.
- Create the application that will use the Java persistence objects.

Java object

Hibernate supports the use of Plain Old Java Objects (POJO) for the definition of a persistence object. Thus, there is a special requirement when writing the Java class that represents the object.

Example 6-5 shows the contents of the State.java file that is used to define the State object. The State Java object is created as any normal Java object. The object has three properties (id, code, and sname) and the usual methods to set and get those properties, such as setcode() or setsname(). The id attribute is used to uniquely identify the Java persistence object.

Example 6-5 The State.java file

```
public class State {
   private Long id;
   private String code;
   private String sname;

   public State() {}
   public State(String text, String text2) {
      this.code = text;
      this.sname = text2;
   }
   public Long getId() {
      return id;
   }
   private void setId(Long id) {
      this.id = id;
   }
   public String getcode() {
      return code;
   }
   public void setcode(String text) {
      this.code = text;
   }
   public String getsname() {
      return sname;
   }
   public void setsname(String text) {
      this.sname = text;
   }
}
```

Example 6-6 shows the CLASSPATH variable used in the environment and how to compile the `State.java` source code.

Example 6-6 Compile line for State.java

```
C:\work>set classpath
CLASSPATH=C:\work\lib\commons-collections-3.1.jar;C:\work\lib\dom4j-1.6.1.jar;C
:\work\lib\hibernate-jpa-2.0-api-1.0.0.Final.jar;C:\work\lib\hibernate3.jar;C:\
work\lib\ifxjdbc.jar;C:\work\lib\javassist-3.9.0.GA.jar;C:\work\lib\jta-1.1.jar
;C:\work\lib\slf4j-api-1.6.1.jar;C:\work\lib\slf4j-nop-1.6.1.jar;C:\work\lib\an
tlr-2.7.6.jar;.

C:\work>javac State.java
C:\work>
```

XML mapping file

An XML mapping file is required to link the Java object with an Informix database object. The `States` table keeps all the instances of the Java `State` object.

The common convention for the name of the XML mapping file is *`objectname_hbm`*`.xml`.

Example 6-7 shows the contents of the `State_hbm.xml` file.

Example 6-7 The State_hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
   <class
      name="State"
      table="States">
      <id
         name="id"
         column="id">
         <generator class="increment"/>
      </id>

      <property
         name="code"
         column="code"/>
      <property
         name="sname"
         column="sname"/>

   </class>
</hibernate-mapping>
```

The name of the class is `State`. It is mapped to the Informix SQL table `States`. The `State` class contains two types of elements:

- `<id>`

  Defines the mapping from that property to the primary key column. It usually contains the generator element that is used to generate unique identifiers for the ID property.

  The generator class supports different methods of generating identifiers. *Increment* is the most basic method, and *sequence* allows to use an SQL sequence to obtain the identifier value. For a list of all the methods implemented in the generator interface, refer to:

  http://docs.jboss.org/hibernate/core/3.3/reference/en/html/mapping.html#mapping-declaration-id

- *`<property name>`*

  Defines a property for the object, including the name of the property and the name of the mapped database table column.

The elements of the XML mapping file can have additional attributes to define specific characteristics of the columns it maps. You can use attributes such as `type` and `not-null` to specify the data type and the nullability of a column.

Hibernate configuration file

The Hibernate configuration file must reside in the root directory of the CLASSPATH. It contains configuration values for the Hibernate service and connection details for the database server.

Example 6-8 shows the contents of the `hibernate.cfg.xml` file that we used in our sample. The connection details correspond to the IBM Informix JDBC Driver.

Example 6-8 The hibernate.cfg.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
   <hibernate-configuration>
   <session-factory name="Informix_Session">
<!-- properties -->
   <property name="connection.driver_class">
 com.informix.jdbc.IfxDriver
   </property>
   <property name="connection.url">
 jdbc:informix-sqli://kodiak:9088/stores_demo:INFORMIXSERVER=demo_on;
   </property>
   <property name="hibernate.connection.username">
 informix
   </property>
   <property name="hibernate.connection.password">
 password
   </property>
   <property name="dialect">
 org.hibernate.dialect.InformixDialect
   </property>

   <property name="hibernate.show_sql">false</property>
   <property name="hbm2ddl.auto">update</property>
<!-- mapping files -->
   <mapping resource="State.hbm.xml"/>

   </session-factory>
```

Example 6-9 shows the attributes that are required for an IBM Data Server Driver for JDBC connection. The configuration file includes the `State.hbm.xml` XML mapping for the `State` object.

Example 6-9 IBM Data Server Driver hibernate.cfg.xml file

```
  <property name="dialect">
org.hibernate.dialect.DB2Dialect
  </property>
  <property name="connection.driver_class">
com.ibm.db2.jcc.DB2Driver
  </property>
  <property name="connection.url">
jdbc:ids://kodiak:9089/stores_demo;
  </property>
```

HibernateUtil helper class

The `HibernateUtil` class is used to interact with the Hibernate service. It performs the operations related to the Hibernate `SessionFactory` classes that provide a convenient way for the application to access the Hibernate session.

The Java file that contains the `HibernateUtil` helper class is `HibernateUtil.java`. Example 6-10 shows a typical helper class.

Example 6-10 The HiberanteUtil.java file

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
  private static final SessionFactory sessionFactory;
  static {
    try {
// Creates the SessionFactory from hibernate.cfg.xml
      sessionFactory = new Configuration().configure().buildSessionFactory();
    }
    catch (Throwable ex) {
      System.err.println("SessionFactory creation failed." + ex);
      throw new ExceptionInInitializerError(ex);
    }
 }

  public static SessionFactory getSessionFactory() {
     return sessionFactory;
  }

  public static void shutdown() {
// Close caches and connection pools
     getSessionFactory().close();
  }
}
```

This class creates a Hibernate session using the parameters that are specified in the Hibernate configuration file. Create the class by compiling the `HibernateUtil.java` Java file as follows:

```
javac HibernateUtil.java
```

Java application

The only task that is required by a Java application to use the Hibernate persistence object is to create a Hibernate Session instance using the `HibernateUtil` helper class. After that, the application can create persist objects in the same manner that it does for any other Java object.

6.3.2 Working with a Hibernate object

Developers must follow an object-orientated methodology when writing applications using the Hibernate API. With Hibernate, the data is represented by the status and properties of an object, not by tables and rows in a relational database.

Storing

When a persistent object is created and saved or stored, an insert operation is performed on the database.

Example 6-11 shows basic Java code that creates a new `State` object using the parameters that are supplied in the command line. The example creates a Hibernate Session object using the helper class and then creates a new `State` object called `myState`. The `myState.setcode()` and `myState.setsname()` methods are used to stored the values that are passed from the command line.

Example 6-11 The Create.java code

```
import java.util.*;
import org.hibernate.*;
import javax.persistence.*;

public class create {
 public static void main(String[] args) {

   Session Session = HibernateUtil.getSessionFactory().openSession();
   Transaction Transaction = Session.beginTransaction();

   State myState = new State();
   myState.setcode(args[0]);
   myState.setsname(args[1]);
   Long stateId = (Long) Session.save(myState);
   System.out.println("Stated added: "+myState.getcode()+", "
                      +myState.getsname());

   Transaction.commit();
   Session.close();
// Shutting down the application
   HibernateUtil.shutdown();
  }
}
```

Example 6-12 shows the compile line and the output of the `create.java` sample.

Example 6-12 Compiling the create.java sample and the output

```
C:\work>javac create.java

C:\work>java create AZ Arizona
Stated added: AZ, Arizona

C:\work>
```

The `hbm2ddl.auto` Hibernate property is set to `update` in the configuration file, which means that if the table does not exist in the database, it is created automatically using the table model that is defined in the XML mapping file.

Example 6-13 shows the schema of the created `States` table and the new row that is added to the table.

Example 6-13 States table schema

```
D:\Infx\ids1150>dbaccess stores_demo -

Database selected.

> INFO COLUMNS FOR states;


Column name         Type                                    Nulls

id                  int8                                    no
code                varchar(255)                            yes
sname               varchar(255)                            yes
> SELECT * FROM states;



id     1
code   AZ
sname  Arizona

1 row(s) retrieved.

>
```

The Java object is made persistent, thus saving the state of the object in the database, using the `Session.save()` Hibernate method and committing the unit of work with `Transaction.commit()`.

Loading

To retrieve a persistent object from the database server, the application must create a Hibernate session and load the state of the object into the current session.

The process of loading a persistence object can be achieved using several methods:

- Use `Session.load()` and `Session.get()` to retrieve the state of an object from the database using the object identifier as reference. For example, to load the `State` object with ID equal to 1, the following code is needed:

  ```
  State mystateobj = (State) Session.load(State.class,1);
  ```

The only difference between the `load()` and `get()` methods is the returned value. If the object is not found, `get()` returns a `null` and `load()` throws an exception.

- Use an SQL or HQL query. HQL is similar to SQL but is optimized for an object-orientated environment. You can use methods, such as `Session.createQuery()` and `Session.createCiteria()`, to load the state of one or multiple objects from the database server. The following command loads all `State` objects into a list:

  ```
  List<State> states = session.createQuery("from state").list();
  ```

Example 6-14 demonstrates how to load a single object from the database using the `Session.load()` method.

Example 6-14 The load.java sample

```
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.criterion.*;

public class load {
 public static void main(String[] args) {

   Long stateId = null;

   Session Session = HibernateUtil.getSessionFactory().openSession();
   Transaction Transaction = Session.beginTransaction();

   stateId=Long.parseLong(args[0]);
   State mystate = (State) Session.load(State.class,stateId);

   System.out.println(mystate.getcode() +", " +
                      mystate.getsname() );

   Transaction.commit();
   Session.close();

  }
}
```

Example 6-15 shows how to compile and run the previous example. The application loads the `State` object identified by the `id` value passed through the command line.

Example 6-15 Output of the load.java sample

```
C:\work>javac load.java

C:\work>java load 1
AZ, Arizona

C:\work>java load 2
CA, California

C:\work>
```

The application can use an HQL query to load all the objects, or *entities,* of a particular type. All the `State` objects are kept in the `States` table. Example 6-16 shows how to retrieve a list of the `State` entities.

Example 6-16 The list.java sample

```
C:\work>cat list.java
import java.util.*;
import org.hibernate.*;
import javax.persistence.*;

public class list {
 public static void main(String[] args) {

   Session newSession = HibernateUtil.getSessionFactory().openSession();
   Transaction newTransaction = newSession.beginTransaction();
   List states = newSession.createQuery("from State order by id asc").list();

   for ( Iterator iter = states.iterator();
      iter.hasNext(); ) {
      State state = (State) iter.next();
      System.out.println(state.getId() +", "+ state.getcode()
                        +", " + state.getsname() );
   }
   newTransaction.commit();
   newSession.close();

   HibernateUtil.shutdown();
  }
}
C:\work>javac list.java

C:\work>java list
```

```
1, AZ, Arizona
2, CA, California

C:\work>
```

Updating

Updating an persistent object is the process of changing the state of the object. No specific operation is required for this task. The application must load the object, change the object properties, and save it.

Example 6-17 demonstrates how to update individual objects. It uses the parameters from the command line to change the `code` and `sname` properties of a `State` object.

Example 6-17 The update.java sample

```
C:\work>cat update.java

import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class update {
 public static void main(String[] args) {

   Long stateId = null;

   Session Session = HibernateUtil.getSessionFactory().openSession();
   Transaction Transaction = Session.beginTransaction();

   stateId=Long.parseLong(args[0]);
   State mystate = (State) Session.load(State.class,stateId);

   mystate.setcode(args[1]);
   mystate.setsname(args[2]);

   Session.save(mystate);
   System.out.println("new values: "+ mystate.getcode()+", "+
                    mystate.getsname() );

   Transaction.commit();
   Session.close();

  }
}
C:\work>javac update.java
```

```
C:\work>java update 1 AZ ARIZONA
new values: AZ, ARIZONA

C:\work>
```

Criteria

Criteria queries are a feature of HQL that allow the building of complex queries using an object-orientated API.

Example 6-18 shows how to update a president object using an HQL criteria to retrieve the object from the database. The HQL criteria is created using the name of the class for the object that you want to select, `State.class`. An HQL Restriction, `Restriction.eq("code", args[0])`, is used to specify a filter for the criteria. This restriction specifies that the query returns the objects with a specific code value.

Example 6-18 The update2.java sample

```
import java.util.*;
import org.hibernate.*;
import javax.persistence.*;
import org.hibernate.criterion.*;

public class update2 {
 public static void main(String[] args) {

   Session Session = HibernateUtil.getSessionFactory().openSession();
   Transaction Transaction = Session.beginTransaction();

   Criteria crit = Session.createCriteria(State.class);
   crit.add( Restrictions.eq( "code", args[0]) );
   List states = crit.list();

   for ( Iterator iter = states.iterator(); iter.hasNext(); ) {
      State mystate = (State) iter.next();
      mystate.setsname(args[1]);
      Session.flush();
      Long msgId = (Long) Session.save(mystate);
      System.out.println("new values: "+ mystate.getcode()+", "+
                         mystate.getsname() );
   }

   Transaction.commit();
   Session.close();

   HibernateUtil.shutdown();
  }
}
```

Example 6-19 shows the output of the example.

Example 6-19 The update2.java output

```
C:\work>javac update2.java

C:\work>java update2 AZ Arizona
new values: AZ, Arizona

C:\work>
```

Deleting

In the Hibernate framework, deleting a persistence object means to make the object *transient*. A transient object is a typical Java object, but the state of the object is not stored anywhere. Thus, the object exists only during the life of the application.

An object can be deleted using the `Session.delete()` method. Example 6-20 demonstrates how to use the `Session.delete()` method.

Example 6-20 The delete.java sample

```
C:\work>cat delete.java
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class delete {
 public static void main(String[] args) {

   Long stateId = null;

   Session Session = HibernateUtil.getSessionFactory().openSession();
   Transaction Transaction = Session.beginTransaction();

   stateId=Long.parseLong(args[0]);
   State mystate = (State) Session.load(State.class,stateId);

   Session.delete(mystate);
   System.out.println( "Object deleted");

   Transaction.commit();
   Session.close();

  }
}
C:\work>javac delete.java
```

```
C:\work>java delete 2
Object deleted

C:\work>java list
1, AZ, Arizona

C:\work>
```

One of the main features of using Hibernate is that it abstracts the JDBC layer from the application. Thus, applications written using the Hibernate API are not tied to a specific JDBC driver or relational database server.

You can run all the examples that we present in this section using any of the two Informix JDBC drivers that are available, IBM Informix JDBC Driver or IBM Data Server Driver for JDBC. Refer to the "Hibernate configuration file" on page 195 for information regarding how to switch between JDBC drivers.

For more information about how to develop using the Hibernate API, refer to:

http://docs.jboss.org/hibernate/core/3.3/reference/en/html/objectstate.html

6.3.3 Using annotations

Java annotations is a feature added into the Java 5 Software Development Kit (SDK) that allows the inclusion of special annotations within the Java source code to express metadata relating to program objects.

Hibernate supports Java annotations through the use of the Hibernate Annotations Extensions. These extensions allow you to include the definition of specific Hibernate properties, such as configuration properties or object mappings properties, directly into the Java code.

With annotations, there is no need for a specific XML mapping file to link the Java objects with the database objects.

Example 6-21 shows the definition of the `State` object using Hibernate annotations. Annotations, such as `@Entity`, `@Table` or `@Column`, define the mapping information that is needed to persist the Java objects into the database.

Example 6-21 The state.java sample with annotations

```
import javax.persistence.*;

@Entity
@Table(name="States")
public class State {
   private Long id;
   private String code;
   private String sname;

   public State() {}
   public State(String text, String text2) {
      this.code = text;
      this.sname = text2;
   }
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY )
@Column(name="id")
   public Long getId() {
      return id;
   }
   private void setId(Long id) {
      this.id = id;
   }
@Column(name="code", length=2, nullable=false)
   public String getcode() {
      return code;
   }
   public void setcode(String text) {
      this.code = text;
   }
@Column(name="sname", length=15, nullable=false)
   public String getsname() {
      return sname;
   }
   public void setsname(String text) {
      this.sname = text;
   }
}
```

Because all the information that is required to map the objects is specified as annotations, you do not need to include a *<mapping>* section in the Hibernate `hibernate.cfg.xml` configuration file.

When using annotations, the `HibernateUtil` helper class, `HibernateUtil.java`, uses the `AnnotationConfiguration()` interface to retrieve the Hibernate properties and the metadata definition from the object class. Example 6-22 shows the `HibernateUtil.java` file that is used to create a `SessionFactory` for the `State.class` object.

Example 6-22 The HibernateUtil.java sample for annotations

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class HibernateUtil {

private static final SessionFactory sessionFactory;
 static {
  try {
// Create the SessionFactory from hibernate.cfg.xml

    sessionFactory = new AnnotationConfiguration()
                         .configure()
                         .addAnnotatedClass(State.class)
                         .buildSessionFactory();

  } catch (Throwable ex) {
    System.err.println("Initial SessionFactory creation failed." + ex);
    throw new ExceptionInInitializerError(ex);
  }
 }

 public static SessionFactory getSessionFactory() {
  return sessionFactory;
 }
 public static void shutdown() {
  getSessionFactory().close();
 }

}
```

You do not need to change the code for the Java applications that use the persistence object. The process when using Hibernate annotations for manipulating the objects is the same as when using the XML mapping files.

You can find a description of all the Hibernate annotation extensions at:

http://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/#entity-hibspec

7

Working with IBM Informix OLE DB Provider

This chapter describes IBM Informix OLE DB Provider. It includes the following topics:

- IBM Informix OLE DB Provider
- Setup and configuration
- Developing an OLE DB application
- Visual Basic, ADO.NET, and SQL Server
- Troubleshooting and tracing

7.1 IBM Informix OLE DB Provider

Informix OLE DB Provider is a Universal Data Access component that enables IBM Informix Server access from OLE DB consumers.

Microsoft OLE DB is a specification for a set of interfaces that is designed to expose data from a variety of sources (relational and non-relational). OLE DB uses the Component Object Model (COM) to accomplish this.

You can use IBM Informix OLE DB Provider to enable client applications, such as ActiveX Data Object (ADO) applications and web pages, to access data on an IBM Informix database server.

Table 7-1 shows the name and COM class ID of the IBM Informix OLE DB Provider.

Table 7-1 COM class ID

Name	DLL	CLSID
ifxoledbc	iifxoledbc.dll	{A6D00422-FD6C-11D0-8043-00A0C90F1C59}

Table 7-2 lists the Informix database servers that support IBM Informix OLE DB Provider.

Table 7-2 Supported databases

Database Server	Versions
IBM Informix	10.0,11.10,11.50
IBM Informix Extended Parallel Server	8.50 and higher
IBM Informix Online	5.20 and higher

7.2 Setup and configuration

In this section, we discuss how to install and set up the OLE DB provider and how to perform basic connectivity tests.

7.2.1 Installation and setup

IBM Informix OLE DB Provider is included only in the Windows version of Informix Client Software Development Kit (Client SDK). The Informix OLE DB

Provider is selected by default in the Client SDK installation. During the installation process, the provider is registered automatically on the Windows registry as a command component.

The default installation directory is the `C:\Program Files\IBM\Informix\Client-SDK` directory. The INFORMIXDIR environment variable should point to the directory where the product is installed. The provider shared library, `ifxoledbc.dll`, is located in `%INFORMIXDIR%\bin` directory.

Because by default Informix OLE DB Provider is registered during Client SDK installation, manual registration is usually not required. However, if needed, you can register Informix OLE DB Provider manually using the Microsoft `regsvr32.exe` tool using the following command:

```
regsvr32.exe %INFORMIXDIR%\bin\ifxoledbc.dll
```

Note: On a Windows x64 (64-bit) system, there are two versions of the `regsvr32.exe` tool:

- The 32-bit version is located in `C:\WINDOWS\SysWOW64`
- The 64-bit version is located in `C:\WINDOWS\System`

Use the correct version when registering Informix OLE DB Provider. Otherwise, the application will fail to load the shared library due to a mismatched version.

After installation, you must run the `coledbp.sql` script on the database server against the `sysmaster` database as user `informix` to add the tables and functions that are required by Informix OLE DB Provider to work.

The `coledbp.sql` script is located in the `%INFORMIXDIR%\etc` directory. If you want to remove the support functions and tables, use the `doledbp.sql` script that is located in the same directory.

7.2.2 Verifying connectivity

Client SDK does not contain any specific tool to test Informix OLE DB Provider.

Internally, Informix OLE DB Provider uses the same Informix connection libraries as ESQL/C or ODBC. You can test the basic connection details for your database server using the `iLogin` utility that is included in the `%INFORMIXDIR%\bin` directory.

Testing using Visual Basic Scripting Edition

Visual Basic Scripting Edition (VBScript) is a scripting language that is included as part of the Windows operating system. It provides access to ADO objects through COM. You can use it to test whether the Informix OLE DB Provider is configured properly.

A VBScript file is a text file with a `.vbs` extension that is loaded automatically and executed by Visual Basic at run time.

Example 7-1 shows a simple VBScript that loads Informix OLE DB Provider and selects a single row from the database.

Example 7-1 The connect.vbs script

```
' ---- Test_Ifx.vbs ----
On Error Resume Next

args = WScript.Arguments.Count
If args < 1 then
  WScript.Echo "usage: connect.vbs Connection_string"
  WScript.Echo " e.g.: connect.vbs ""Data Source=stores_demo@demo_on;User
ID=informix;Password=password;"""
  WScript.Quit
end If

set conn=createobject("ADODB.Connection")
conn.provider = "Ifxoledbc"
conn.connectionstring = WScript.Arguments.Item(0)
conn.open
If Err then
  WScript.Echo "Error!! "+conn.Errors(0).Description
Else
  WScript.Echo "Connected"
  conn.close
End If
' ---- Test_Ifx.vbs ----
```

Example 7-2 shows how run the `connect.vbs` script with a data source string as the parameter to test the database connectivity.

Example 7-2 Output of the connect.vbs script

```
c:\work>connect.vbs
usage: connect.vbs Connection_string
 e.g.: connect.vbs "Data Source=stores_demo@demo_on;User
ID=informix;Password=password;"

c:\work>connect.vbs "Data Source=stores_demo@demo_on;User
ID=informix;Password=password;"
Connected
```

```
c:\work>connect.vbs "Data Source=stores_demo@demo_on;User
ID=informix;Password=invalid;"
Error!! EIX000: (-951) Incorrect password or user informix@dubito is not known
on the database server.
```

Rowset Viewer

You can also test Informix OLE DB Provider using the Rowset Viewer. The Rowset Viewer is included with the Microsoft Data Access Components (MDAC) 2.8 Software Development Kit (SDK), which you can download from the following location:

http://www.microsoft.com/downloads/details.aspx?FamilyID=6c050fe3-c795-4b7d-b037-185d0506396c

To test Informix OLE DB Provider using the Rowset Viewer tool, perform the following steps:

1. Run the Rowset tool.
2. Select **File ∅ Full Connect**.
3. In the Full Connect dialog box, choose **Ifxoledbc** in the Provider field.
4. In the DataSource field, enter the database server name to which you want to connect, for example `stores_demo@demo_on`.
5. Click **OK**.

You also can use Rowset Viewer to run SQL statements against the database server. To execute an SQL statement, write the SQL text in the Rowset pane and click SQL.

7.3 Developing an OLE DB application

This section describes the interfaces that are implemented in Informix OLE DB Provider and demonstrates how to perform basic database operations.

In this section, we discuss the following topics:

- Supported interfaces
- Connecting to database
- Type mapping
- Cursors
- Typical database operations

7.3.1 Supported interfaces

An OLE DB application creates objects based on ADO interfaces to perform operations against a database server. Table 7-3 lists a few ADO interfaces that are implemented in Informix OLE DB Provider as examples. For a complete list, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.oledb.doc/sii-xa-21954.htm#sii-xa-21954

Table 7-3 Informix OLE DB Provider supported interfaces

Interface	Description
IAccessor	Provides methods for accessor management
IColumnsInfo	Provides information about columns of a rowset or prepared command
ICommand	Executes commands
IDBCreateCommand	Obtains a new command
IDBCreateSession	Obtains a new session
IDBDataSourceAdmin	Creates, destroys, and modifies data source objects
IDBProperties	Gets and sets the values of properties on the data source object or enumerator and obtains information about all properties that are supported
IErrorLookup	Used by OLE DB error objects to determine the values of the error message, source, Help file path, and context ID based on the return code and a provider-specific error number
IGetDataSource	Obtains an interface pointer to the data source object
IRowsetIdentity	Indicates row instance identity is implemented on the rowset and enables testing for row identity
ISessionProperties	Returns information about the properties a session supports and the current settings of those properties
ITransaction	Used to commit, abort, and obtain status information about transactions

7.3.2 Connecting to database

In this section, we describe the connection string options with Informix OLE DB Provider.

The connection details for the database server are passed to Informix OLE DB Provider using a connection string. If the `Provider` attribute of the Connection object is not set, you must include the `provider` keyword as part of the connection string to specify the name of Informix OLE DB Provider, *ifxoledbc*. For example:

```
connStr="Provider=Ifxoledbc;Data Source=stores_demo@demo_on"
```

Table 7-4 describes the specific connection string attributes for Informix OLE DB Provider.

Table 7-4 Connection string attributes

Keyword	**Description**
Data Source	Database and Informix Server to which to connect. The syntax for the Data Source parameter is as follows: `database@server` If `@server` is not specified, the default database server is used (corresponding to the value specified by the client's INFORMIXSERVER registry entry or environment variable).
User ID	The user ID used to connect to the Informix server
Password	The password for the user ID
Persist Security Info	Specifies whether the data source can keep authentication information such as the password
Client_locale	The client locale for the application which correspond to the locale used by the Windows OS
Db_locale	The database locale that was used when the database was created
UNICODE	Indicates whether to use IBM Informix GLS Unicode Controls and how the code set conversion to Unicode is done
decasr8	If set, floating point numbers with a scale greater than 30 are returned as `DBTYPE_R8`

Keyword	Description
RSASWS or REPORTSTRINGASWSTRING	Enables you to control the data mapping for wide strings
FBS or FETCHBUFFERSIZE	The size in bytes of the buffer size used to send data to or from the database. Minimum value is 4096, Maximum 32767. Default is 4096.

A typical connection string for Informix OLE DB Provider looks similar to the following string:

```
Data Source=stores_demo@demo_on; User ID=informix; Password=password; Persist
Security Info=True; CLIENT_LOCALE=en_US.CP1252; DB_LOCALE=en_US.819
```

The connection details about the Informix server must be stored in the registry using the `setnet32.exe` utility included in Client SDK.

7.3.3 Type mapping

Informix OLE DB provider supports all the Informix data types, both built-in and extended.

Table 7-5 lists the mappings between the standard OLE DB data types and specific Informix OLE DB types.

Table 7-5 Informix specific type mapping table

Informix SQL	Informix OLE DB Provider
BIGINT	DBTYPE_I8
BIGSERIAL	DBTYPE_I8
BLOB	DBTYPE_BYTES
BOOLEAN	DBTYPE_BOOL
BYTE	DBTYPE_BYTES
CLOB	DBTYPE_STR
DATETIME	DBTYPE_DBDATE, DBTYPE_DBTIME, DBTYPE_DBTIMESTAMP
DECIMAL	DBTYPE_NUMERIC
DISTINCT	Same as underlying type
FLOAT	DBTYPE_R8

Informix SQL	Informix OLE DB Provider
INT8	DBTYPE_I8
INTERVAL	DBTYPE_STR if mapped as a string DBTYPE_Ix (8,4,2 or 1) if mapped as a numeric
LIST	DBTYPE_VARIANT
LVARCHAR	DBTYPE_STR
MONEY (p<=19 s<=4	DBTYPE_CY
MONEY (p>19 s<>4)	DBTYPE_NUMERIC
MULTISET	DBTYPE_VARIANT
Named ROW	DBTYPE_VARIANT
NCHAR	DBTYPE_STR
OPAQUE	DBTYPE_BYTES
SERIAL	DBTYPE_I4
SERIAL8	DBTYPE_I8
SET	DBTYPE_VARIANT
TEXT	DBTYPE_STR
Unnamed ROW	DBTYPE_VARIANT

There are three Informix OLE DB Provider date and time data types that map to one DATETIME Informix SQL data type. Which data type you use depends on the precision of the data type that you desire. See Table 7-6 for a summary.

Table 7-6 DATETIME OLE DB mapping table

Informix OLE DB Provider	Date and time precision
DBTYPE_DBDATE	Day to day, month to day, month to month, year to day, year to month, and year to year
DBTYPE_DBTIME	Hour to hour, hour to minute, hour to second, minute to minute, minute to second, and second to second

Informix OLE DB Provider	Date and time precision
DBTYPE_DBTIMESTAMP	Day to fraction, day to hour, day to minute, and day to second Fraction to fraction, hour to fraction, and minute to fraction Month to fraction, month to hour, month to minute, and month to second Second to fraction Year to fraction, year to hour, year to minute, and year to second

For a complete list of all the data type mapping, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.oledb.doc/using990839.htm#using990839

7.3.4 Cursors

An ADO application uses cursors to move among the data that returned by a recordset. ADO defines the following types of cursors:

- Forward-Only (`adOpenForwardOnly`):

  Provides a copy of the records at the time the Recordset was created. This is the default cursor type of ADO.

- Static (`adOpenStatic`):

  Provides a static copy of the records.

- Dynamic (`adOpenDynamic`):

  Provides a real-time copy of the records, new and altered records by other users are also included.

- Keyset (`adOpenKeyset`):

  Provides a updateable copy of the records at the time the Recordset was created. Only existing records are included.

Informix OLE DB Provider supports all these cursor types with some limitations regarding the use of extended data types and location of the cursor (client or server side). For example:

- Updates of tables with extended data types are not allowed with client-side scrollable cursors.
- Server-side scrollable cursors are not supported with simple large objects (BYTE and TEXT) or collections.
- ROWIDs (internal columns added by the Informix server on non fragmented tables) are required on tables bookmarks and updates.

Refer to the *IBM Informix OLE DB Provider Programmer's Guide* for a complete list of these caveats:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.oledb.doc/sii-using-29878.htm#sii-using-29878`

7.3.5 Typical database operations

In this section, we provide examples of how to use Informix OLE DB Provider to perform typical database operations against an Informix database. We cover operations such as executing SQL statements, selecting data, and handling errors from Informix OLE DB Provider.

For general information about OLE DB architecture and programming, refer to:

`http://msdn.microsoft.com/en-us/library/ms713643%28v=VS.85%29.aspx`

Command

You can execute SQL commands using an OLE DB provider through the `ICommand` interface.

The application has to perform the following steps to execute an SQL statement:

1. Call `QueryInterface` for `IDBCreateCommand` to check if commands are supported in the session.
2. Call `IDBCreateCommand::CreateCommand` to create the command.
3. Call `ICommandText::SetCommandText` to specify the SQL statement for the command.
4. Call `ICommand::Execute` to execute the SQL statement.

Example 7-3 demonstrates how to connect to the database and execute an UPDATE statement. In this example, we use the following operations and interfaces:

1. Connect to the database
   a. Initialize common property options (prompt, DSN, user, and password).
   b. Get an Interface for properties:
      ```
      InterfacepIDBInitialize->QueryInterface(IID_IDBProperties)
      pIDBProperties->SetProperties()
      ```
   c. Get an Interface for creating session object:
      ```
      pIDBInitialize->QueryInterface(IID_IDBCreateSession)
      ```
   d. Create a session object:
      ```
      pCreateSession->CreateSession()
      ```
2. Create a command object:
   a. Get an Interface for creating command object:
      ```
      pSession->QueryInterface(IID_IDBCreateCommand)
      ```
   b. Create the command object and get `ICommandText`:
      ```
      Interface:pCreateCommand->CreateCommand()
      ```
3. Set the `CommandText` for the command object:
   ```
   pCommandText->SetCommandText()
   ```
4. Execute the command:
   ```
   pCommandText->Execute()
   ```
5. Clean up.

Example 7-3 The command.cpp sample

```
#define UNICODE
#define _UNICODE
#include <oledb.h>
#include <oledberr.h>
#include <msdaguid.h>
#include <msdadc.h>
#include <comdef.h>
#include <windows.h>
#include <stdio.h>

// CLSID For IBM-Informix Client Side OLE DB Provider
const GUID CLSID_IFXOLEDBC= {0xa6d00422, 0xfd6c, 0x11d0,
                            {0x80, 0x43, 0x0, 0xa0, 0xc9, 0xf, 0x1c, 0x59}};

#define CHECK(hr) if (((HRESULT)(hr)) < 0) {printf( "Error"); return(hr);}
```

```
int main()
{
    HRESULT     hr = S_OK;
    IDBInitialize       *pIDBInitialize;
    IDBCreateSession    *pCreateSession;
    IDBCreateCommand    *pCreateCommand;
    IUnknown            *pSession;
    ICommandText        *pCommandText;
    IDBProperties       *pIDBProperties;
    DBPROP              InitProperties[4];
    DBPROPSET           rgInitPropSet;

    _bstr_t bstrDsnName = "stores_demo@demo_on";
    _bstr_t bstrUserName = "informix";
    _bstr_t bstrPassWord = "password";
    _bstr_t bstrCommand = (WCHAR *) L"UPDATE state SET sname = 'California'
WHERE code = 'CA'";


    CoInitialize( NULL );

// Instantiate a data source object
    CHECK( hr = CoCreateInstance((REFCLSID) CLSID_IFXOLEDBC, NULL,
        CLSCTX_INPROC_SERVER, IID_IDBInitialize, (void **) &pIDBInitialize))

// Set all Properties, Prompt, DSN, User and Password
   // Initialize common property options.
    for (ULONG i = 0; i < 4; i++ )
    {
        VariantInit(&InitProperties[i].vValue);
        InitProperties[i].dwOptions = DBPROPOPTIONS_REQUIRED;
        InitProperties[i].colid = DB_NULLID;
        InitProperties[1].vValue.vt = VT_BSTR;
    }

    InitProperties[0].dwPropertyID = DBPROP_INIT_PROMPT;
    InitProperties[0].vValue.vt = VT_I2;
    InitProperties[0].vValue.iVal = DBPROMPT_NOPROMPT;
    InitProperties[1].dwPropertyID = DBPROP_INIT_DATASOURCE;
    InitProperties[1].vValue.bstrVal = bstrDsnName;
    InitProperties[2].dwPropertyID = DBPROP_AUTH_USERID;
    InitProperties[2].vValue.bstrVal = bstrUserName;
    InitProperties[3].dwPropertyID = DBPROP_AUTH_PASSWORD;
    InitProperties[3].vValue.bstrVal = bstrPassWord;
    rgInitPropSet.guidPropertySet = DBPROPSET_DBINIT;
    rgInitPropSet.cProperties = 4;
    rgInitPropSet.rgProperties = InitProperties;

    // Get initialization properties.Interface
    CHECK (hr = pIDBInitialize->QueryInterface(IID_IDBProperties,
                                                (void**) &pIDBProperties))
    CHECK(hr = pIDBProperties->SetProperties( 1, &rgInitPropSet))
    CHECK( hr = pIDBProperties->Release())
// Connect to the Database Server
    CHECK(hr = pIDBInitialize->Initialize())
    // Get an Interface for creating Session Object
    CHECK( hr = pIDBInitialize->QueryInterface(IID_IDBCreateSession,
                                            (void **) &pCreateSession))
    // Create a Session Object
    CHECK( hr = pCreateSession->CreateSession(NULL, IID_IUnknown,
                                           (IUnknown **) &pSession))
    CHECK( hr = pCreateSession->Release())

// Create Command Object
    // Get an Interface for creating Command Object
    CHECK( hr = pSession->QueryInterface( IID_IDBCreateCommand,
                                          (void **) &pCreateCommand))
    // Create Command Object and get ICommandText Interface
    CHECK( hr = pCreateCommand->CreateCommand(NULL, IID_ICommandText,
                                              (IUnknown **) &pCommandText ))
    CHECK( hr = pCreateCommand->Release())

// Set the CommandText
    CHECK( hr = pCommandText->SetCommandText(DBGUID_DBSQL, bstrCommand))

// Executing the Command
    CHECK( hr = pCommandText->Execute( NULL, IID_NULL, NULL, NULL, NULL))
    printf( "Row Updated");

// Cleanup
    pCommandText->Release();
    pCommandText = NULL;
    pSession->Release();
    pSession = NULL;
    pIDBInitialize -> Uninitialize();
    pIDBInitialize -> Release();
    pIDBInitialize = NULL;

    CoUninitialize();
    return(0);
}
```

Example 7-4 describes how to compile and run Example 7-3 on page 226. The connection string is constructed inside the program. No command line parameters are passed.

Example 7-4 Output of the command.cpp sample

```
C:\work>cl /EHsc /nologo command.cpp
command.cpp

C:\work>command
Row Updated
C:\work>
```

Rowset

A *rowset* is a set of rows, and each row has a number of columns of data. Rowsets are the main OLE DB objects that are used to expose data from a data source.

You can create a rowset object with the following methods:

- Explicitly create a rowset by calling `IOpenRowset::OpenRowset()`.
- Execute an SQL statement such as SELECT that returns rows with a `ICommand:Execute` method.
- Execute any method that returns a rowset or a schema rowset, for example, `ColumnsRowset::GetColumnsRowset` or `IDBSchemaRowset::GetRowset`.

Example 7-5 demonstrates how to create a rowset object and retrieve metadata information. The operations and interfaces to initialize the session and connect to the database are always the same in a OLE DB application. This example also does the following operations:

1. Connect to the database.
2. Create a Command Object.
3. Create a `OpenRowSet` Object:

   ```
   pCreateCommand->QueryInterface(IID_IOpenRowset)
   pIOpenRowset->OpenRowset()
   ```

4. Obtain access to the `IColumnsInfo` interface from the rowset object:

   ```
   pRowset->QueryInterface(IID_IColumnsInfo)
   ```

5. Retrieve the Column Information:

   ```
   pColumnsInfo->GetColumnInfo()
   ```

6. Clean up.

Example 7-5 The rowset.cpp sample

```
#define UNICODE
#define _UNICODE
#include <oledb.h>
#include <oledberr.h>
#include <msdaguid.h>
#include <msdadc.h>
#include <comdef.h>
#include <windows.h>
#include <stdio.h>

// CLSID For IBM-Informix Client Side OLE DB Provider
const GUID CLSID_IFXOLEDBC= {0xa6d00422, 0xfd6c, 0x11d0,
                            {0x80, 0x43, 0x0, 0xa0, 0xc9, 0xf, 0x1c, 0x59}};
```

```
#define CHECK(hr) if (((HRESULT)(hr)) < 0) {printf( "Error"); return(hr);}

int main()
{
    HRESULT     hr = S_OK;

    IDBInitialize       *pIDBInitialize;
    IUnknown            *pSession;
    ITransactionJoin    *pITransactionJoin;
    ICommandText        *pCommandText;
    IOpenRowset         *pIOpenRowset;
    IColumnsInfo        *pColumnsInfo;

    IDBProperties*      pIDBProperties;
    DBPROP              InitProperties[4];
    DBPROPSET           rgInitPropSet;
    IDBCreateSession    *pCreateSession = NULL;
    IDBCreateCommand    *pCreateCommand = NULL;
    IRowset          *pRowset = NULL;
    DBID TableID;
    DBPROPSET rgPropSets[1];
    DBCOLUMNINFO *pDBColumnInfo;
    WCHAR *pStringsBuffer;
    ULONG lNumCols;
    int i=0;

    _bstr_t            bstrDsnName = "stores_demo@demo_on";
    _bstr_t            bstrUserName = "informix";
    _bstr_t            bstrPassWord = "password";
    _bstr_t            TableName =  L"customer";

    CoInitialize( NULL );

// Instantiate a data source object
    CHECK( hr = CoCreateInstance((REFCLSID) CLSID_IFXOLEDBC, NULL,
        CLSCTX_INPROC_SERVER, IID_IDBInitialize, (void **) &pIDBInitialize))

// Set all Properties, Prompt, DSN, User and Password
   // Initialize common property options.
    for (ULONG i = 0; i < 4; i++ )
    {
        VariantInit(&InitProperties[i].vValue);
        InitProperties[i].dwOptions = DBPROPOPTIONS_REQUIRED;
        InitProperties[i].colid = DB_NULLID;
        InitProperties[1].vValue.vt = VT_BSTR;
    }

    InitProperties[0].dwPropertyID = DBPROP_INIT_PROMPT;
    InitProperties[0].vValue.vt = VT_I2;
    InitProperties[0].vValue.iVal = DBPROMPT_NOPROMPT;
    InitProperties[1].dwPropertyID = DBPROP_INIT_DATASOURCE;
    InitProperties[1].vValue.bstrVal = bstrDsnName;
    InitProperties[2].dwPropertyID = DBPROP_AUTH_USERID;
    InitProperties[2].vValue.bstrVal = bstrUserName;
    InitProperties[3].dwPropertyID = DBPROP_AUTH_PASSWORD;
    InitProperties[3].vValue.bstrVal = bstrPassWord;
    rgInitPropSet.guidPropertySet = DBPROPSET_DBINIT;
    rgInitPropSet.cProperties = 4;
    rgInitPropSet.rgProperties = InitProperties;

    // Get initialization properties.Interface
    CHECK (hr = pIDBInitialize->QueryInterface(IID_IDBProperties,
                                               (void**) &pIDBProperties))
    CHECK(hr = pIDBProperties->SetProperties( 1, &rgInitPropSet))
    CHECK( hr = pIDBProperties->Release())

// Connect to the Database Server
    CHECK(hr = pIDBInitialize->Initialize())
    // Get an Interface for creating Session Object
    CHECK( hr = pIDBInitialize->QueryInterface(IID_IDBCreateSession,
                                               (void **) &pCreateSession))
    // Create a Session Object
    CHECK( hr = pCreateSession->CreateSession(NULL, IID_IUnknown,
                                          (IUnknown **) &pSession))
    CHECK( hr = pCreateSession->Release())

// Create Command Object
    // Get an Interface for creating Command Object
    CHECK( hr = pSession->QueryInterface( IID_IDBCreateCommand,
                                          (void **) &pCreateCommand))
    // Create Command Object and get ICommandText Interface
    CHECK( hr = pCreateCommand->CreateCommand(NULL, IID_ICommandText,
                                        (IUnknown **) &pCommandText ))
    CHECK( hr = pCreateCommand->Release())

// Create a OpenRowSet
   CHECK( hr = pCreateCommand->QueryInterface( IID_IOpenRowset,
                                          (void **)&pIOpenRowset ))
   TableID.eKind = DBKIND_NAME;
   TableID.uName.pwszName = TableName;

   CHECK( hr = pIOpenRowset->OpenRowset( NULL, &TableID, NULL, IID_IRowset,
                                  0, rgPropSets, (IUnknown **)&pRowset
))

   // Obtain access to the IColumnsInfo inteface, from the Rowset object
       CHECK( hr = pRowset->QueryInterface(IID_IColumnsInfo, (void ** )
                                           &pColumnsInfo))

   // Retrieve the Column Information
   CHECK( hr = pColumnsInfo->GetColumnInfo( &lNumCols, &pDBColumnInfo,
                                           &pStringsBuffer))
   wprintf( L"\nTable : %s\nColumns :%d\nName\t\tType\tLength",
                                          TableID.uName.pwszName, lNumCols);
   for ( i=0; i<(int)lNumCols; ++i)
   {
      wprintf( L"\n%s\t%d\t%ld", (pDBColumnInfo+i)->pwszName,
                     (pDBColumnInfo+i)->wType, (pDBColumnInfo+i)->ulColumnSize);
   }

// Free the Column Information Interface
   CHECK( (hr = pColumnsInfo->Release()))

// Cleanup
    pCommandText->Release();
    pCommandText = NULL;
    pSession->Release();
    pSession = NULL;
    pIDBInitialize -> Uninitialize();
    pIDBInitialize -> Release();
    pIDBInitialize = NULL;

    CoUninitialize();
    return(0);
}
```

Example 7-6 shows the `name`, `type`, and `length` columns of the `customer` table.

Example 7-6 Output of the rowset.cpp sample

```
C:\work>cl /EHsc /nologo rowset.cpp
rowset.cpp

C:\work>rowset

Table : customer
Columns :7
Name            Type    Length
customer_num    3       4
customer_type   129     1
customer_name   129     32767
customer_loc    3       4
contact_dates   129     32767
cust_discount   131     19
credit_status   129     1
C:\work>
```

Large objects

Large objects (simple and smart) do not require any special handling when using Informix OLE DB Provider.

Smart large objects (BLOB and TEXT) support the ADO `GetChunk()` and `AppendChunk()` methods.

Example 7-7 demonstrates how to retrieve a CLOB column using the GetChunk() method from the ADO library in C++. This example fetches the first three rows of the catalog table from the sample stores_demo database and displays the data for the catalog_number and advert_descr columns.

Example 7-7 The select.cpp sample

```
#include <stdio.h>
#include <afxdisp.h>

#import "c:\program files\common files\system\ado\msado15.dll" rename

("EOF","adoEOF") no_namespace

#define CREATEiNSTANCE(sp,riid) { HRESULT _hr =sp .CreateInstance( __uuidof( riid

) ); \
                                  if (FAILED(_hr)) _com_issue_error(_hr); }

#define RsITEM(rs,x) rs->Fields->Item[_variant_t(x)]->Value
#define UC (char *)
struct InitOle {
    InitOle()  { ::CoInitialize(NULL); }
    ~InitOle() { ::CoUninitialize();   }
} _init_InitOle_;

void main(){

    _RecordsetPtr    spRS;
    _ConnectionPtr   spCON;
    _variant_t       varBLOB;
    long             lDataLength = 0;
    int              nrows=3;

    try{
        CREATEiNSTANCE(spCON,Connection);
        spCON->Provider="ifxoledbc";
        spCON->ConnectionString = L"Data Source=stores_demo@demo_on;"
                                  L"User ID=informix; Password=password;";

// Connect to the database
         spCON->Open( "", "", "", -1 );
         CREATEiNSTANCE(spRS,Recordset)
         spRS->PutRefActiveConnection( spCON );
       spRS->Open(_bstr_t("catalog"),vtMissing, adOpenForwardOnly,
                             adLockOptimistic,adCmdTable);

        while( (spRS->adoEOF == false) && nrows>0 ){
          nrows--;
          printf("catalog_num = %s\n", UC _bstr_t(RsITEM(spRS,"catalog_num")));
// Get the size of the large object
          lDataLength = spRS->Fields->Item["advert_descr"]->ActualSize;
          if(lDataLength > 0) {
// Call GetChunk to retrieve the Blob data
           VariantInit(&varBLOB);
           varBLOB = spRS->Fields->Item["advert_descr"]->GetChunk(lDataLength);
           printf("advert_descr = %s\n",   UC _bstr_t(varBLOB));
           }
// Move the cursor to the next row
          spRS->MoveNext();
        }

        spRS->Close();
        spCON->Close();
    }
    catch( _com_error &e){
        _bstr_t bstrSource(e.Source());
        _bstr_t bs =  _bstr_t(" Error: ") + _bstr_t(e.Error()) +
                _bstr_t(" Msg: ") + _bstr_t(e.ErrorMessage()) +
                _bstr_t(" Description: ") + _bstr_t(e.Description());
        printf("Error %s\n",  bs);
    }
}
#undef UC
```

Example 7-8 shows the content of the output.

Example 7-8   Output of the select.cpp sample

```
C:\work>cl /EHsc /nologo select.cpp /DWINVER=0x0600
select.cpp

C:\work>select
catalog_num = 10001
advert_descr = Brown leather. Specify first baseman's or infield/outfield
style.  Specify right- or left-handed.
catalog_num = 10002
catalog_num = 10003

C:\work>
```

Errors

Informix OLE DB Provider supports the `ISupportErrorInfo` interface. Applications can retrieve information about an OLE DB error using this interface.

Methods such as `IErrorRecords->GetErrorInfo()`, `IErrorInfo->GetDescription()` and `IErrorInfo->GetSource()` are fully supported by Informix OLE DB Provider.

Example 7-9 illustrates how to use these methods. This example retrieves the error information directly after the `IDBInitialize->Initialize()` function call. Because error handling should be done after every use of an interface function, it is a good practice to create an error handle routine to avoid code redundancy.

Example 7-9 The errorinfo.cpp sample

```
#define UNICODE
#define _UNICODE
#include <oledb.h>
#include <oledberr.h>
#include <msdaguid.h>
#include <msdadc.h>
#include <comdef.h>
#include <windows.h>
#include <stdio.h>

// CLSID For IBM-Informix Client Side OLE DB Provider
const GUID CLSID_IFXOLEDBC= {0xa6d00422, 0xfd6c, 0x11d0,
                            {0x80, 0x43, 0x0, 0xa0, 0xc9, 0xf, 0x1c, 0x59}};

#define CHECK(hr) if (((HRESULT)(hr)) < 0) {printf( "Error"); return(hr);}

HRESULT GetDetailedErrorInfo(
           HRESULThresult,
           IUnknown *pBadObject,
           GUID  IID_BadIntereface);



int main()
{
    HRESULT     hr = S_OK;

    IDBInitialize       *pIDBInitialize;
    IUnknown            *pSession;

    IDBProperties*      pIDBProperties;
    DBPROP              InitProperties[4];
    DBPROPSET           rgInitPropSet;
    IDBCreateSession    *pCreateSession = NULL;

    IErrorInfo          *pErrorInfo = NULL;
    IErrorInfo          *pErrorInfoRec = NULL;
    IErrorRecords       *pErrorRecords = NULL;
    ISupportErrorInfo   *pSupportErrorInfo = NULL;

    ULONG               i,ulNumErrorRecs;
    BSTR                bstrDescriptionOfError = NULL;
    BSTR                bstrSourceOfError = NULL;

    _bstr_t            bstrDsnName = "wrong_db@demo_on"; // WRONG DATABASE
    _bstr_t            bstrUserName = "informix";
    _bstr_t            bstrPassWord = "password";

    CoInitialize( NULL );

// Instantiate a data source object
    CHECK( hr = CoCreateInstance((REFCLSID) CLSID_IFXOLEDBC, NULL,
        CLSCTX_INPROC_SERVER, IID_IDBInitialize, (void **) &pIDBInitialize))

// Set all Properties, Prompt, DSN, User and Password
   // Initialize common property options.
    for (ULONG i = 0; i < 4; i++ )
    {
        VariantInit(&InitProperties[i].vValue);
        InitProperties[i].dwOptions = DBPROPOPTIONS_REQUIRED;
        InitProperties[i].colid = DB_NULLID;
        InitProperties[1].vValue.vt = VT_BSTR;
    }

    InitProperties[0].dwPropertyID = DBPROP_INIT_PROMPT;
    InitProperties[0].vValue.vt = VT_I2;
    InitProperties[0].vValue.iVal = DBPROMPT_NOPROMPT;
    InitProperties[1].dwPropertyID = DBPROP_INIT_DATASOURCE;
    InitProperties[1].vValue.bstrVal = bstrDsnName;
    InitProperties[2].dwPropertyID = DBPROP_AUTH_USERID;
    InitProperties[2].vValue.bstrVal = bstrUserName;
    InitProperties[3].dwPropertyID = DBPROP_AUTH_PASSWORD;
    InitProperties[3].vValue.bstrVal = bstrPassWord;
    rgInitPropSet.guidPropertySet = DBPROPSET_DBINIT;
    rgInitPropSet.cProperties = 4;
    rgInitPropSet.rgProperties = InitProperties;

    // Get initialization properties.Interface
    CHECK (hr = pIDBInitialize->QueryInterface(IID_IDBProperties,
                                             (void**) &pIDBProperties))
    CHECK(hr = pIDBProperties->SetProperties( 1, &rgInitPropSet))
    CHECK( hr = pIDBProperties->Release())

// Connect to the Database Server
    hr = pIDBInitialize->Initialize();
    if (hr < 0) {
      pIDBInitialize->QueryInterface(IID_ISupportErrorInfo,
                                     (LPVOID FAR*)&pSupportErrorInfo);
      pSupportErrorInfo->InterfaceSupportsErrorInfo(__uuidof(pIDBInitialize));
      GetErrorInfo(0,&pErrorInfo);

//Get the IErrorRecord interface and get the count of error recs.
      pErrorInfo->QueryInterface(IID_IErrorRecords,(LPVOID FAR*)&pErrorRecords);
      pErrorRecords->GetRecordCount(&ulNumErrorRecs);

//Get the error record, (we only get the first one)
      pErrorRecords->GetErrorInfo(0, GetUserDefaultLCID(), &pErrorInfoRec);
      pErrorInfoRec->GetDescription(&bstrDescriptionOfError);
```

```
       pErrorInfoRec->GetSource(&bstrSourceOfError);

       printf("ERROR!\nResult of 0x%0x (%ld) returned\n",(long)hr,(long)hr);
       printf("Error Source: %S\n",bstrSourceOfError);
       printf("Error Description: %S\n", bstrDescriptionOfError);

       pErrorInfo->Release();
       pErrorRecords->Release();
       pSupportErrorInfo->Release();
       pErrorInfoRec->Release();
       SysFreeString(bstrSourceOfError);
       SysFreeString(bstrDescriptionOfError);
    } // if

/* ... some useful code here .... */

    pIDBInitialize -> Uninitialize();
    pIDBInitialize -> Release();
    pIDBInitialize = NULL;

    CoUninitialize();
    return(0);
}
```

Example 7-10 shows the output of Example 7-9 on page 235. We use a non-existent database for the connection. An error is expected during initialization.

Example 7-10 Output of the errorinfo.cpp sample

```
C:\work>cl /EHsc /nologo errorinfo.cpp
errorinfo.cpp

C:\work>errorinfo
ERROR!
Result of 0x80004005 (-2147467259) returned
Error Source: Ifxoledbc
Error Description: EIX000: (-329) Database not found or no system permission.

C:\work>
```

7.4 Visual Basic, ADO.NET, and SQL Server

ActiveX and COM objects are a core component of any Windows operating system. For this reason, OLE DB providers are used extensively by many Windows applications. In this section, we describe how to use Informix OLE DB

Provider with some of the available Microsoft technologies and applications, such as ADO.NET or Microsoft SQL Server.

7.4.1 OLE DB with Visual Basic

Visual Basic (VB) is a programming language developed by Microsoft that focuses on the use of COM objects. Visual Basic is design to be easy to learn and to allow the development of database applications with far less effort than C or C++ languages. ADO and Informix OLE DB Provider are widely used with Visual Basic.

In this section, we demonstrate how to use Informix OLE DB Provider to access an Informix database using a VBScript that is based on Visual Basic.

Select data

Example 7-11 shows how to query the state table using a recordset with a VBScript file. We open the recordset using a static server cursor (`adUseServer=2` and `adOpenStatic=3`).

Example 7-11 The select.vbs script

```
' Create the ADO objects
   set cx=createobject("ADODB.Connection")
   set cr=createobject("ADODB.Recordset")

' Set the connection string
   cx.provider="Ifxoledbc"
   cx.connectionstring="Data Source=stores_demo@demo_on;"
' Open the Connection
   cx.open
   set cr.activeconnection=cx
  cr.cursorlocation=2
' Open the Recordset
   cr.open  "SELECT * FROM state WHERE code='CA'", cx, 2, 3
   WScript.Echo cr.fields("sname")
```

Example 7-12 shows the output of the this script.

Example 7-12 Output of the select.vbs script

```
c:\work>select.vbs
California

c:\work>
```

Comparing this example with Example 7-5 on page 229, which is written on C++, shows how easy it is to use OLE DB with Visual Basic.

Example 7-13 demonstrates how to scroll through a recordset.

Example 7-13 The scroll.vbs script

```
' Create the ADO objects
   set cx=createobject("ADODB.Connection")
   set cr=createobject("ADODB.Recordset")
   set cm=createobject("ADODB.Command")
' Set the connection string
   cx.provider = "Ifxoledbc.2"
   cx.Open "Data Source=stores_demo@demo_on"
' Set the Command SQL text
   cm.ActiveConnection = cx
   cm.CommandText = "SELECT * FROM state"
' Open the cursor
   cr.CursorLocation = 3 ‘ adUseClient
   cr.Open cm
   WScript.Echo "First row: " & cr.fields("sname")
' Scroll to the last element in the recordset
   cr.MoveLast
   WScript.Echo "Last row : " & cr.fields("sname")
```

We use the `MoveLast()` method to position the cursor in the last record, as shown in the output in Example 7-14.

Example 7-14 Output of the scroll.vbs script

```
c:\work>scroll.vbs
First row: Alaska
Last row : Puerto Rico

c:\work>
```

Add new data

Example 7-15 demonstrate how to add a new record to a table using the `AddNew()` method. After adding a new record to the table, the code opens a cursor and positions it at the end of the recordset to retrieve the last inserted row.

Example 7-15 The addnew.vbs script

```
' Create the ADO objects
   set cx = CreateObject("ADODB.Connection")
   set cr = CreateObject("ADODB.Recordset")
' Set the connection string
   cx.provider = "Ifxoledbc"
   cx.connectionstring = "Data Source=stores_demo@demo_on;"
   cx.Open
   Set cr.activeconnection = cx
' Open the recordset
   cr.Open "SELECT * FROM state", cx, 3, 2
' Add a new record to the recordset
   cr.AddNew
   cr.fields("code") = "UN"
   cr.fields("sname") = "Unkown"
   cr.fields("sales_tax") = "0.0"
' Update the database table
   cr.Update
   cr.Close
' Retrieve the new inserterd row
   cr.open  "SELECT * FROM state", cx, 3, 3
   cr.MoveLast
   WScript.Echo cr.fields("sname")
```

Example 7-16 shows the value of the `sname` column after the update operation.

Example 7-16 Output of the addnew.vbs script

```
c:\work>addnew.vbs
Unkown

c:\work>
```

7.4.2 ADO.NET and the OLEDB bridge

In addition to ADO, developers can also use Informix OLE DB Provider with ADO.NET. Similar to ADO but within the .NET Framework, ADO.NET is a set of .NET components that are used to retrieve, manipulate, and update data from a data source.

One of the .NET data providers that is included with ADO.NET is the Microsoft OLE DB Provider for .NET. This .NET provider acts as bridge between the .NET framework and standard OLE DB providers so that developers can use .NET technology with a database server, even if there is no specific .NET data provider for that particular database.

IBM has specific .NET providers to access an Informix database:

- IBM Informix .NET Provider
- IBM Data Server Provider for .NET

However, developers can use Informix OLE DB Provider through the Microsoft OLE DB .NET Provider.

Example 7-17 demonstrates how to use Informix OLE DB Provider with a .NET application. The connection string for Informix OLE DB Provider is the same. Refer 7.3.2, “Connecting to database” on page 221 for the complete syntax.

Example 7-17 The getoledb.cs sample

```
using System;
using System.Data;
using System.Data.OleDb;

class sample  {
  static void Main(string[] args) {

   OleDbConnection con = new OleDbConnection();
   con.ConnectionString = "Provider=Ifxoledbc;"
       +" Data Source=stores_demo@demo_on; User ID=informix;"
       +" Password=password";

   DataSet ds = new DataSet();
   string sql = "SELECT first 3 * FROM STATE";
   OleDbDataAdapter da = new OleDbDataAdapter(sql,con);

   da.Fill(ds,"state");

   foreach(DataRow dr in ds.Tables[0].Rows) {
     Console.WriteLine(dr["code"]);
     Console.WriteLine(dr["sname"]);
     Console.WriteLine(dr["sales_tax"]);
   }

   con.Close();
   con.Dispose();
  }
}
```

Example 7-18 demonstrates how to compile and run this .NET example.

Example 7-18 Output of the getoledb.cs sample

```
c:\work>csc.exe /nologo getoledb.cs

c:\work>getoledb
AK
Alaska
0.00000
HI
Hawaii
0.04000
CA
```

```
California
0.08250

c:\work>
```

7.4.3 SQL Server

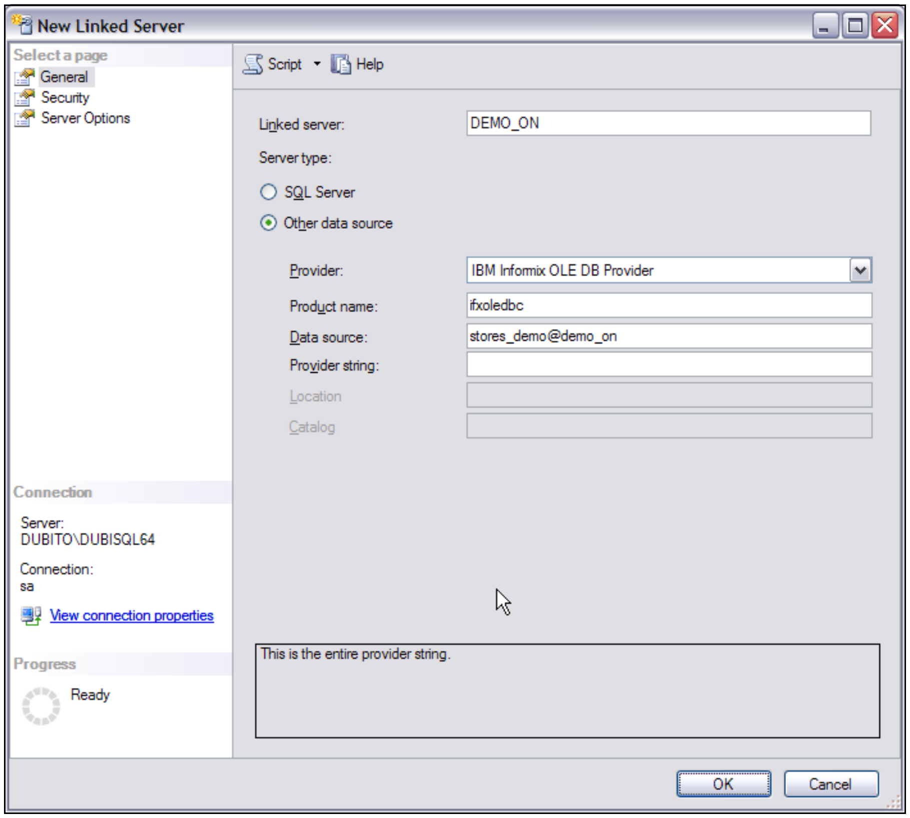
Using Informix OLE DB Provider, you can select data from an Informix database directly from Microsoft SQL Server.

SQL Server uses OLE DB to create linked servers and to retrieve data from any OLE DB data source.

There are no specific steps to configure or to use Informix OLE DB Provider with SQL Server as a linked server. If Informix OLE DB Provider is installed and configured correctly, SQL Server can use it to connect to an IBM Informix database server.

We describe how to set up a linked server using the Informix provider in this section.

Figure 7-1 shows the New Linked Server dialog box where you set the connection details for Informix OLE DB Provider.



Linked server: DEMO_ON

Server type:

SQL Server [✓ Other data source]

Provider: IBM Informix OLE DB Provider

Product name: ifxoledbc

Data source: stores_demo@demo_on

Provider string:

Location

Catalog

Server:
DUBITO\DUBISQL64

Connection:
sa

Ready

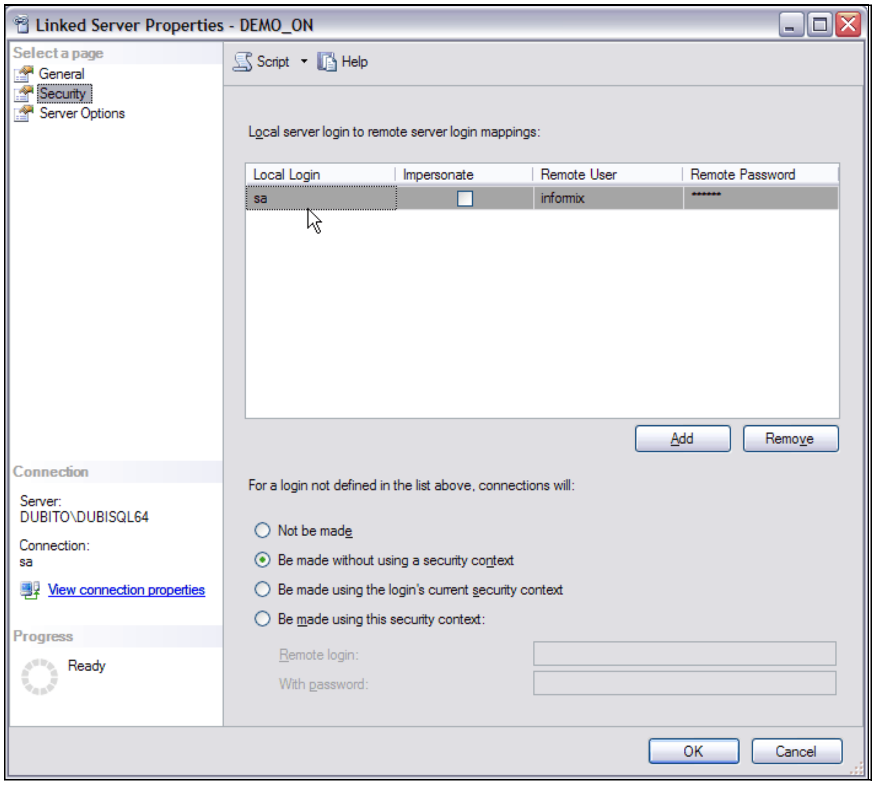
This is the entire provider string.

Figure 7-1 New Linked Server dialog box

In this dialog box, set the following fields:

- Linked server: Specify the name for the SQL Server to link.
- Provider: Choose **IBM Informix OLE DB Provider** from the drop-down list.
- Product name: Specify the name of the Informix provider, which in this example is `ifxoledbc`.
- Data source: Specify the name of the data source as `database@server`.
- Provider string: Specify any additional connection string parameters that the provider uses.

The user ID for the Informix database server might differ from the one that is used with SQL Server. If so, you need to set a remote user mapping. Figure 7-2 shows a user mapping between the `sa` and `informix` users.



Local server login to remote server login mappings:

Local Login	Impersonate	Remote User	Remote Password
sa	☐	informix	******

For a login not defined in the list above, connections will:

- ○ Not be made
- ● Be made without using a security context
- ○ Be made using the login's current security context
- ○ Be made using this security context:

Remote login:

With password:

Connection

Server:
DUBITO\DUBISQL64

Connection:
sa

View connection properties

Progress

Ready

Figure 7-2 Linked Server Properties: Security options

You can also create linked servers using only SQL statements. Example 7-19 illustrates how to create a linked server to connect to the `demo_on` Informix instance. We use the same connection details shown in Figure 7-1 on page 243.

Example 7-19 Linked Server SQL script

```
EXEC master.dbo.sp_addlinkedserver
 @server = N'demo_on',
 @srvproduct=N'ifxoledbc',
 @provider=N'ifxoledbc',
 @datasrc=N'stores7@demo_on',
 @provstr=N''

EXEC sp_addlinkedsrvlogin 'demo_on',false,'sa','informix','password'

SELECT * FROM demo_on.stores7.informix.customer
```

The last SQL statement in the script uses the linked server to retrieve the data from the customer table:

```
SELECT * FROM demo_on.stores7.informix.customer
```

For more information about linked servers, refer to:

http://msdn.microsoft.com/en-us/library/ms188279.aspx

7.5 Troubleshooting and tracing

In this section, we discuss typical errors that can occur when using Informix OLE DB Provider and how to enable the tracing facility to collect diagnostic information.

7.5.1 Typical errors

Most errors encountered when using Informix OLE DB Provider are due to an incorrect setup or an invalid connection string passed to the provider.

Informix OLE DB Provider uses the same connectivity libraries as other components of Client SDK. It also uses the information that is stored in the registry through the `setnet32.exe` utility for the database server connection.

It is always a good practice to test this basic connectivity using the `iLogin` utility that is included in the Client SDK directory.

Provider not found

Here, we explain the reason and the solution for a “`Provider not found`” error.

Reason

This error appears if the Windows operating system fails to locate the OLE DB provider class that is specified by the application.

Solution

During the installation of Client SDK, the OLE DB provider is registered automatically within the Windows registry. You can register Informix OLE DB Provider manually using the `regsvr32` utility as follows:

```
regsvr32.exe %INFORMIXDIR%\bin\ifxoledbc.dll
```

If you are running a 64-bit version of Windows, make sure that you register the provider using the correct version of the `regsvr32` binary. Otherwise, it can happen that the 32-bit provider is registered as a 64-bit provider (or vice versa) and the Windows operating system will fail to find and load the provider.

You can verify that the provider library is registered correctly by examining the Windows registry. Example 7-20 shows the registry entry for 32-bit and 64-bit Informix OLE DB Provider. Remember that the name of the Informix provider is `ifxoledbc`.

Example 7-20 Informix OLE DB registry keys

```
C:\work>c:\windows\syswow64\reg query
"HKEY_CLASSES_ROOT\CLSID\{A6D00422-FD6C-11D0-8043-00A0C90F1C59}\InprocServer32"

HKEY_CLASSES_ROOT\CLSID\{A6D00422-FD6C-11D0-8043-00A0C90F1C59}\InprocServer32
    (Default)    REG_SZ    C:\Program Files
(x86)\IBM\Informix\Client-SDK\bin\ifxoledbc.dll
    ThreadingModel    REG_SZ    Both


C:\work>c:\windows\system32\reg query
"HKEY_CLASSES_ROOT\CLSID\{A6D00422-FD6C-11D0-8043-00A0C90F1C59}\InprocServer32"

HKEY_CLASSES_ROOT\CLSID\{A6D00422-FD6C-11D0-8043-00A0C90F1C59}\InprocServer32
    (Default)    REG_SZ    C:\Program
Files\IBM\Informix\Client-SDK\bin\ifxoledbc.dll
    ThreadingModel    REG_SZ    Both


C:\work>
```

Failed to load the Ifxoledbc.dll

Here, we explain the reason and the solution for a "`Failed to load the Ifxoledbc.dll`" error.

Reason

The application fails to load Informix OLE DB Provider or one of its libraries.

Solution

The value of INFORMIXDIR (as a environment variable or in the registry) should be the directory where Client SDK is installed. Ensure that the environment variable PATH contains the `%INFORMIXDIR%\bin` directory. For example:

```
PATH=C:\Program Files\IBM\Informix\Client-SDK\bin;%PATH%
```

When an application loads Informix OLE DB Provider, Client SDK libraries that the provider needs for communication and registry access are loaded from the directory that is used in the registration process.

On 64-bit version of Windows operating system, ensure that the PATH and INFORMIXDIR variables are set correctly for all users. Some applications, such as SQL Server, are executed under credentials that are different than the logged user and that can have different settings for this variables.

Failed to connect to the database

Here, we explain the reason and the solution for a "`Failed to connect to the database`" error.

Reason

The most common reason for this type of error is due to invalid parameters in the connection string or invalid connectivity information.

Solution

If the application is getting an Informix OLE DB error, it has managed to load Informix OLE DB Provider, but it is failing during connection.

For any connection error, always ensure that the basic connectivity works. Testing whether the client system has access to the database using the `iLogin` utility can help to narrow the problem.

If `iLogin` fails, verify that the connection parameters for the database (server name, port number, host name, and so on) are correct. Use the `setnet32.exe` utility to verify that the information stored in the registry is valid.

The user credentials (user ID and password) can also be set in the registry using the `setnet32.exe` utility. Verify that both the user ID and password are valid.

The majority of the connection string parameters must be included exactly as they appear in Table 7-4 on page 221. Invalid parameters are ignored and can cause an error connection. See Table 7-7 for examples of valid parameters.

Table 7-7 Valid parameters

Valid	Invalid
Data Source	DataSource, DSN
User ID	UserID, UID
Password	PWD

If you are using GLS variables (`Client_Locale` or `Db_locale`), make sure that they are set correctly. If you do not use UNICODE, the code set part of `Client_Locale` should be the same as your operating system. `Db_Locale` should be your database locale.

Failure when handling database data

Here, we explain the reason and the solution for a "`Failed when handling database data`" error.

Reason

These types of errors are caused by using the wrong data type when accessing Informix tables.

Solution

Informix OLE DB Provider requires additional functions and tables in the `sysmaster` database to handle Informix data types correctly. These objects are created when running the `coledbp.sql` SQL script.

This script is located in the `%INFORMIXDIR%\etc` directory, and must be executed against the `sysmaster` database as the `informix` user.

If you are upgrading from an older version of Client SDK, remove these objects by running the `doledbp.sql` script from the old client before using the new `coledbp.sql` script.

7.5.2 Tracing

Tracing is useful when diagnosing application problems such as SQL errors or unexpected behaviors and can even be used for performance analysis.

A developer can set the following types of tracing when using Informix OLE DB Provider:

- Informix OLE DB Trace
- SQLIDEBUG

Informix OLE DB Trace

Informix OLE DB Trace is specific to Informix OLE DB Provider. You can set the IFXOLEDBCTRACE environment variable to point to the location of the trace file, for example:

```
IFXOLEDBCTRACE=oledbtrace.txt
```

The trace file that is generated contains the calls to all the OLE DB interfaces that the provider uses.

Example 7-21 shows a typical Informix OLE DB trace file.

Example 7-21 OLE DB trace

```
Wed Jun 30 13:59:01 2010
```

```
     A44:     53C Enter Clsfact::QueryInterface
    Object Address:  0x032C2CB8
     A44:     53C Enter Clsfact::AddRef
    Object Address:  0x032C2CB8
     A44:     53C Exit    Object Address:  0x032C2CB8  hr=2

qi: 032C2CB8 (pif 032C2CB8)
     A44:     53C Enter Clsfact::Release
    Object Address:  0x032C2CB8
     A44:     53C Exit    Object Address:  0x032C2CB8  hr=1

     A44:     53C Enter Clsfact::AddRef
    Object Address:  0x032C2CB8
     A44:     53C Exit    Object Address:  0x032C2CB8  hr=2

     A44:     53C Enter Clsfact::Release
    Object Address:  0x032C2CB8
     A44:     53C Exit    Object Address:  0x032C2CB8  hr=1

     A44:     53C Enter Clsfact::QueryInterface
    Object Address:  0x032C2CB8
     A44:     53C Enter Clsfact::AddRef
    Object Address:  0x032C2CB8
     A44:     53C Exit    Object Address:  0x032C2CB8  hr=2
...
```

In addition to the OLE DB interfaces, the trace facility collects the return value for any method that is used during the OLE DB session, as shown in Example 7-22.

Example 7-22   Entry and exit points OLE DB trace

```
  A44:     53C Enter Datasrc::QueryInterface
  A44:     53C Enter Datasrc::QueryInterface
  A44:     53C Enter Datasrc::AddRef
  A44:     53C Exit    Object Address:  0x032CC370  hr=2
  A44:     53C Enter Datasrc::Release
  A44:     53C Exit    Object Address:  0x032CC370  hr=1
  A44:     53C Enter Datasrc::QueryInterface
  A44:     53C Enter Datasrc::XIDBProperties::SetProperties
  A44:     53C Exit    Object Address:  0x032CC38C  hr=0
  A44:     53C Enter Datasrc::QueryInterface
  A44:     53C Enter Datasrc::XIDBProperties::SetProperties
  A44:     53C Exit    Object Address:  0x032CC38C  hr=0
  A44:     53C Enter Datasrc::QueryInterface
  A44:     53C Enter Datasrc::XIDBInitialize::Initialize
  A44:     53C Exit    Object Address:  0x032CC384  hr=0
  A44:     53C Enter Datasrc::QueryInterface
  A44:     53C Enter Datasrc::XIDBProperties::GetProperties
```

```
 A44:     53C Exit    Object Address:  0x032CC38C  hr=0
 A44:     53C Enter Datasrc::QueryInterface
 A44:     53C Enter Datasrc::XIDBCreateSession::CreateSession
```

SQLIDEBUG

SQLI is the communication protocol used by Client SDK. SQLIDEBUG is a trace facility of all the messages between an Informix client and an Informix database server. This trace is common to all the Client SDK components (ESQL/C, ODBC, OLE DB, and so on).

You can find a description and how to enable SQLIDEBUG in 3.3.6, “Troubleshooting” on page 117.

When diagnosing a performance problem, sometimes it is useful to collect an SQLIDEBUG trace to have an idea of where the delay occurs. Using the `summary` option when processing the SQLIDEBUG file with `sqliprint` produces a detailed description of all the messages between the client and server, as shown in Example 7-23.

Example 7-23   Sqliprint summary output

```
>>>>>>>>>>>>>>>>>> SUMMARY INFORMATION <<<<<<<<<<<<<<<<<

>>>>>>>>>>TOTAL ELAPSED CLOCK TIME (sec): 0.047000

>>>>>>>>>>CLIENT ELAPSED CLOCK TIME (sec): 0.047000

>>>>>>>>>>SERVER+NETWORK CLOCK TIME (sec): 0.000000

FROM C->S
Msg          occured    Total        Avg        Min        Max
-----------------------------------------------------------------
SQ_PREPARE       1    0.000000    0.000000    0.000000    0.000000
SQ_ID            4
SQ_CLOSE         1
SQ_RELEASE       2
SQ_EOT           7
SQ_NDESCRIBE     1
SQ_SFETCH        1    0.000000    0.000000    0.000000    0.000000
SQ_WANTDONE      1
SQ_EXIT          1
SQ_INFO          1
SQ_RET_TYPE      1
SQ_INTERNALVER     1
SQ_PROTOCOLS     1
-----------------------------------------------------------------

FROM S->C
Msg          occured    Total        Avg        Min        Max
-----------------------------------------------------------------
```

```
UNKNOWN          1
SQ_DESCRIBE      1    0.000000    0.000000    0.000000    0.000000
SQ_EOT           6
SQ_TUPLE         1
SQ_TUPID         1
SQ_PROTOCOLS     1
------------------------------------------------------------------

=======================================================================
  COMMAND TEXT INFORMATION (first 99 are listed)
a(    b) CMD[##]= '  first 60 bytes of cmd text '
where a = 'P'repare, or 'N'ot prepared, and b = length of command string
-----------------------------------------------------------------------
P(   28) CMD[ 0]='SELECT rowid, * FROM state;'
-----------------------------------------------------------------------

=======================================================================
 Fetch Array feature not used
 OPTOFC feture not used
 Number of open/reoptimzation encountered = 0
 Number of C->S message send = 22
 Number of S->C message send = 21

 Number of prepare statements encountered = 1
 Number of execute statements encountered = 0
 Number of   singleton select encountered = 0
 Number of        open cursor encountered = 0
 Number of       close cursor encountered = 1

 Number of   non-blob put = 0, averge size of each   put is 0.000000
 Number of non-blob fetch = 1, averge size of each fetch is 25.000000
 Number of       blob put = 0, averge size of each   put is 0.000000
 Number of     blob fetch = 0, averge size of each fetch is 0.000000

>>>>>>>>>>>>>>> END SUMMARY INFORMATION <<<<<<<<<<<<<<<
```

8

Working with .NET data providers

This chapter describes the .NET providers that are available for working with an Informix database server. It includes the following topics:

- Informix and .NET data providers
- Setup and configuration
- Developing a .NET application
- Visual Studio Add-In for Visual Studio

8.1 Informix and .NET data providers

A .NET *data provider* is a .NET assembly that lets .NET applications access and manipulate data in a database by implementing several interfaces that follow the Microsoft ADO.NET architecture. Any application that can be executed by the Microsoft .NET Framework can use a .NET data provider.

IBM provides the following .NET data providers to work with an IBM Informix database:

- IBM Infomix .NET Provider
- IBM Data Server Provider for .NET

Both providers rely on internal calls to the ODBC and CLI drivers. They are not 100% managed-code providers.

8.2 Setup and configuration

This chapter describes the .NET providers that are available to connect to an IBM Informix database. We explain how to configure and test the connectivity with both Infomix .NET Provider and Data Server Provider for .NET.

8.2.1 IBM Informix .Net Provider

Informix .NET Provider is part of the Informix Client Software Development Kit (Client SDK). Informix .NET Provider is certified to work on both 32-bit and 64-bit editions of Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008, and Windows 7.

The provider requires that Microsoft .NET Framework SDK 1.1 or higher is already installed on the computer. The provider is installed by default when performing a complete installation of Client SDK.

Informix .NET Provider supports the following Informix database servers:

- IBM Informix 10.00, 11.10, and 11.50
- IBM Informix Extended Parallel Server 8.50 and later

The assembly name of the Informix .NET Provider is *IBM.Data.Informix*.

The assembly files are located in the Client SDK directory. The provider for a .NET 1.1 application is in the `%INFORMIXDIR%\netf11` directory. For a .NET 2.0 and above, use the provider located in the `%INFORMIXDIR\netf20` directory.

The installation process registers two strong-named assemblies in the global assembly cache (GAC).

Table 8-1 shows Informix .NET Provider assembly versions.

Table 8-1 Informix .NET Provider assembly versions

.NET framework	Assembly version
1.1	2.81.0.0
2.0 and above	3.0.0.2

The assemblies are registered in the common section of the GAC (`GAC_MSIL`). Thus, only one version of the provider (32-bit or 64-bit) can be stored in the GAC at a time.

Note: There is no 64-bit version of the .NET 1.1 Framework, so the .NET 1.1 Provider is not installed on a Windows 64-bit platform.

8.2.2 IBM Data Server Provider for .NET

Data Server Provider for .NET is included in the IBM Data Server Package. The provider supports multiple IBM data servers, including IBM Informix (11.x) and DB2.

As with Informix .NET Provider, the Data Server Provider for .NET is not 100% managed code. Thus, it requires the DB2 CLI component for communication with the database.

The following Data Server providers for an Informix database are available:

- `IBM.Data.DB2` is the preferred .NET provider when developing new applications.
- `IBM.Data.Informix` is the .NET provider assembly that is normally used to migrate an application that is developed using Informix .NET Provider.

The assembly files are located in the `netf20` subdirectory under the Data Server Client directory, which by default is `C:\Program Files\IBM\IBM DATA SERVER DRIVER\netf20`.

Note: The 64-bit version of the Data Server Client Package also includes the 32-bit drivers.

Both providers have the same assembly version, 9.0.0.2. Table 8-2 shows the Data Server Provider for .NET assembly version.

Table 8-2 Data Server Provider for .NET assembly versions

.NET Provider	Assembly version
IBM.Data.DB2	9.0.0.2
IBM.Data.Informix	9.0.0.2

8.2.3 Verifying connectivity

Informix .NET Provider uses communication libraries internally as do the other drivers that are included in Client SDK. Thus, it uses the same connectivity information that is normally stored in the registry through the use of the `setnet32.exe` utility.

When there is no specific tool to verify the connectivity with Informix .NET Provider, you can use a simple .NET application to test it.

The Microsoft .NET Framework SDK is required to use a .NET provider that contains a .NET language compiler.

Example 8-1 shows a simple C code sample that verifies the connection with the database server. This example takes the connection string as the first argument and uses it to connect to the database.

Example 8-1 A connect.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class Connect
  {
    static void Main(string[] args)
      {
         IfxConnection conn;

         if (args.Length>0) {
            try {
               conn = new IfxConnection(args[0]);
               conn.Open();
               Console.WriteLine("Connected");
               Console.WriteLine(String.Format("Server Type: {0}, Server
Version: {1}", conn.ServerType, conn.ServerVersion));
```

```
               Console.WriteLine(String.Format("Database: {0}",
conn.Database));
            }
            catch (Exception e) {
               Console.WriteLine(e.Message);
            }
         }
         else
            Console.WriteLine("Need a connection string as argument\n e.g:
\"Host=kodiak;Service=9088;Server=demo_on;Database=stores_demo;User
ID=informix;password=;\"\n");
   }
}
```

Example 8-2 shows how to compile and run Example 8-1 on page 256.

Example 8-2 Output of the connect.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.informix.dll /nologo
connect.cs
C:\work>connect "Server=demo_on;Database=stores_demo"
Connected
Server Type: Informix, Server Version: 11.50.0000 FC6
Database: stores_demo

C:\work>
```

If the connection information for the Informix server is stored in the registry, the only parameter that is required in the connection string is the name of the database server. The remainder of the information, such as Host or Service, is taken from the registry.

At compile time, you must pass the assembly as a reference to the `csc.exe` compiler using the `/R` parameter. For example:

```
/R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll
```

If you want to use the Data Server provider, the reference path is as follows:

```
/R:”C:\Program Files (x86)\IBM\IBM DATA SERVER DRIVER\bin\netf20\IBM.Data.Informix.dll”
```

Example 8-3 shows how to compile and execute the sample with Data Server Provider for .NET.

Example 8-3 Data Server connect.cs sample

```
C:\work>csc.exe /R:"C:\Program Files (x86)\IBM\IBM DATA SERVER
DRIVER\bin\netf20\IBM.Data.Informix.dll" /nologo connect.cs

C:\work>connect "Server=kodiak:9089;Database=stores_demo;User
ID=informix;password=password;"
Connected
Server Type: IDS/NT64, Server Version: 11.50.0000
Database: STORES_DEMO

C:\work>
```

Data Server Provider for .NET includes a connection tool called `testconn20.exe` that you can use to test the connection with the database. Example 8-4 shows a typical output of the `testconn20.exe` utility.

Example 8-4 Output of the testconn20.exe utility

```
C:\work>testconn20.exe "server=kodiak:9089; database=stores_demo; uid=informix;
pwd=password;"

Step 1: Printing version info
        .NET Framework version: 2.0.50727.3603
        DB2 .NET provider version: 9.0.0.2
        DB2 .NET file version: 9.7.2.2
        Capability bits: ALLDEFINED
        Build: 20100514
        Factory for invariant name IBM.Data.DB2 verified
        VSAI assembly version: 9.1.0.0
        VSAI file version: 9.7.1.53
        Elapsed: 1.015625

Step 2: Validating db2dsdriver.cfg against db2dsdriver.xsd schema file
        C:\PROGRA~1\IBM\IBMDAT~2\cfg\db2dsdriver.cfg against C:\Program
Files\IBM\IBM DATA SERVER DRIVER\cfg\db2dsdriver.xsd
        Elapsed: 0.015625

Step 3: Connecting using "server=kodiak:9089; database=stores_demo;
uid=informix; pwd=password;"
        Server type and version: IDS 11.50.0000
        Elapsed: 0.46875

Step 4: Selecting rows from informix.systables to validate existance of
packages
   SELECT * FROM informix.systables
```

```
        Elapsed: 0.171875

Step 5: Calling GetSchema for tables to validate existance of schema functions
        Elapsed: 0.21875


Test passed.

C:\work>
```

8.3 Developing a .NET application

This section describes the connection string attributes for two Informix .NET providers and provides samples of basic database operations.

8.3.1 Connecting to the database

In this section, we describe the connection string attributes for Informix .NET Provider and Data Server Provider for .NET

Informix .NET Provider connection attributes

Table 8-3 lists the connection attributes that Informix .NET Provider supports.

Table 8-3 Informix .NET Provider connection string attributes

Attribute	Description	Default
Client Locale, Client_Locale	Locale used on the application	en_us.1252
Connection Lifetime	Time in seconds that a connection is allowed in the pool	0
Database, DB	Database to connect to	
Database Locale, DB_LOCALE	Locale of the database	en_US.819
DELIMIDENT	If set, any string within double quotes (") is treated as an identifier, and any string within single quotes (’) is treated as a string literal	y
Enlist	Enables or disables automatic enlistment in a distributed transaction	true

Attribute	Description	Default
Exclusive, XCL	if set the database is open in exclusive mode	No
Host	Name or IP address of the system on which the Informix server is running	localhost
Max Pool Size	Maximum number of connections allowed in the pool	100
Min Pool Size	Minimum number of connections allowed in the pool	0
Optimize OpenFetchClose, OPTOFC	Reduces the number of client-server messages when using cursors	0
Packet Size, Fetch Buffer Size, FBS	Size in bytes of the buffers used to send data to or from the server	32767
Password, PWD	Password associated with the User ID	
Persist Security Info	If set the security-sensitive information, such as the password, is returned as part of the connection string	false
Pooling	Enable Connection Pooling	true
Protocol, PRO	Communication protocol	
Server	Name or alias of the Informix server	
Service	Service name or port number used by the server for incoming connection	
Skip Parsing	If set, there is no internal SQL parsing it increases performance but the SQL must be valid	false
UserDefinedTypeFormat	Changes the mapping of UDTs to either `DbType.String` or `DbType.Binary`	
Leave Trailing Spaces	If set, disable the automatic trailing of spaces done in a varchar column	false
User ID, UID	Login account.	

Data Server Provider for .NET connection attributes

Table 8-4 lists the most common connection attributes that Data Server Provider for .NET uses.

Table 8-4 Data Server Provider for .NET connection string attributes

Attribute	Description	Default Value
ConnectTimeout, Connect Timeout	The time (in seconds) to wait for the database connection to be established.	0
Connection Lifetime	Time in seconds that a connection is allowed in the pool.	60
Database, DB	Database to connect to.	
Enlist	Enables or disables automatic enlistment in a distributed transaction.	true
Max Pool Size	Maximum number of connections allowed in the pool.	0
Min Pool Size	Minimum number of connections allowed in the pool.	0
Password, PWD	Password associated with the User ID	
Persist Security Info	If set the security-sensitive information, such as the password, is returned as part of the connection string.	false
Pooling	Enable Connection Pooling within the .NET Provider	true
Query Timeout	Time to wait for an SQL query response	5
Server	Name or alias of the Informix server followed by service name or the port number	
User ID, UID	Login account	

The configuration parameters for Data Server Provider for .NET and most of the components are stored in the `db2sdriver.cfg` file. For a complete list of the configuration parameters in this file, refer to:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.dbclient.config.doc/doc/c0054698.html

If the connection string does not contain all the required information, the provider takes the missing arguments from the `db2sdriver.cfg` configuration file.

The majority of the connection string attributes of Data Server Provider for .NET have similar meanings and format as Informix .NET Provider string attributes except the `Server` attribute.

For Informix .NET Provider, the `Server` keyword contains the name (or alias) of the Informix server, which is normally the same as the value of the INFORMIXSERVER variable.

For Data Server Provider for .NET, the `Server` keyword contains the host name where the Informix server is running and the service name or port number that is used to listen for DRDA client connections.

Example 8-5 shows the `Server` attribute format. In this example, we connect to an Informix database server that is running in a system named `kodiak.ibm.com`. The Informix server DRDA alias is using the port number `9089`.

Example 8-5 Server attribute for the Data Server Provider for .NET

```
C:\work>connect
"Server=kodiak.ibm.com:9089;Database=stores_demo;UID=informix;password=password
;"
Connected
Server Type: IDS/NT64, Server Version: 11.50.0000
Database: STORES_DEMO

C:\work>
```

8.3.2 Data type mapping

The data types that are used by the .NET Framework differ from the data types that are used by the IBM Informix database. In this section, we describe the optimal data types to use when accessing data in an Informix database. The optimal data type depends on the .NET method that is used to access the data.

You can use types other than those that we describe here. For example, you can use the `IfxDataReader.GetString` method to obtain any Informix data type. The types that we mention here are the most efficient and least likely to change in value in data conversion.

Informix .NET type mapping

Table 8-5 shows the mapping for the specific IBM Informix data types.

Table 8-5 Type mapping for Informix specific data

Informix data type	For data reader	For data set
BIGINT	Int64	Int64
BIGSERIAL	Int64	Int64
BLOB	IfxBlob	Byte[]
BOOLEAN	Boolean	Boolean
BYTE	Byte[]	Byte[]
CLOB	IfxClob	Byte[]
COMPLEX (ROW, LIST)	String	String
DECIMAL(p<=28) fixed scale	IfxDecimal	Decimal
DECIMAL(p<=28) floating point	IfxDecimal	Double
DECIMAL (p>28)	IfxDecimal	String
IDSSECURITYLABEL	Int64[]	Int64[]
INT8	Int64	Int64
INTERVAL, year-month	IfxMonthSpan	String
INTERVAL, day-fraction	IfxTimeSpan	TimeSpan
LVARCHAR	String	String
MONEY	IfxDecimal	Decimal with same precision
NCHAR	String	String
SERIAL	nt32	Int32
SERIAL8	Int64	Int64
TEXT	String	String

Data Server Provider for .NET type mapping

Table 8-6 shows the type mapping with the Data Server Provider for .NET when using specific Informix data types.

Table 8-6 Type mapping for Informix specific data

Informix data type	Optimal for data reader	Optimal for data set
LVARCHAR	`IfxString`	String
BLOB, BYTE	`IfxBlob`	Byte[]
CLOB, TEXT	`IfxClob`	String
BOOLEAN, SMALLINT	`IfxInt16`	Int16
BIGINT, BIGSERIAL, INT8, SERIAL8	`IfxInt64`	Int64
DECIMAL(p<=28) fixed scale	`IfxDecimal`	Decimal
DECIMAL(p<=28) floating point	`IfxDouble`	Double
DECIMAL (p>28)	`IfxDouble`	Double
MONEY	`IfxDecimal`	Decimal

8.3.3 Performing database operations

This section describes the main classes in the Informix .NET Provider and demonstrates how to use these classes with basic examples.

IfxConnection

The `IfxConnection` object represents a unique connection with the database server. You can specify the connectivity details about the database when creating the connection object or by setting the `ConnectionString` property later.

Because Informix .NET Provider uses native resources (that are not managed by the .NET CLR), always close or dispose any open `IfxConnection` class when it is not longer needed.

Table 8-7 lists the public method of the `IfxConnection` class.

Table 8-7 IfxConnection public methods

Method	Description
`void Open()`	Opens a database connection
`void Close()`	Closes the connection to the database
`void ChangeDatabase(String)`	Changes the current database
`void EnlistTransaction()`	Enlists the connection in a DTC transaction
`IfxTransaction BeginTransaction()` `BeginTransaction(IsolationLevel)`	Begins a database transaction
`IfxCommand CreateCommand()`	Creates an instance of an `IfxCommand` object that is associated with this `IfxConnection`
`IfxBlob GetIfxBlob()`	Creates an instance of an `IfxBlob` object that is associated with this connection
`IfxClob GetIfxClob()`	Creates an instance of an `IfxClob` object that is associated with this connection

We described the connection string attributes in 8.3.1, “Connecting to the database” on page 259. Example 8-6 demonstrates how to connect to the database

Example 8-6 The connect_sample.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class sample {
    static void Main(string[] args) {

            IfxConnection conn;

            try {
               conn = new IfxConnection("Server=demo_on;database=stores_demo");
               conn.Open();
               Console.WriteLine("Connected to "+conn.Database);
               conn.Close();
            }
            catch (Exception e) {
               Console.WriteLine(e.Message);
            }
   }
}
```

Example 8-7 shows the output of Example 8-6.

Example 8-7 Output of the connect_sample.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /nologo
connect_sample.cs

C:\work>connect_sample
Connected to stores_demo

C:\work>
```

IfxCommand

The `IfxCommand` object represents an SQL statement or stored procedure to execute against a database server.

Table 8-8 shows methods that the `IfxCommand` class provides for executing an SQL statement.

Table 8-8 IfxCommand public methods

Method	Description
`Int32 ExecuteNonQuery()`	Executes an SQL statement against the `IfxConnection` object
`IfxDataReader ExecuteReader()` `ExecuteReader(DataCommandBehavior behavior)`	Executes the command in the `CommandText` property against the `IfxConnection` object and builds an `IfxDataReader` object
`Object ExecuteScalar()`	Executes the query, and returns the first column of the first row
`void Cancel()`	Attempts to cancel the execution of a command
`IfxParameter CreateParameter()`	Creates a new instance of an `IfxParameter` object
`void Prepare()`	Creates a prepared (or compiled) version of the command against the database

Example 8-8 demonstrates how to run a simple DELETE SQL statement.

Example 8-8 The command_sample.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class sample {
    static void Main(string[] args) {

       IfxConnection conn;
       int drows=0;

       try {
          conn = new IfxConnection("Server=demo_on;database=stores_demo");
          conn.Open();

          IfxCommand cmmd = conn.CreateCommand();
          cmmd.CommandText = "DELETE FROM customer WHERE customer_num = 103";
          drows = cmmd.ExecuteNonQuery();
          Console.WriteLine("Deleted rows: "+drows.ToString());
          conn.Close();
       }
       catch (Exception e) {
          Console.WriteLine(e.Message);
       }
   }
}
```

Example 8-9 shows the output of Example 8-8.

Example 8-9 Output of the command_sample.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /nologo
command_sample.cs


C:\work>command_sample
Deleted rows: 1

C:\work>
```

IfxDataAdapter

The `IfxDataAdapter` class represents a set of data commands that are used to communicate between a data set and the database. A *data set* is a copy of the database data that is stored in memory.

Table 8-9 lists the methods that the `IfxDataAdapter` class provides for accessing the data.

Table 8-9 IfxDataAdapter public methods

Method	**Description**
`Int32 Fill(DataSet)`	Adds or refreshes rows in the data set
`DataTable FillSchema(DataSet, SchemaType)`	Adds a data table to the data set, and configures the schema to match that in the database based on the specified `SchemaType`
`IDataParameter GetFillParameters()`	Returns the parameters set by the user when executing a SELECT statement
`Int32 Update(DataSet)`	Executes the SQL statement that is associated with the `InsertCommand`, `UpdateCommand`, or `DeleteCommand` for each inserted, updated, or deleted row in the specified data set

Example 8-10 demonstrates how to retrieve data from the database using the `IfxDataAdapter` class. In this example, we retrieve the data rows directly from the data set.

Example 8-10 The dataadapter_sample.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class sample {
   static void Main(string[] args) {

     IfxConnection conn;

     conn = new IfxConnection("Server=demo_on;database=stores_demo");
     conn.Open();

     IfxCommand cmmd = conn.CreateCommand();
     cmmd.CommandText = "SELECT * FROM state WHERE code='CA'";

     IfxDataAdapter dadap = new IfxDataAdapter();
     DataSet dset = new DataSet();
     dadap.SelectCommand = cmmd;
     dadap.Fill(dset);

     foreach(DataRow dr in dset.Tables[0].Rows) {
       Console.WriteLine(String.Format("\tCode\tState\n"));
       Console.WriteLine(String.Format("\t{0}\t{1}", dr["code"], dr["sname"]));
```

```
      }

      conn.Close();

    }
}
```

Example 8-11 shows the data that is returned by the data set object.

Example 8-11 Output of the dataadapter_sample.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /nologo
dataadapter_sample.cs

C:\work>dataadapter_sample.exe
        Code    State

        CA      California

C:\work>
```

IfxDataReader

The `IfxDataReader` class provides fast read-only, forward-only access to a set of rows, similar to SQL cursors data. To create an `IfxDataReader` object, the application calls the `ExecuteReader()` method of the `IfxCommand` object.

The `IfxDataReader` class is a part of the `System.Data.Common.DbDataReader`. It provides all common `DbDataReader` methods as well as additional methods to handle Informix specific data types.

Table 8-10 contains the additional public methods of the `IfxDataReader` class.

Table 8-10 Additional methods of IfxDataReader

Method	Description
`Boolean GetBoolean()`	Gets the value of the specified column as a Boolean
`TimeSpan GetTimeSpan()`	Gets the time span value of the specified field
`IfxBlob GetIfxBlob()`	Gets the `IfxBlob` value of the specific field
`IfxClob GetIfxClob()`	Gets the `IfxClob` value of the specific field
`IfxDateTime GetIfxDateTime()`	Gets the `IfxDateTime` value of the specific field
`IfxDecimal GetIfxDecimal`	Gets the `GetIfxDecimal` value of the specific field

Method	Description
`IfxMonthSpan GetIfxMonthSpan`	Gets the `IfxMonthSpan` value of the specific field
`IfxTimeSpan GetIfxTimeSpan`	Gets the `IfxTimeSpan` value of the specific field

Refer to *IBM Informix .NET Provider Reference Guide* for a complete list of all the `IfxDataReader` public methods:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.netpr.doc/netpr031023156.htm#netpr031023156

Note: Even if the `IfxDataReader.Read()` method returns only one row each time it is called, the provider can retrieve more than one row from the database. Informix .NET Provider does this to increase performance. The application can set the number of rows that the provider fetches from the database using the `IfxCommand` class `RowFetchCount` property.

Example 8-12 demonstrates how to use a `IfxDataReader` class to access a database table.

Example 8-12 The datareader_sample.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class sample {
   static void Main(string[] args) {

     IfxConnection conn;

     conn = new IfxConnection("Server=demo_on;database=stores_demo");
     conn.Open();

     IfxCommand cmmd = conn.CreateCommand();
     cmmd.CommandText = "SELECT * FROM state WHERE code='CA'";

     IfxDataReader drdr;
     drdr = cmmd.ExecuteReader();
     while (drdr.Read()) {
       Console.WriteLine(String.Format("\tCode\tState\n"));
       Console.Write(String.Format("\t{0}", drdr.GetString(0)));
       Console.WriteLine(String.Format("\t{0}", drdr.GetString(1)));
     }

     drdr.Close();
     conn.Close();
```

```
    }
}
```

Example 8-13 shows the output of Example 8-12 on page 270.

Example 8-13 Output of the datareader_sample.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /nologo
datareader_sample.cs

C:\work>datareader_sample.exe
        Code    State

        CA      California

C:\work>
```

IfxError

The `IfxError` class collects information that is relevant to a warning or error that the database returns. You can use the `IfxError` and `IfxErrorCollection` classes to retrieve additional information when an error occurs.

An application can access this information using the properties of the `IfxError` object listed in Table 8-11.

Table 8-11 IfxError properties

Property	**Type**	**Description**
`Message`	String	Text message
`NativeError`	Int32	IBM Informix error code
`SQLState`	String	ANSI SQL error code

Example 8-14 demonstrates a typical use of the `IfxError` class. The `IfxException` class contains one or more `IfxError` objects.

Example 8-14 The error_sample.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class Connect  {
    static void Main(string[] args) {

        IfxConnection conn;
        int drows=0;

        try {
           conn = new IfxConnection("Server=demo_on;database=wrong_db");
           conn.Open();
           Console.WriteLine("Connected");
           conn.Close();
        }
        catch (IfxException e) {

          Console.WriteLine("----------------------------");
          if (e.Errors.Count > 0) {

             IfxError ifxErr = e.Errors[0];
             Console.WriteLine("Message :" + ifxErr.Message);
             Console.WriteLine("Native error :" + ifxErr.NativeError);
             Console.WriteLine("SQL state    :" + ifxErr.SQLState );
          }
          Console.WriteLine(e.StackTrace);
          Console.WriteLine("----------------------------");
        }
    }
}
```

IfxParameter

The `IfxParameter` class is used to pass parameters to the `IfxCommand` method. `IfxParameter` classes are stored as a collection in a `IfxParameterCollection` object.

Table 8-12 contains the public attributes of an `IfxParameter` object.

Table 8-12 IfxParameter attributes

Property	Type	Description
`DbType`	DbType	Defines the DbType of the parameter
`Direction`	ParameterDirection	Defines the direction of the parameter (Input, Output, or both)
`IfxType`	IfxType	Defines the `IfxType` class of the parameter
`IsNullable`	Boolean	Specifies whether the parameter is allowed to be null
`ParameterName`	String	Name of the parameter, which is a unique reference in the parameter collection

Property	Type	Description
`SourceColumn`	String	Specifies the column that is mapped to the data set
`SourceVersion`	DataRowVersion	Defines the `DataRowVersion` to use when you load `Value`
`Value`	Object	Defines the value of the parameter

Example 8-15 shows how to use `IfxParameter` class to perform an UPDATE operation.

Example 8-15 The parameter_sample.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;

class sample {
    static void Main(string[] args) {

       IfxConnection conn;
       int urows=0;

       try {
          conn = new IfxConnection("Server=demo_on;database=stores_demo");
          conn.Open();

          IfxCommand cmmd = conn.CreateCommand();
          cmmd.CommandText = "UPDATE state SET sname = ? where code = ?";
          IfxParameter pcode = new IfxParameter("code", DbType.String);
          IfxParameter psname = new IfxParameter("sname", DbType.String);

          pcode.Value="CA";
          psname.Value="CALIFORNIA";
          cmmd.Parameters.Add(psname);
          cmmd.Parameters.Add(pcode);

          urows = cmmd.ExecuteNonQuery();
          Console.WriteLine("Updated rows: "+urows.ToString());

          conn.Close();
       }
       catch (Exception e) {
          Console.WriteLine(e.Message);
       }
   }
}
```

Example 8-16 shows the output of Example 8-15 on page 273.

Example 8-16 Output of the parameter_sample.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /nologo
parameter_sample.cs

C:\work>parameter_sample.exe
Updated rows: 1

C:\work>
```

IfxTransaction

The `IfxTransaction` class represents an SQL transaction to be made at a database. The application creates an transaction by calling the `BeginTransaction` method of the `IfxConnection` object.

The `IfxConnection.BeginTransaction()` method returns an `IfxTransaction` object. When the application decides how to resolve the transaction, it uses the `Commit` and `Rollback` methods of the `IfxTransaction` object.

Any `IfxCommand` that is involved in a transaction must have the transaction property set to the `IfxTransaction` object.

Example 8-17 demonstrates how to create a transaction.

Example 8-17 Creating a transaction using the IfxTransaction class

```
using System;
using System.Data;
using IBM.Data.Informix;

class sample {
    static void Main(string[] args) {

       IfxConnection conn;
       int drows=0;

       try {
          conn = new IfxConnection("Server=demo_on;database=stores_demo");
          conn.Open();

          IfxTransaction trans = conn.BeginTransaction();

          IfxCommand cmmd = conn.CreateCommand();
          cmmd.Transaction = trans;
          cmmd.CommandText = "DROP TABLE state";
          drows = cmmd.ExecuteNonQuery();
```

```
            trans.Rollback();
            conn.Close();
         }
         catch (Exception e) {
            Console.WriteLine(e.Message);
         }
   }
}
```

You can specify the following isolation levels with the `BeginTransaction()` method:

- ► `ReadUncommitted`
- ► `ReadCommitted`
- ► `RepeatableRead`
- ► `Serializable`

The default Isolation level is `ReadCommited`.

Example 8-18 demonstrates how to run a distributed transaction.

Example 8-18 The transact_cts.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;
using System.Transactions;

class sample {
    static void Main(string[] args) {

       IfxConnection conn;

       try {
          conn = new IfxConnection("Server=demo_on;database=stores_demo");

  // Transaction options
          TransactionOptions tsopt = new TransactionOptions();
          tsopt.IsolationLevel =
                        System.Transactions.IsolationLevel.RepeatableRead;
          tsopt.Timeout = new TimeSpan(0, 60, 0);

          using (TransactionScope tscope = new TransactionScope(
                    TransactionScopeOption.RequiresNew, tsopt,
                    EnterpriseServicesInteropOption.Full))
          {

             conn.Open();
```

```
            IfxCommand cmmd = conn.CreateCommand();
            cmmd.CommandText = "DROP TABLE state";
            cmmd.ExecuteNonQuery();

  // Rollback the distributed transaction not calling Complete()
  //         tscope.Complete();
         }
      }
      catch (Exception e) {
         Console.WriteLine(e.Message);
      }
   }
}
```

8.3.4 Handling Informix specific data types

In this section, we discuss how to work with IBM Informix specific data types such as smart large objects or decimal using Informix .NET Provider.

IfxBlob and IfxClob

CLOB and BLOB are Informix data types that are used to store large amounts of character or binary data.The application handles these smart large objects using the `IfxClob` and `IfxBlob` classes.

The method that is implemented on these classes allows random access of its contents. The application can read or write to certain positions in the large object without reading or writing through all of the data up to that position.

BLOBs and CLOBs are both smart large object type. Both types share many of the same methods. BLOBs differ from CLOBs in that the data in a CLOB is treated as text characters but the data in a BLOB is not. The data in a BLOB is considered to be binary data, and no translation or conversion of any kind is performed on it when it is moved to or from the database server.

Properties and methods

Table 8-13 lists the `IfxBlob` and `IfxClob` public properties.

Table 8-13 Properties of IfxBlob and IfxClob

Property	Type	Description
`EstimatedSize`	Int64	Estimates final size of the large object
`ExtentSize`	Int32	Finds the next extent size for this large object (disk space)

Property	Type	Description
`Flags`	Int32	Flags for this large object
`IsNull`	Boolean	Determines whether the large object is NULL
`IsOpen`	Boolean	Determines whether the large object is open
`LastAccessTime`	Int32	Determines the last time that the large object was accessed
`LastChangeTime`	Int32	Determines the last time that the status was changed
`LastModificationTime`	Int32	Determines the last time that the large object was modified
`MaxBytes`	Int64	Defines the maximum size for the large object
`Position`	Int64	Defines the current position on the large object
`ReferenceCount`	Int32	Provides the number of records in the database that currently contain a reference to this large object
`SBSpace`	String	Defines the Sbspace in which the large object is stored
`Size`	Int64	Defines the size of the large object in bytes

Table 8-14 lists the available public methods from the `IfxClob` and `IfxBlob` classes.

Table 8-14 Public methods of IfxClob and IfxBlob

Method	Description
`void Open(mode)`	Opens a large object in a specific mode
`IfxSmartLOBLocator GetLocator()`	Returns the `IfxSmartLOBLocator` that is associated with this instance
`Int64 Read(buff), Read(buff, buffOffset, numBytesToRead, sLOBOffset, whence)`	Reads the complete data or a portion of a large object as a `Byte[]` or `Char[]`
`Int64 Write(buff), Write(buff, buffOffset, numBytesToWrite, sLOBOffset, whence)`	Writes the complete buffer or a portion into a large object
`void Truncate(offset)`	Truncates everything in the large object past the position offset

Method	Description
`Int64 Seek(offset, whence)`	Changes the current position within the large object
`void FromFile(filename, appendTosLOB, fileLocation)`	Reads an operating system file and writes the complete content into the large object.
`String ToFile(filename, mode, fileLocation)`	Writes the contents of the large object to an operating system file
`void Lock(sLOBOffset, whence, range, lockMode)`	Locks the complete large object or only a portion of the large object
`void Unlock(sLOBOffset, whence, range)`	Unlocks a large object
`void Release()`	Frees database server resources
`void Close()`	Closes the large object

Creating a smart large object

You can use the `GetIfxClob()` and `GetIfxBlob()` methods of the `IfxConnection()` object to create a large object. To create a large object:

1. Create an instance of a the large object, `IfxClob` or `IfxBlob`.
2. Open the large object.
3. Write data into the large object.
4. Execute the SQL Statement using the `IfxClob` or `IfxBlob` classes as a parameter for the statement.
5. Close the large object.

Example 8-19 demonstrates how to update a CLOB column. The code reads a text file, `sample_blob.cs`, and uses the contents for the CLOB data.

Example 8-19 The lo_create.cs sample

```
using System;
using System.IO;
using System.Data;
using IBM.Data.Informix;

class sample {
    static void Main(string[] args) {

     IfxConnection conn;
     int urows=0;
     IfxClob vclob;
     char[] vclobBuff;
     try {

       conn = new IfxConnection("Server=demo_on;Database=stores_demo");
       conn.Open();

// Create and Open the Clob
       vclob = conn.GetIfxClob();
       vclob.Open(IfxSmartLOBOpenMode.ReadWrite);

// Read a text file and insert into the clob
       vclobBuff = File.ReadAllText("lo_create.cs").ToCharArray();
       vclob.Write(vclobBuff);

// Create the UPDATE Ifxcommand
       IfxCommand cmmd = conn.CreateCommand();
       cmmd.CommandText = "UPDATE catalog set advert_descr = ? WHERE" +
                               " catalog_num = ?;";

// Bind the IfxClob value and execute the UPDATE statement
       IfxParameter padvt_desc  = new IfxParameter(null, vclob);
       IfxParameter pcatalog_n = new IfxParameter(null, "10072");
       cmmd.Parameters.Add(padvt_desc);
       cmmd.Parameters.Add(pcatalog_n);
       urows = cmmd.ExecuteNonQuery();

       Console.WriteLine("Updated rows: "+urows.ToString());

// Close the IfxClob object
       vclob.Close();
       conn.Close();
      }
      catch (Exception e) {
        Console.WriteLine(e.Message);
      }
   }
}
```

If the application requires more control over the smart large object, it can use the specific properties of the `ifxClob` and `IfxBlob` classes to set up values, such as in which smart BLOB space the large object should create or the maximum size.

Example 8-20 Illustrates how to use the properties of the `IfxBlob` class.

Example 8-20 Large object extended properties

```
...
// Create and Open the Clob
       vclob = conn.GetIfxClob();
       vclob.EstimatedSize = 5000;
       vclob.ExtentSize = 1000;
       vclob.Flags = (int) IfxSmartLOBCreateTimeFlags.DontKeepAccessTime |
                     (int) IfxSmartLOBCreateTimeFlags.NoLog;
       vclob.MaxBytes = 10000;
       vclob.SBSpace = "sbspace";
       vclob.Open(IfxSmartLOBOpenMode.ReadWrite);
...
```

Selecting a smart large object

Reading a large object using the large object extensions requires the following steps:

1. Execute the SQL statement, and retrieve the large object column.
2. Open the large object.
3. Read from the large object into an application buffer.
4. Close the large object.

Example 8-21 shows how to read a large object column from a database table.

Example 8-21 The lo_select.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;


class sample {
    static void Main(string[] args) {

     IfxConnection conn;
     IfxClob vclob;
     int maxSize = 2000;
     char[] vclobBuff = new char[maxSize];


     try {

       conn = new IfxConnection("Server=demo_on;Database=stores_demo");
       conn.Open();

// Select the large object from the database
       IfxCommand cmmd = conn.CreateCommand();
       cmmd.CommandText = "SELECT * FROM catalog WHERE catalog_num = 10072";
       IfxDataReader drdr;
       drdr = cmmd.ExecuteReader();
       while (drdr.Read()) {

// Get the large object. The clob is the 6th column
      vclob = drdr.GetIfxClob(5);

// Open the large object
          vclob.Open(IfxSmartLOBOpenMode.ReadOnly);
          vclob.Read(vclobBuff, 0, maxSize, 0, IfxSmartLOBWhence.Current);
          Console.WriteLine(vclobBuff);

// Close the large object
          vclob.Close();
       }

       conn.Close();
      }
      catch (Exception e) {
        Console.WriteLine(e.Message);
      }
   }
}
```

Random access

One of the benefits of smart large objects over a normal large object is the option of reading partial data.

The `Read(buff, buffOffset, BytesToRead, loOffset, whence)` method allows you to position the data pointer anywhere in the large object data and read from there.

Example 8-22 shows how to read the last 10 bytes of a large object.

Example 8-22 Partial large object read

```
...
// Read only the last 10 bytes of large object
           int  bytesToRead = 10;
           vclob.Read(vclobBuff, 0, bytesToRead, (-1) * bytesToRead,
                                        IfxSmartLOBWhence.End);
...
```

Smart large objects support random I/O access. Using the `Seek(offset, whence)` method, you can position the data pointer anywhere in the large object and perform I/O operations with the data.

Example 8-23 demonstrates how to update only a portion of a large object using the `Seek()` method.

Example 8-23 A lo_seek.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;


class sample {
    static void Main(string[] args) {
     IfxConnection conn;
     IfxClob vclob;
     int urows=0;
     int maxSize = 100;
     char[] vclobBuff = new char[maxSize];

     try {

       conn = new IfxConnection("Server=demo_on;Database=stores_demo");
       conn.Open();

// Select the large object from the database
       IfxCommand cmmd = conn.CreateCommand();
       cmmd.CommandText = "SELECT * FROM catalog WHERE catalog_num = 10072";
       IfxDataReader drdr;
       drdr = cmmd.ExecuteReader();
       drdr.Read();

// Get the large object. The clob is the 6th column
       vclob = drdr.GetIfxClob(5);

// Open the large object
       vclob.Open(IfxSmartLOBOpenMode.ReadWrite);

// Move the pointer 15 bytes from the beginning of the large object.
       vclob.Seek(15,IfxSmartLOBWhence.Begin);

// Update the large object.
       vclobBuff="//------------//".ToCharArray();
       vclob.Write(vclobBuff);
       drdr.Close();

// Update the column in the table
       cmmd.CommandText = "UPDATE catalog set advert_descr = ? WHERE" +
                               " catalog_num = ?;";

// Bind the IfxClob value and execute the UPDATE statement
       IfxParameter padvt_desc  = new IfxParameter(null, vclob);
       IfxParameter pcatalog_num = new IfxParameter(null, "10072");
       cmmd.Parameters.Add(padvt_desc);
       cmmd.Parameters.Add(pcatalog_num);
       urows = cmmd.ExecuteNonQuery();

       Console.WriteLine("Updated rows: "+urows.ToString());

// Close the large object
       vclob.Close();

       conn.Close();
      }
      catch (Exception e) {
        Console.WriteLine(e.Message);
      }
   }
}
```

Note: Data Server Provider for .NET does not support random I/O access.

IfxDateTime

The `IfxDateTime` class represents the Informix DATETIME data type. The `IfxDateTime` class provides support for all the precisions that are allowed in an Informix DATETIME.

The Informix DATETIME data type is composed of the following time units:

- Year
- Month
- Day
- Hour
- Minute
- Second
- Fractions of a second

In an Informix database, the maximum precision for a DATETIME column is `YEAR TO FRACTION(5)`. A DATETIME column can be defined with any precision, from a year to fraction of a second allowing any subset of these units.

Table 8-15 lists the public properties of the `IfxDateTime` class.

Table 8-15 Public properties of the IfxDateTime class

Property	Type	Description
Date	`IfxDateTime`	A Year to Day `IfxDateTime` instance
Day	`Int32`	The day portion of the value
EndTimeUnit	`IfxTimeUnit`	The end time unit of the instance
Hour	`Int32`	The hour portion of the value
MaxValue	`IfxDateTime`	Maximum value allowed for this `IfxDateTime`
Millisecond	`Int32`	Millisecond unit in this `IfxDateTime`
MinValue	`IfxDateTime`	Smallest value allowed for this `IfxDateTime`
Minute	`Int32`	Minute unit in this `IfxDateTime`
Month	`Int32`	Month unit in this `ifxDateTime`
Now	`IfxDateTime`	Current date with a range of Year to Fraction (5)
Second	`Int32`	Second unit of this `IfxDateTime`
StartTimeUnit	`IfxTimeUnit`	Start time unit of this `IfxDateTime`
Ticks	`Int64`	Ticks from midnight on 1 Jan 0001
Today	`IfxDateTime`	Current time with a range of Year to Day

Property	Type	Description
Year	`Int32`	Year unit of the value

Table 8-16 lists the public methods of the `IfxDateTime` class.

Table 8-16 Public methods of the IfxDateTime class

Method	Description
`Add(IfxTimeSpan or IfxMonthSpan)`	Current value plus by object passed
`AddDays(days)`	Current value plus days passed
`AddMilliseconds(milliseconds)`	Current value plus milliseconds passed
`AddMinutes(minutes)`	Current value plus minutes passed
`AddMonths(months)`	Current value plus months passed
`AddSeconds(seconds)`	Current value plus seconds passed
`AddYears(years)`	Current value plus years passed
`Compare(ifxDT1, ifxDT1)`	Compare two `IfxDateTime` instances
`CompareTo(obj)`	Compare two `IfxDateTime` instances
`DaysInMonth(year, month)`	Number of days in the month of the year
`Equals(ifxDT1, ifxDT2)`	True if `ifxDT1` is equal to `ifxDT2`
`GreaterThan(ifxDT1, ifxDT2)`	True if `ifxDT1` is later than `ifxDT2`
`GreaterThanOrEqual(ifxDT1, ifxDT2)`	True if `ifxDT1` is later or equal than `ifxDT2`
`LessThan(ifxDT1, ifxDT2)`	True if `ifxDT1` is earlier than `ifxDT2`
`LessThanOrEqual(ifxDT1, ifxDT2)`	True if `ifxDT1` is earlier or equal than `ifxDT2`
`NotEquals(ifxDT1, ifxDT2)`	True if `ifxDT1` is different than `ifxDT2`
`Parse(dateTimeStr)` `Parse(dateTimeStr, start, end)` `Parse(dateTimeStr, format, start, end)`	New `IfxDateTime` with a value based on `dateTimeStr`
`ToString(), ToString (format)`	Value of the instance as a string

To access a DATETIME column in the database, the application uses the `IfxGetDateTime()` method from the `IfxDataReader()` object.

Example 8-24 demonstrates how to select a DATETIME data type from a table.

Example 8-24 The datetime.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;


class sample {
    static void Main(string[] args) {

     IfxConnection conn;
     IfxDateTime vdtime;


     try {

        conn = new IfxConnection("Server=demo_on;Database=stores_demo");
        conn.Open();

// Select the datetime from the database
        IfxCommand cmmd = conn.CreateCommand();
        cmmd.CommandText = "SELECT * FROM cust_calls WHERE "
                            + "customer_num = 106";
        IfxDataReader drdr;
        drdr = cmmd.ExecuteReader();
        drdr.Read();

// Get the IfxDateTime from the recordset, 2nd columnd
        vdtime = drdr.GetIfxDateTime(1);
        Console.WriteLine("Call:\t"+vdtime);
        Console.WriteLine("Hour :\t "+vdtime.Hour);
        Console.WriteLine("Minute:\t"+vdtime.Minute);

        conn.Close();
     }
     catch (Exception e) {
        Console.WriteLine(e.Message);
     }
   }
}
```

Example 8-25 shows the `Hour` and `Minute` units extracted from the `IfxDateTime` object.

Example 8-25 Output of the datetime.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /nologo
datetime.cs

C:\work>datetime
Call:   2008-06-12 08:20
```

```
Hour :   8
Minute: 20

C:\work>c
```

Arithmetic operations between two `IfxDateTime` objects might return an `lfxDateSpan()` object or an `IfxMonthSpan()` object, depending on the precision that is used by the `IfxDateTime` objects.

Example 8-26 shows the use of the `IfxDateSpan()` class.

Example 8-26 The datetime2.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;


class sample {
    static void Main(string[] args) {

     IfxConnection conn;
     IfxDateTime vdtime,vrestime;
     IfxTimeSpan vtimetaken;

     try {

        conn = new IfxConnection("Server=demo_on;Database=stores_demo");
        conn.Open();

// Select the datetime from the database
        IfxCommand cmmd = conn.CreateCommand();
        cmmd.CommandText = "SELECT * FROM cust_calls WHERE "
                             + "customer_num = 106";
        IfxDataReader drdr;
        drdr = cmmd.ExecuteReader();
        drdr.Read();

// Get the IfxDateTime from the recordset
        vdtime = drdr.GetIfxDateTime(1);
        vrestime = drdr.GetIfxDateTime(5);

// Stores the difference in an IfxTimeSpan object.
        vtimetaken=vrestime-vdtime;
        Console.WriteLine("Initiated at:\t"+vdtime);
        Console.WriteLine("Resolved at:\t"+vrestime);
        Console.WriteLine("Minutes taken:\t"+vtimetaken.Minutes);

        conn.Close();
     }
     catch (Exception e) {
        Console.WriteLine(e.Message);
     }
   }
}
```

IfxDecimal

An `IfxDecimal` object represents the Informix decimal data type. The DECIMAL data type on an Informix database can take two forms:

- DECIMAL(p) floating point
- DECIMAL(p,s) fixed point

The IfxDecimal object in Informix .NET Provider supports both versions.

Table 8-17 contains the description of the public properties of an `IfxDecimal` object.

Table 8-17 Public properties of the IfxDecimal class

Property	Type	Description
`E`	IfxDecimal	Irrational number e
`IsFloating`	Boolean	Whether it is a floating number
`IsNull`	Boolean	Whether it is NULL
`IsPositive`	Boolean	Whether it is positive
`MaxPrecision`	Byte	Maximum precision supported (32)
`MaxValue`	IfxDecimal	Largest value allowed
`MinusOne`	IfxDecimal	-1
`MinValue`	IfxDecimal	Smallest value allowed
`Null`	IfxDecimal	Null
`One`	IfxDecimal	1
`Pi`	IfxDecimal	Irrational number pi
`Zero`	IfxDecimal	0

Table 8-18 contains the description for the `IfxDecimal` public methods.

Table 8-18 Public methods of the IfxDecimal object

Method	Description
`Abs(IfxDec)`	Absolute value of `IfxDec`
`Add(IfxDec1, IfxDec2)`	Sum of `IfxDec1` and `IfxDec2`
`Ceiling(IfxDec)`	Smallest integer that is not less than `IfxDec`
`Clone()`	Creates a duplicate of this instance

Method	Description
`Compare(IfxDec1, IfxDec2)`	Compares two `IfxDecimal` values
`CompareTo(obj)`	Compares current instance with object
`Divide(Dividend, Divisor)`	Dividing result for Dividend by Divisor
`Equals(obj)`	Equal
`Equals(IfxDec1, IfxDec2)`	True if `IfxDec1` is the same as `IfxDec2`
`Floor(IfxDec)`	Largest integer not larger than `IfxDec`
`GreaterThan(IfxDec1, IfxDec2)`	True if `IfxDec1 > IfxDec2`
`GreaterThanOrEqual(IfxDec1,IfxDec2)`	True if `IfxDec1 >= IfxDec2`
`LessThan(IfxDec1, IfxDec2)`	True if `IfxDec1 < IfxDec2`
`LessThanOrEqual(IfxDec1, IfxDec2)`	True if `IfxDec1 <= IfxDec2`
`Max(IfxDec1, IfxDec2)`	Returns whichever is larger, `IfxDec1` or `IfxDec2`
`Min(IfxDec1, IfxDec2)`	Whichever is smaller, `IfxDec1` or `IfxDec2`
`Modulo(a, b)`	Returns the remainder
`Multiply(IfxDec1, IfxDec2)`	Returns `IfxDec1` times `IfxDec2`
`Negate(IfxDec)`	Current value negated
`NotEquals(IfxDec1, IfxDec2)`	True if `IfxDec1<>IfxDec2`
`Parse(DecString)`	Returns a new `IfxDecimal` based on `DecString`
`Remainder(a, b)`	Remainder of the integer division of a by b
`Round(IfxDec1, FractionDigits)`	Value of `IfxDec1` rounded to `FractionDigits`
`Subtract(IfxDec1, IfxDec2)`	Returns `IfxDec1` minus `IfxDec2`
`ToString(), ToString (format)`	Returns the current value as a string
`Truncate(IfxDec1, FractionDigits)`	Round() with truncation

An application uses the `GetIfxDecimal()` method from the `IfxDataReader()` class to access a DECIMAL column from the database.

Example 8-27 demonstrates how to retrieve a DECIMAL column

Example 8-27 The decimal.cs sample

```
using System;
using System.Data;
using IBM.Data.Informix;


class sample {
    static void Main(string[] args) {

     IfxConnection conn;
     IfxDecimal vtax;

     try {

        conn = new IfxConnection("Server=demo_on;Database=stores_demo");
        conn.Open();

// Select the datetime from the database
        IfxCommand cmmd = conn.CreateCommand();
        cmmd.CommandText = "SELECT * FROM state WHERE "
                             + "code = 'CA'";
        IfxDataReader drdr;
        drdr = cmmd.ExecuteReader();
        drdr.Read();

// Get the IfxDecimal from the recordset
        vtax = drdr.GetIfxDecimal(2);
        Console.WriteLine("Value Sales Tax :\t "+vtax);
        Console.WriteLine("Ceiling Sales Tax :\t " + IfxDecimal.Ceiling(vtax));
        Console.WriteLine("Negative Sales Tax :\t " + IfxDecimal.Negate(vtax));

        conn.Close();
     }
     catch (Exception e) {
        Console.WriteLine(e.Message);
     }
   }
}
```

Example 8-28 shows the output of the previous example.

Example 8-28 Output of the decimal.cs sample

```
C:\work>csc.exe /R:%INFORMIXDIR%\bin\netf20\IBM.Data.Informix.dll /platform:x86
decimal.cs
```

```
C:\work>decimal
Value Sales Tax :         0.0825
Ceiling Sales Tax :       1.0
Negative Sales Tax :      -0.0825

C:\work>
```

8.3.5 Troubleshooting

In this section, we discuss typical errors that occur when using the .NET providers and how to enable the tracing facility to collect diagnostic information.

Typical errors

If the application fails to connect to the Informix database server, verify that connection string details are correct for the server and version of the .NET provider that you are using.

Both .NET providers use the ODBC or CLI layer to communicate with the database server. You can test the connection with these components before using a .NET provider.

When using Informix .NET Provider, always test the connectivity first using the `iLogin` utility that is included with Client SDK.

Data Server Provider for .NET includes the `Testconn11.exe` and `Testconn20.exe` executables that you can use to diagnose connectivity and setup problems when using Data Server Provider for .NET.

Table 8-19 lists typical errors and how to resolved them.

Table 8-19 Typical errors

Error	Reason	Solution
`ERROR [IM009] [Informix .NET provider] Unable to load translation shared library (DLL)`	The Informix .NET Provider failed to load the ODBC shared library.	Verify that the values for the INFORMIXDIR environment variable and PATH are correct. INFORMIXDIR should point to the Client SDK installation directory, and PATH should contain the `%INFORMIXDIR%\bin` directory.
`System.BadImageFormatException: An attempt was made to load a program with an incorrect format`	A 32-bit application is attempting to load a 64-bit `IBM.Data.Informix` assembly.	Register the proper version of the .NET assembly into the GAC using the `gacutil.exe` tool.

Error	Reason	Solution
`Unhandled Exception: System.EntryPointNotFoundException: Unable to find an entry point named 'InterlockedIncrement' in DLL 'kernel32.dll'.`	A 64-bit application is attempting to load a 32-bit `IBM.Data.Inforimx.dll` assembly.	Register the proper version of the .NET assembly into the GAC using the `gacutil.exe` tool.
`Unhandled Exception: System.DllNotFoundException: Unable to load DLL 'IfxDotNetIntrinsicModule.dll': The specified module could not be found.`	Failed to load `IfxDotNetIntrinsicModule` assembly that is required by the 64-bit version of Informix .NET Provider.	When using the 64-bit version of Informix .NET Provider, the PATH environment variable needs to define the `%INFORMIXDIR%\bin` and `%INFORMIXDIR%\bin\netf20` directories.

8.3.6 Tracing

Your application can set the following types of tracing when using Informix .NET Provider:

- Informix .NET Trace
- SQLIDEBUG

Informix .NET Trace

This trace is specific to Informix .NET Provider. To use this feature, the application must use the Trace version of the `IBM.Data.Informix` assembly located in the `%INFORMIXDIR%\netf20` directory. The name of shared library is `IBM.Data.Trace.dll`.

Register the assembly in the GAC and enable the trace by setting the IFXDOTNETTRACE and IFXDOTNETTRACEFILE environment variables.

Example 8-29 demonstrates how to register the trace library and generate a .NET trace file.

Example 8-29 Trace setup

```
C:\work>gacutil /i %INFORMIXDIR%\bin\netf20\IBM.Data.IfxTrace.dll /nologo
Assembly successfully added to the cache

C:\work>set IFXDOTNETTRACE=2

C:\work>set IFXDOTNETTRACE=trace.txt

C:\work>decimal
Value Sales Tax :       0.0825
```

```
Ceiling Sales Tax :      1.0
Negative Sales Tax :     -0.0825

C:\work>dir trace.txt
 Volume in drive C is W2003
 Volume Serial Number is 50DA-70D7

 Directory of C:\work

25/06/2010  13:52            13,172 trace.txt
               1 File(s)         13,172 bytes
               0 Dir(s)  76,259,311,616 bytes free

C:\work>
```

The .NET trace file contains information about all the .NET classes that are used and their return values. Example 8-30 shows some of the entries in the .NET trace file.

Example 8-30 .NET Trace file

```
...
3532:1  Entry: IfxDecimal.ToString()
3532:1  Exit: IfxDecimal.ToString
IfxDecimal  0.0825
3532:1  Entry: IfxDecimal.ToString()
532:1  Exit: IfxDecimal.ToString
IfxDecimal  -1.0
)
3532:1  Entry: IfxDecimal.get_IsNull()
3532:1  Exit: IfxDecimal.get_IsNull
532:1  Exit: IfxDecimal.Multiply
3532:1  Exit: IfxDecimal.Negate
3532:1  Entry: IfxConnection.Close()
3532:1  Entry: IfxConnection.GetLatch
(
  String  IfxConnection.Close
)
...
```

The level of tracing is determine by the value of the IFXDOTNETTRACE variable:

0	No tracing
1	Tracing of API entry and exit, with return code
2	Tracing of API entry and exit, with return code, plus tracing of parameters to the API

Levels 3 and 4 are for internal use only.

SQLIDEBUG and DRDADEBUG

Refer to “SQLIDEBUG” on page 119 for the description and use of the SQLIDEBUG.

8.4 Visual Studio Add-In for Visual Studio

This section gives an overview of Visual Studio Add-in. The IBM Database Add-Ins for Visual Studio are a collection of features that integrate into Microsoft Visual Studio development environment so that you can work with IBM data servers.

Visual Studio Add-in Version 9.5 is compatible with Visual Studio 2005, and it uses Informix .NET Provider to connect to the database server.

Version 9.7 is compatible with Visual Studio 2005 and Visual Studio 2008 and uses Data Server Provider for .NET.

Although it is not included in the Client SDK bundle, Visual Studio Add-in is available as a separate product. For more information, refer to:

`https://www.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-vsai`

Visual Studio Add-In provides tools and wizards that simplify the .NET development for the following common tasks:

- Explore catalog information of the database server.
- Generate Data Definition Language (DDL) for database objects.
- Drag server objects onto .NET application components to generate the required ADO.NET code automatically.

To use Visual Studio Add-In, you first define the Informix server in the Data Connections group of the Visual Studio environment. Select the Add Connection to add a new server definition.

In the Add Connection panel, specify the server name, server log on ID and password, and the database name.

When the database is connected, you can drag the Informix tables from the Server Explorer pane to the .Net data set pane.

For more details about using IBM Database Add-Ins and Data Server Provider for .NET for application development, refer to the IBM Information Management and Visual Studio .NET zone at:

`http://www.ibm.com/developerworks/data/zones/vstudio`

9 Working with PHP

PHP is a powerful server-side scripting language that was invented and that is designed for creating dynamic web applications with non-static content. The PHP code can be a stand-alone program or an insert inside Hypertext Markup Language (HTML) or Extensible Hypertext Markup Language (XHTML). The PHP syntax is based mostly on and is similar to the C, Java, and Perl programming language. You can use PHP based on an open-source license.

This chapter discusses the use of PHP with Informix as the database server. We discuss various PHP database extensions that you can use to connect to Informix. We also discuss other components, such as Apache Web Server and OpenAdmin Tool (OAT), and how to bring these components together. We describe in detail the use of programming interfaces that are provided by selected PHP database extensions through the use of various examples.

This chapter includes the following topics:

- Informix and PHP extensions
- Setup and configuration
- Developing a PHP application

9.1 Informix and PHP extensions

The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions.

Informix database can be accessed by PHP using the following PDO drivers:

- PDO_INFORMIX

  This driver is also called *PHP Driver for IBM Informix*. To compile and use the driver, you must install Informix Client Software Development Kit (Client SDK). You can download this driver from:

  `http://pecl.php.net/package/PDO_INFORMIX`

- PDO_IBM

  This driver is also know as *PHP Driver for Data Server clients*. To use this driver, you must install IBM Data Server Driver for ODBC and CLI on the same computer. You can download this from:

  `http://pecl.php.net/package/PDO_IBM`

In addition to these two drivers, the following PHP extensions allow you to access an IBM Informix database:

- PHP_INFORMIX

  This extension provides support only for standard Informix data types. This extension is developed by the open source community and is available at:

  `http://cvs.php.net/viewvc.cgi/pecl/informix/`

- IBM_DB2

  This extension is available with IBM Data Server Driver and is written, maintained, and supported by IBM. It supports the Informix and DB2 databases.

Your application development environment, including hardware and software, determines which PHP drivers and extensions to use. If the platform of your choice is not supported by IBM Data Server Driver or if you are using some of the data types that are not supported by IBM Data Server Driver, then use the Informix driver. The Data Server Driver for ODBC and CLI requires an Informix database of Version v11.x or later. If your database server is exclusively Informix, then you can use the Informix driver. You can also use both drivers simultaneously if needed.

9.2 Setup and configuration

To develop a PHP application for Informix, your application development system must have the following software:

- PHP drivers and extensions

  Choose one of the PHP drivers or extensions that Informix supports. The PHP Driver for IBM Informix requires Client SDK and the PHP Driver for Data Server clients requires the IBM Data Server Driver for ODBC and CLI. Refer to 2.2, "Client setup" on page 34 for the details.

- Web server

  You can either install Apache Web Server or use OpenAdmin Tool (OAT), which is a PHP-based web browser administration tool. Installing OAT is an easier option, because it contains everything that is required for PHP connectivity. At the time of writing this book, OAT version 2.2.7 has the following components:

  – Apache 2.2.4
  – PHP 5.2.4
  – PDO_INFORMIX 1.2.6

In this section, we discuss OAT installation and driver configuration. For information about how to install the latest Apache Web Server, refer to the following website:

`http://httpd.apache.org/`

9.2.1 Installing OAT

OpenAdmin Tool (OAT) is available for Windows, Linux, and Macintosh OS X (64-bit only) at:

`http://www.openadmintool.org`

This website leads you to the IBM site, where you must register and log in.

After you complete the download, the installation procedure is as follows:

1. Start the installation with one of the following methods:

   – GUI mode

     Launch the installation executable using one of the following commands:

     - On Windows systems: Run `install.exe`.
     - On Linux: Run `install.bin`.
     - On MAC OS X: Extract the `install.zip` file, and run `install.app`.

   - Console mode

     Enter one of the following commands:

     - On Windows systems: `install.exe -i console`
     - On Linux: `install.bin -i console`

2. Accept the license agreement to continue.
3. Select the installation directory. The default directory is as follows:
   - For Windows systems: `C:\Program Files\OpenAdmin`
   - For Linux: `/opt/OpenAdmin`
   - For MAC OS X: `/Applications/OpenAdmin/`
4. Choose an available port number for the web server.
5. For Windows operating system users, provide an Apache service name.
6. Specify the host name, which is the name of the computer where the database server is located.
7. In the Security Features panel, enable password-protection for OAT administration pages.
8. In the OAT Administrator login setup panel, enter a user name and password.
9. In the Plug-in page, ensure that the plug-ins that you want to install are selected. Accept the license agreement.
10. In the Pre-Installation Summary panel, review your selections and proceed with the installation. When the installation is complete, the following message displays:

    ```
    OpenAdmin Tool has been installed successfully. Please visit
    http://servername:portnumber/openadmin/ to use the OpenAdmin Tool
    ```

    Where:

    - *`servername`* is the name of the system where the web server is running. This name can be `localhost` on Windows.
    - *`portnumber`* is the port number that you provided in step 4.
11. Click **Done**.
    - For Windows systems: You need to reboot. You can access OAT from the **Start ∅ All Programs ∅ OpenAdmin Tool for IDS**.
    - For Linux and MAC OS X: The OAT configuration page opens in your default web browser.

For an insight into OAT, read the release notes and the README file that are bundled with the product.

9.2.2 Verifying the PDO_INFORMIX setup

We discuss the installation of the PDO_INFORMIX driver in 2.2.3, "Setting up IBM Data Server drivers" on page 43. The PDO_INFORMIX driver is installed automatically when you install OAT.

We use the PDO_INFORMIX driver and the Apache Web Server that is installed with OAT in a Windows system for the examples in this chapter.

On successful installation of OAT, the `php_pdo_informix.ddl` shared library (`.so` extension in Linux) is placed in the PHP extension directory. For example, if your installation directory is `C:\Program FIles\OpenAdmin`, the shared library is in the `C:\Program Files\OpenAdmin\PHP_5.2.4\ext` directory.

To verify that the PHP extensions are working, enter the following URL in a web browser:

```
http://localhost:8080/phpinfo.php
```

9.2.3 Verifying the PDO setup

We discuss the installation of the PDO_IBM driver in 2.2.3, "Setting up IBM Data Server drivers" on page 43. Depending on your platform copy, add the `.dll` or `.so` to the OAT extension directory and change the `php.ini` to include the extension `php_pdo_ibm.dll` or `php_pdo_ibm.so`.

You can see which PDO driver is loaded by the Apache Web Server by typing the following address in your browser:

```
http://localhost:8080/phpinfo.php
```

9.2.4 Verifying connectivity

You can use a simple PHP program to verify the connectivity to the Informix server using the installed Informix PHP drivers or extensions.

Example 9-1 shows a simple PHP script that connects to the database server using the PDO_INFORMIX driver. In the `conn_string` variable, set the proper host name, user name, password, and respective locales. The first keyword in the connection string identifies the PDO driver that is used. The PDO_INFORMIX driver uses `informix`, and the PDO_IBM driver uses `ibm`.

Example 9-1 The connect.php program

```
<?php
 $informixdir = getenv("INFORMIXDIR");
 $uname = "informix"; $password= "123456";
 $conn_string = "informix: host=9.12.4.65; service=9088; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp; CLIENT_LOCALE=EN_US.CP1252 ;

 $sql = "SELECT dbinfo('version', 'major') version FROM systables WHERE
tabid=1";
 try {
  print "<li>informixdir = $informixdir</li>\n";
  $conn = new PDO($conn_string, $uname, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  print "<li>Got a connection</li>\n";
  $stmt = $conn->query($sql);
  if ( ! $stmt ) {
    print "Error in execute: stmt->execute()\n";
    print "errInfo[0]=>$err[0]\nerrInfo[1]=>$err[1]\nerrInfo[2]=>$err [2]\n";
  }
  $row = $stmt->fetch();
  print "<li>IDS version " . $row['VERSION'] . "</li>\n";
 }
 catch (Exception $e) {
  print "Exception messsage:{$e->getMessage()}\n";
 exit(0);}
?>
```

If you run the script from a browser, an output similar to the following displays:

```
informixdir = C:\Program Files\IBM\Informix\
Got a connection
IDS version 11
```

You can use the same script to test the connectivity using the PDO_IBM driver by just changing the connection string. Example 9-2 shows a typical connection string using the PD_IBM driver.

Example 9-2 PDO_IBM connection string

```
$conn_string = "ibm: DRIVER={IBM DB2 ODBC
DRIVER};DATABASE=stores_demo;HOSTNAME=kodiak;PORT=9089;PROTOCOL=TCPIP;";
```

At this point, you know that your environment is set up properly, and you are ready to begin using PHP with Informix.

9.3 Developing a PHP application

PHP is a powerful, server-side scripting language that was invented and designed for creating dynamic web applications with non-static content. The PHP code can be a stand-alone program or an insert inside Hypertext Markup Language (HTML) or Extensible Hypertext Markup Language (XHTML). The PHP syntax is based mostly on and similar to C, Java, and Perl. You can use PHP with an open-source license. You can run the PHP program directly from the command line.

We chose PHP for our applications for the following reasons:

- Easy to use

  PHP is a scripting language that is included directly in HTML. Thus, getting started is easy. There is no need to compile PHP programs or spend time learning tools to create PHP. You can simply insert statements and get quick turnaround as you make changes.

- Fully functional

  The PHP language has built-in functions to access your favorite database. With PHP, your HTML pages can reflect current information from databases. You can use information for the user who is viewing your HTML web page to customize the page specifically for that user. You can create classes for object-oriented programming or use flat file or Lightweight Directory Access Protocol (LDAP) databases. It also includes a spelling checker, XML functions, image generation functions, and more.

- Compatible and quick

  Because PHP generates plain HTML, PHP is compatible with all web browsers.

- Secure

  Although PHP is open source, it is a secure environment. Web clients can only see the pure HTML code. The logic of the PHP program is never exposed to the client; therefore, security exposures are reduced.

- Open source

  PHP is an open-source programming language. It is easy to get started and find examples from websites. For example:

  `http://www.sourceforge.net`

9.3.1 Connecting to a database

Regardless of which PHP driver you use, PDO_INFORMIX or PDO_IBM, you must decide the type of connection to use for any database connection and the user type to use for authentication.

Connection type

The following connection type behavior is controlled by the web server:

- Persistent connection

  For the *persistent* connection, the web server leaves the connection to the database open after the PHP script completes its work. The next attempt to connect to the database with the same parameters reuses the connection.

- Standard connection

  For the *standard* connection, the database connection is closed when the script completes its execution.

Here are some considerations for persistent connection:

- When using the persistent connection, the same connection is used for the next request with the same connection parameters. This connection might cause connection pooling, depending on the implementation and how you decide whether the connection is reused. Unintentional connection pooling is most likely a source of serious problems, even if you use transactions.
- Opening up a persistent connection makes sense only in a web server environment if the connection is not closed in the same script with a connection close call. Calling the PHP command line processor on the shell closes the connection to the Informix database server at the end of the execution of the script.

- The web server spawns more than one process to handle the incoming web requests. So, reusing the existing persistent connection is valid only for a specific web server process. It results in the database server having more connections open than actual running web server processes.
- Restarting the database server without cleaning up the remaining persistent connections in the web server environment produces no errors at connection time when the connection is reused but fails at the first database statement. This type of error is hard to diagnose.

The user type

The user type is controlled by the database and has to do with authentication at the database server. You can choose an authentication with user ID and password in the connection string or a trusted user connect where the database server does not apply an authentication. This connection is controlled by settings on operating system resources, such as `.rhosts` or `/etc/hosts.equiv` files. Because you have to specify user names and passwords in the PHP scripts, pay attention to these files in regard to security. Use encapsulation for files that contain passwords, and put these files in directories that are secured with the `.htaccess` or `httpd.conf` mechanism of the Apache Web Server.

Informix PDO requires, at connection time, a defined set of parameters:

- Database name
- Database server name

We suggest that you use a user ID and password for untrusted user sessions. Some optional parameters can influence the cursor behavior. In this section, we focus on the required connection parameters.

Example 9-3 gives an overview about the various parameter settings in an attempt to connect to the Informix database server. The examples shown are for the PDO_INFORMIX driver. Choose one of the connection statements that meets your requirements.

Example 9-3 The PDO_INFORMIX driver connection strings

```
/*--- standard connect ---*/
$dbc = new PDO("informix:; database=sysmaster;
server=ol_svr_custom;","informix", "123456");

/*--- standard connect trusted user ---*/
$dbc = new PDO("informix:; database=sysmaster; server=ol_svr_custom;");

/*--- persistent connect untrusted user ---*/
$dbc = new PDO("informix:; database=sysmaster;
server=ol_svr_custom;","informix","123456",array(PDO::ATTR_PERSISTENT=> true));
```

```
/*--- persistent connect trusted user ---*/
$dbc = new PDO("informix:; database=sysmaster;
server=ol_svr_custom;",NULL,NULL,array(PDO::ATTR_PERSISTENT=> true));
```

The PDO_IBM driver is based on the Data Server Driver CLI driver. Thus, it uses the same connectivity information as the ODBC and CLI driver. You can store this information, as well as user credentials, in the `db2cli.ini` file that is located in the user profile directory. Example 9-4 shows an example `db2cli.ini` file.

Example 9-4 An example db2cli.ini file

```
[dsc_dsn]
Protocol=TCPIP
Port=9089
Hostname=kodiak
Database=stores_demo
PWD=password
UID=password
```

Having the details inside the `db2cli.ini` file makes it easier to manage the information that is needed for the connection. You can use either of the connection strings mentioned in the Example 9-5. The second `conn_string` uses the `dbscli.ini` file shown in Example 9-4 on page 304.

Example 9-5 PDO_IBM connection strings

```
$conn_string = "ibm: DRIVER={IBM DB2 ODBC DRIVER};DATABASE=stores_demo;
HOSTNAME=kodiak;PORT=9089;PROTOCOL=TCPIP;";

$conn_string = "ibm: DSN=dsc_dsn";
```

9.3.2 Performing database operations

Informix supports both static and dynamic SQL statements for the client applications such as PHP programs. In this section, we provide an in-depth discussion about the abilities of handling dynamic and static SQL of PDO Informix. We focus on the INSERT statement but also give short examples for the DELETE and UPDATE statements.

Static SQL statements

Example 9-6 shows various methods to execute a static INSERT with Informix PDO.

Example 9-6 INSERT, UPDATE, and DELETE using the PDO_INFORMIX driver

```
<?php
$informixdir = getenv("INFORMIXDIR");
$dbc = new PDO("informix:; database=stores_demo;
server=ol_svr_custom;","informix", "123456");
$sql = "INSERT INTO customer VALUES(0,'Carla','Gomes','All kitchen supplies',
'2440 Cavaleras blvd', '', 'San Jose','CA','93086', '408-777-8075');";
$dbc->exec($sql);
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" execute insert failed with %s <br>",$error["1"]);
exit(1);
}

/*--- Using PDO::query for the insert ---*/
$dbc->query($sql);
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" execute insert failed with %s <br>",$error["1"]);
exit(1);
}

/*--- Using PDO::prepare and PDO::sql::execute for the insert ---*/
$stmt=$dbc->prepare($sql);
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" prepare insert failed with %s <br>",$error["1"]);
exit(1);
}
$stmt->execute();
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" execute insert failed with %s <br>",$error["1"]);
exit(1);
}
printf("RowCount: %d <br>",$stmt->rowCount());
$stmt=$dbc->query("SELECT COUNT(*) FROM customer ");
$row=$stmt->fetch(PDO::FETCH_NUM);
printf("<br>Total Rows in Table %d <br>", $row[0]);
?>
```

In a web application, the input data frequently comes from an HTML form that is entered by a web user. Example 9-7 shows, based on a simple HTML form, how the data flows from the HTML over the PHP script into the database table.

Example 9-7 HTML sample

```
basic html format
<?php
printf("<form method=\"post\" action=\"<your file name>.php\" >");
printf("fname <input name=\"fname\" /><br />");
printf("lname <input name=\"lname\" /><br />");
printf("company <input name=\"company\" /><br />");
printf("address1 <input name=\"address1\" /><br />");
printf("address2 <input name=\"address2\" /><br />");
printf("city <input name=\"city\" /><br />");
printf("state <input name=\"state\" /><br />");
printf("zipcode <input name=\"zipcode\" /><br />");
printf("phone <input name=\"phone\" /><br />");
printf("<input type=\"submit\" value=\"Confirm\" name=\"Button\"/>");
printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
printf("</form>");
?>
```

Example 9-8 shows the PHP script that handles the HTML form. The `$_POST` array contains all the settings for the buttons and the values for the text fields from the HTML.

Example 9-8 PHP Script handling the insert

```
<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456");
if (!dbc) {
exit();
}
/* --data from the Input Format
0|Carla|Gomes|Red Socks|217 Milpitas|Alum Rock|San Jose|CA|408-988-9887|
---*/
$statement="INSERT INTO customer VALUES (0,".
"'" . $_POST["fname"] . "'," .
"'" . $_POST["lname"] . "'," .
"'" . $_POST["company"] . "'," .
"'" . $_POST["address1"] . "'," .
"'" . $_POST["address2"] . "'," .
"'" . $_POST["city"] . "'," .
"'" . $_POST["state"] . "'," .
"'" . $_POST["zipcode"] . "'," .
"'" . $_POST["phone"] . "');";
printf ("STATEMENT = %s", $statement);
```

```
$dbc->query($statement);
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" prepare insert failed with %s \n",$error["1"]);
exit(1);
}
?>
```

Example 9-9 shows how to perform an update and a delete using the PDO Informix extension.

Example 9-9 Update a PHP script

```
<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456");
if (!dbc) {
exit();
}
/*--- Static update that changes Redwood city to Fremont ---*/
$statement="UPDATE customer SET city='Fremont' where city='Redwood City'";
$stmt=$dbc->query($statement);
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" update failed with %s <br>",$error["1"]);
}
printf("#Records updated: %d <br>",$stmt->rowCount());

/*--- Delete all records beloning to the company Sportstown; ---*/
$dbc->beginTransaction();
$statement="DELETE FROM customer WHERE company='Sportstown'";
$stmt=$dbc->prepare($statement);
$stmt->execute();
printf("#Records deleted: %d <br>",$stmt->rowCount());
if ($stmt->rowCount()>10000)
{
$dbc->rollback();
}
else {
$dbc->commit();
}
?>
```

Dynamic SQL statements

Dynamic SQL statements allow an Informix client side program to build an SQL statement at run time, so that the content of the statement can be determined by user input. Informix PDO provides several methods to use dynamic SQL. You must use placeholders, such as “?” or “`:parameter`” to prepare a statement for dynamic usage. These placeholders are later substituted with the values.

The “prepare once, execute multiple times” capability of dynamic statements allows you to specify different values at each execution time. To determine which statements and clauses allow placeholders, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.sqls.doc/sqls.htm

This document also provides a comprehensive reference of Informix SQL statements.

Example 9-10 shows different INSERT statements using Informix PDO. To learn more about using placeholders (“?”) and bind parameters, refer to PHP manual at:

http://php.net/manual/en/pdostatement.bindparam.php

Example 9-10 INSERT statements using Informix PDO

```
<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo; server=ol_svr_custom;
protocol=onsoctcp ", "informix", "123456");
if (!dbc) {
exit();
}

/*--- dynamic inserts with "?" and bindParam ---*/
$stmt=
$dbc->prepare("INSERT INTO customer VALUES (0,?,?,?,?,?,?,?,?,?)");
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" prepare insert1 failed with %s \n",$error["1"]);
exit(1);
}
$stmt->bindParam(1, $_POST["fname"]);
$stmt->bindParam(2, $_POST["lname"]);
$stmt->bindParam(3, $_POST["company"]);
$stmt->bindParam(4, $_POST["address1"]);
$stmt->bindParam(5, $_POST["address2"]);
$stmt->bindParam(6, $_POST["city"]);
$stmt->bindParam(7, $_POST["state"]);
$stmt->bindParam(8, $_POST["zipcode"]);
$stmt->bindParam(9, $_POST["phone"]);
$stmt->execute();
$error=$dbc->errorInfo();
if ( $error["1"]) {
```

```
printf(" prepare insert2 failed with %s \n",$error["1"]);
exit(1);
}

/*--- dynamic inserts with :<> placeholders and bindParam ---*/
$stmt=
$dbc->prepare("INSERT INTO customer VALUES (0,:p1,:p2,:p3,:p4,:p5,:p6,:p7,:p8,:p9)");
$stmt->bindParam(':p1', $_POST["fname"]);
$stmt->bindParam(':p2', $_POST["lname"]);
$stmt->bindParam(':p3', $_POST["company"]);
$stmt->bindParam(':p4', $_POST["address1"]);
$stmt->bindParam(':p5', $_POST["address2"]);
$stmt->bindParam(':p6', $_POST["city"]);
$stmt->bindParam(':p7', $_POST["state"]);
$stmt->bindParam(':p8', $_POST["zipcode"]);
$stmt->bindParam(':p9', $_POST["phone"]);
$stmt->execute();
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" prepare insert2 failed with %s \n",$error["1"]);
exit(1);
}

/*--- dynamic inserts with "?" placeholders and bindValue ---*/
$stmt=
$dbc->prepare("INSERT INTO customer VALUES (99,?,?,?,?,?,?,?,?,?)");
$stmt->bindValue(1, $_POST["fname"],PDO::PARAM_STR);
$stmt->bindValue(2, $_POST["lname"],PDO::PARAM_STR);
$stmt->bindValue(3, $_POST["company"],PDO::PARAM_STR);
$stmt->bindValue(4, $_POST["address1"],PDO::PARAM_STR);
$stmt->bindValue(5, $_POST["address2"],PDO::PARAM_STR);
$stmt->bindValue(6, $_POST["city"],PDO::PARAM_STR);
$stmt->bindValue(7, $_POST["state"],PDO::PARAM_STR);
$stmt->bindValue(8, $_POST["zipcode"],PDO::PARAM_STR);
$stmt->bindValue(9, $_POST["phone"],PDO::PARAM_STR);
$stmt->execute();
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" prepare insert3 failed with %s \n",$error["1"]);
exit(1);
}

/*--- dynamic inserts with :<> placeholders and bindValue ---*/
$stmt=
$dbc->prepare("INSERT INTO customer VALUES
(199,:p1,:p2,:p3,:p4,:p5,:p6,:p7,:p8,:p9)");
$stmt->bindValue(':p1', $_POST["fname"],PDO::PARAM_STR);
$stmt->bindValue(':p2', $_POST["lname"],PDO::PARAM_STR);
$stmt->bindValue(':p3', $_POST["company"],PDO::PARAM_STR);
$stmt->bindValue(':p4', $_POST["address1"],PDO::PARAM_STR);
$stmt->bindValue(':p5', $_POST["address2"],PDO::PARAM_STR);
$stmt->bindValue(':p6', $_POST["city"],PDO::PARAM_STR);
$stmt->bindValue(':p7', $_POST["state"],PDO::PARAM_STR);
$stmt->bindValue(':p8', $_POST["zipcode"],PDO::PARAM_STR);
```

```
$stmt->bindValue(':p9', $_POST["phone"],PDO::PARAM_STR);
$stmt->execute();
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" prepare4 insert failed with %s \n",$error["1"]);
exit(1);
}
?>1
```

9.3.3 Handling complex data types

In this section, we discuss how to use complex data types in a PHP program. We cover row types, collection types (such as SET, LIST, and MULTISET), and the BLOB and SBLOB data types. We explain how to work with the complex data types using examples with Informix PDO.

Named row types

Example 9-11 shows the Data Definition Language (DDL) to create a ROW type and the table that uses the ROW type.

Example 9-11 DDL for creating ROW type and table

```
/*Create a row type and a table by name customer_rtype as below

CREATE ROW TYPE address_rtype (
street_num  int,
street_name char(20),
city        char(20),
state       char(20),
zipcode     char(10)
);

CREATE TABLE customer_rtype (
customer_num  serial,
lname         char(15),
fname         char(15),
company       char(20),
address       address_rtype,
phone         char(18)
);
```

The PHP script in Example 9-12 shows SQL statements using ROW type. This script inserts two rows and selects the data from the database with and without a filter on the row type. It then updates one of the rows and delete the rows from the table.

Example 9-12 Insert complex types sample

```
<?php

$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo; server=ol_svr_custom;
protocol=onsoctcp ", "informix", "123456");
if (!dbc) {
exit();
}
printf(" <br>[ INSERT ] <br><br>");

/*--- Insert a row with a rowtype -- add a row to the newly created table ---*/
$dbc->query(' INSERT INTO customer_rtype VALUES (
0,"Carla","Gomes","All Sports",row(12345,"Broadway","San
Fransisco","CA","12345")::address_rtype, "408-908-8887")');
$dbc->query(' INSERT INTO customer_rtype VALUES (
0,"Smith","jones","Collin sports",row(1222,"Almeda
blvd","Fremont","CA","12345")::address_rtype,"345-908-8887")');
printf(" <br>[ SELECT without filter] <br><br>");

/*--- Select the customer row without any filters ---*/
$stmt1=$dbc->query(" SELECT customer_num,fname,address.city,address.state,phone from
customer_rtype");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
print_r($row);
printf("<br>");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
}
printf(" <br>[ SELECT with row_type filter ] <br><br>");

/*--- Select the row with a row_type filter ( a given address ) ---*/
$stmt1=$dbc->query(' SELECT customer_num , address.state, phone FROM customer_rtype
WHERE row(1222,"Almeda blvd","Fremont","CA","12345")::address_rtype = address');
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
print_r($row);
printf("<br>");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
}
printf(" <br>[ UPDATE ] <br><br>");

/*--- update customer address ---*/
$stmt1=$dbc->query(' UPDATE customer_rtype SET
address=row(1234,"Almeda blvd","Fremont","CA","12345")::address_rtype WHERE
row(1222,"Almeda blvd","Fremont","CA","12345")::address_rtype = address');
printf(" <br>[ SELECT ] <br><br>");
```

```
/*--- Verification ---*/
$stmt1=$dbc->query(' SELECT customer_num,fname, address::lvarchar FROM customer_rtype
WHERE address = row(1234,"Almeda blvd","Fremont","CA","12345")::address_rtype');
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
print_r($row);
printf("<br>");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
}
printf(" <br>[ DELETE ] <br><br>");

/*--- close the buisness ---*/
$dbc->query(' DELETE FROM address_rowtype ');
$dbc->query(' DELETE FROM customer_rtype ');
?>
```

Example 9-13 shows the output from the SELECT statements to give you an idea of how the array with row types looks in PHP.

Example 9-13 Output of ROW type example

```
[ INSERT ]


[ SELECT without row_type filter ]

Array ( [CUSTOMER_NUM] => 1 [FNAME] => Gomes [CITY] => San Fransisco [STATE] => CA
[PHONE] => 408-908-8887 )
Array ( [CUSTOMER_NUM] => 2 [FNAME] => jones [CITY] => Fremont [STATE] => CA [PHONE] =>
345-908-8887 )

[ SELECT with filter ]

Array ( [CUSTOMER_NUM] => 2 [STATE] => CA [PHONE] => 345-908-8887 )

[ UPDATE ]


[ SELECT ]

Array ( [CUSTOMER_NUM] => 2 [FNAME] => jones [] => ROW(1234 ,'Almeda blvd ','Fremont
','CA ','12345 ') )

[ DELETE ]
```

Collection data types

The collection data types are another set of complex data types provided by the Informix. Collection data types include SET, LIST, and MULTISET. In this section, we cover these collection data types. Example 9-14 shows an example that inserts into and selects from a table containing collection data types.

Example 9-14 Using collection data types in PHP

```
<?php
/*create the tables containing collection data types
CREATE TABLE t_collections
(
 seq     serial not null,
 l1 list    (integer not null),
 s1 set     (integer not null),
 m1 multiset(integer not null)
);
*/

$dbc = new PDO("informix:host=9.14.23.34;  database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456",
array(PDO::ATTR_PERSISTENT=> true));
if (!dbc) {
exit();
}
/*--- Insert a row ---*/
printf(" <br>[ Insert ] <br><br>");
$dbc->query(" INSERT INTO t_collections  VALUES ( 0,
'LIST{-1,0,-2,3,0,0,32767,249}', 'SET{-1,0,-2,3}', 'MULTISET{-1,0,0,-2,3,0}')
");
$dbc->query(" INSERT INTO t_collections  VALUES ( 0,
'LIST{-1,0,-2,3,0,0,55555,249}', 'SET{-11,0,-2,3}',
'MULTISET{-1,0,0,-2,3,0,9,10}') ");

/*--- select the row without any filters ---*/
printf(" <br>[ SELECT without filter] <br><br>");
$stmt1=$dbc->query(" SELECT l1::lvarchar LIST, s1::lvarchar SET, m1::lvarchar
MULTISET FROM t_collections");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
print_r($row);
printf("<br>");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
}

/*--- select the row with a set filter for the color of the car ---*/
printf("<br> [ SELECT with filter] <br><br>");
$stmt1=$dbc->query(" SELECT l1::lvarchar LIST, s1::lvarchar SET, m1::lvarchar
MULTISET FROM t_collections WHERE 32767 IN l1");
```

```
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
print_r($row);
printf("<br>");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
}
?>
```

Example 9-15 shows the output of Example 9-14 on page 313.

Example 9-15 The output of collection data type

```
[ Insert ]


[ SELECT without filter]

Array ( [LIST] => LIST{-1 ,0 ,-2 ,3 ,0 ,0 ,32767 ,249 } [SET] => SET{-1 ,0 ,-2
,3 } [MULTISET] => MULTISET{-1 ,0 ,0 ,-2 ,3 ,0 } )
Array ( [LIST] => LIST{-1 ,0 ,-2 ,3 ,0 ,0 ,55555 ,249 } [SET] => SET{-11 ,0 ,-2
,3 } [MULTISET] => MULTISET{-1 ,0 ,0 ,-2 ,3 ,0 ,9 ,10 } )

[ SELECT with filter]

Array ( [LIST] => LIST{-1 ,0 ,-2 ,3 ,0 ,0 ,32767 ,249 } [SET] => SET{-1 ,0 ,-2
,3 } [MULTISET] => MULTISET{-1 ,0 ,0 ,-2 ,3 ,0 } )
```

BLOB and SBLOB data types

With BLOB and SBLOB data types, Informix IDS can handle large objects as binary data in a BYTE data type and text objects in a TEXT data type. The BYTE and TEXT data types, commonly known as *BLOB* data types or *simple large objects*, provides the capability to store images and entire documents in the database. BLOB data types can be stored within all the other data in the table space or in a separate specified BLOB space.

BLOB data types can be used in several operational areas. Commonly, document retrieval systems and geographic information systems are based on this data type. To retrieve BLOB data with SQL, you need to define keywords, which are stored together with the BLOB in the data row. The benefit of storing large data in the database is, in addition to an easy search using keywords and a combination of data stored in different rows but belonging together, the opportunity to backup and restore or delete and update for specific data. Storing the data in a file in the operating system is much more of a maintenance effort.

More advanced than the BLOB data types, smart large objects (commonly known as SBLOB), provide more flexibility for searching data, such as random I/O access to the data, which was impossible with simple large objects).

There are two types of SBLOBs, BLOB, and CLOB. The BLOB data type is used to stored binary data when the CLOB data type is used only for character data. There types require an additional set of functions defined in the server that provides an API for access. They are stored in an SBLOB space (smart BLOB space) in the database server.

You must create an sbspace to store the BLOB types in the server. You can create them with following command:

```
onspaces -c -S sbspace -p <PATH> -o 0 -s 4000
```

We change the `catalog` table in the `stores` database to use BLOB and CLOB instead of BYTE and TEXT using the following command:.

```
ALTER TABLE catalog MODIFY (cat_descr TEXT, cat_descr CLOB, cat_picture BYTE,
cat_picture BLOB)
```

Example 9-16 shows how to insert and retrieve from data with SBLOB data types, that is BLOB and CLOB. The example shows how to insert the large data from files into a table and how to select the same data back into files in a local file system.

Example 9-16 Smart large object sample

```
<<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456");
if (!dbc) { exit(); }

/*--- try to insert the BLOB ---*/
$stmt= $dbc->prepare("INSERT INTO catalog VALUES (0,302,'KAR',?,?,'All sports
Goods')");
$file = fopen ("picture.jpg","r");
$image = fread ( $file, 100000) ;
fclose ( $file);
$file = fopen ("README.txt","r");
$text = fread ( $file, 100000) ;
fclose ( $file);
$stmt->bindParam(1, $text );
$stmt->bindParam(2, $image);
$stmt->execute();
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" execute insert blobs failed with %s \n",$error["1"]);
exit(1);
}

/*--- Get the BLOB from the database back ---*/
$query=$dbc->query("SELECT * FROM catalog where stock_num=302 and
manu_code='KAR' ");
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" select blobs failed with %s \n",$error["1"]);
exit(1);
}
$count=1;
$row=$query->fetch(PDO::FETCH_ASSOC);
while ( $row ) {
$file = fopen ("README.$count.txt","w");
$test=fread($row["CAT_DESCR"],100000);
while($test) {
fwrite ( $file, $test) ;
$test=fread($row["CAT_DESCR"],100000);
}
fclose ( $file);
$file = fopen ("PICTURE.$count.jpg","w");
$test1=fread($row["CAT_PICTURE"],100000);
while($test1) {
fwrite ( $file, $test1) ;
$test1=fread($row["CAT_PICTURE"],100000);
}
fclose ( $file);
$count++;
$row=$query->fetch(PDO::FETCH_ASSOC);
}
?>
```

Using TEXT and BYTE (BLOB data types) is similar to SBLOB types. Users can try Example 9-16 on page 315 and Example 9-17 using TEXT and BYTE instead of CLOB and BLOB respectively.

The BLOB data to be inserted can be read from the file without placing it in a variable first, which simplifies the code as shown in Example 9-17.

Example 9-17 Insert BLOB with file

```
<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456");
if (!dbc) { exit(); }

/*--- try to insert the BLOB ---*/
```

```
$stmt= $dbc->prepare("INSERT INTO catalog VALUES (0,302,'KAR',?,?,'All sports
Goods')");
$file = fopen ("C:\README.txt","r");
$file1 = fopen ("C:\picture.jpg","r");
$stmt->bindParam(1,$file, PDO::PARAM_LOB);
$stmt->bindParam(2,$file1, PDO::PARAM_LOB);
$stmt->execute();
fclose($file);
fclose($file1);
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" execute insert blobs failed with %s \n",$error["1"]);
exit(1);
}

/*---try to get the BLOB from the database back ---*/
$query=$dbc->query("SELECT * FROM catalog where stock_num=302 and
manu_code='KAR' ");
$error=$dbc->errorInfo();
if ( $error["1"]) {
printf(" select blobs failed with %s \n",$error["1"]);
exit(1);
}
$count=1;
$row=$query->fetch(PDO::FETCH_ASSOC);
while ( $row ) {
$file = fopen ("C:\README.$count.txt","w");
$test=fread($row["CAT_DESCR"],100000);
while($test) {
fwrite ( $file, $test) ;
$test=fread($row["CAT_DESCR"],100000);
}
fclose ( $file);
$file = fopen ("C:\PICTURE.$count.jpg","w");
$test1=fread($row["CAT_PICTURE"],100000);
while($test1) {
fwrite ( $file, $test1) ;
$test1=fread($row["CAT_PICTURE"],100000);
}
fclose ( $file);
$count++;
$row=$query->fetch(PDO::FETCH_ASSOC);
}
?>
```

In Example 9-17 on page 316, the data was selected as a stream in string data type. Example 9-18 shows how to bind variables to a SELECT. The data is placed in the variables by column.

Example 9-18 Select BLOB with bind

```
<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456");
if (!dbc) { exit(); }

/*--- try to select the BLOB from the database into bind strings ---*/
$stmt=$dbc->query("SELECT catalog_num, cat_descr, cat_picture FROM catalog
where stock_num=302 and manu_code='KAR'");
$count=1;
$str="";
$str1="";
$id=0;
$stmt->bindColumn(1, $id, PDO::PARAM_INT);
$stmt->bindColumn(2, $str, PDO::PARAM_STR,100000);
$stmt->bindColumn(3, $str1, PDO::PARAM_STR,100000);
while ( $stmt->fetch(PDO::FETCH_BOUND) )
{
$file = fopen ("C:\README.$count.txt","w");
$file1 = fopen ("C:\PICTURE.$count.jpg","w");
fwrite($file,$str);
fwrite($file1,$str1);
fclose ($file);
fclose ($file1);
$count++;
}
?>
```

Example 9-19 shows how to update existing BLOB fields with Informix PDO using a prepare statement and parameter for the BLOB columns.

Example 9-19 BLOB update

```
<?php
$dbc = new PDO("informix:host=9.14.23.34; database=stores_demo;
server=ol_svr_custom; protocol=onsoctcp ", "informix", "123456");
if (!dbc) { exit(); }
/*
try to update the BLOB columns
*/
$stmt= $dbc->prepare("UPDATE catalog SET cat_descr=? , cat_picture=? where
stock_num=302 and manu_code='KAR'");
$error=$dbc->errorInfo();
if ( $error["1"]) {
```

```
printf(" prepare update blob columns failed with %s \n",$error["1"]);
exit(1);
}
$descr="This is an PDO descr clob text";
$file= fopen ("C:\picture1.jpg","r");
$stmt->bindParam(1, $descr);
$stmt->bindParam(2, $file);
$stmt->execute();
$error=$dbc->errorInfo();
?>
```

9.3.4 Working with PHP extensions

In addition to the PDO drivers for the PHP PDO extension, there are two PHP extensions that allow you to connect to an Informix database server:

- PHP_INFORMIX
- IBM_DB2

These extensions provide a set of additional PHP functions to work with an Informix database. In addition to the normal create, read, update, and write database operations, they also offer extensive access to the database metadata.

Table 9-1 lists a few of the functions that are included in the Informix PHP extensions as examples.

Table 9-1 PHP extension functions

Function	Description
`ifx_connect()`	Opens Informix Server connection
`ifx_fieldproperties()`	Lists of SQL field properties
`ifxus_open_slob()`	Opens an SLOB object
`ifx_create_blob()`	Creates an BLOB object
`ifx_fetch_row()`	Gets row as an associative array
`ifx_query()`	Sends Informix query
`db2_connect()`	Returns a connection to a database
`db2_client_info()`	Returns information describing DB2 database client
`db2_primary_keys()`	Returns a result set listing primary keys for a table
`db2_special_columns()`	Returns a result set listing the unique row identifier columns for a table

Function	Description
`db2_bind_param()`	Binds a PHP variable to an SQL statement parameter
`db2_commit()`	Commits a transaction

For a complete list of all the functions that are implemented in the PHP_INFORMIX and IBM_DB2 extensions, refer to:

`http://www.php.net/manual/en/ref.ifx.php`
`http://www.php.net/manual/en/ref.ibm-db2.php`

The PHP_INFORMIX extension

A PHP script can connect to an Informix database using the `ifx_connect()` function that is provided in the PHP_INFORMIX extension.

Example 9-20 shows a simple PHP script that tests connectivity with the PHP_INFORMIX extension.

Example 9-20 The connect.php script

```
C:\work>type connect.php
<?php
$conn= ifx_connect ($argv[1], $argv[2],$argv[3]);
echo "Connection succeeded.\n";
ifx_close($conn);
?>
C:\work>php connect.php stores_demo informix password
Connection succeeded.

C:\work>
```

You can use functions such as `ifx_query()` and `ifx_affected_rows()` to run SQL statements and to retrieve the number of rows affected. Example 9-21 demonstrates how to delete a row from the `state` table using the `ifx_query()` function.

Example 9-21 The delete.php script

```
C:\work>type delete.php
<?php
$conn= ifx_connect ("stores_demo", "informix", "password");
$result = ifx_query("DELETE FROM state where code='".$argv[1]."'", $conn);
printf("Deleted %d records", ifx_affected_rows($result));
ifx_close($conn);
?>

C:\work>php delete.php UK
```

```
Deleted 1 records

C:\work>
```

The application can select data from an Informix database using the `ifx_prepare()` and `ifx_fetch_row()` functions.

Example 9-22 shows how to select the first three rows from the `state` table using a prepared statement and the `ifx_fetch_row()` function.

Example 9-22 The select.php script

```
C:\work>cat select.php
<?php
$conn= ifx_connect ("stores_demo", "informix", "password");

$rid = ifx_prepare ("SELECT FIRST 3 code,sname FROM state",$conn, IFX_SCROLL);
if (! ifx_do ($rid)) {
    die ("error\n");
}
$row = ifx_fetch_row ($rid, "NEXT");
while (is_array($row)) {
    for (reset($row); $fieldname=key($row); next($row)) {
        $fieldvalue = $row[$fieldname];
        printf ("%s = %s ", $fieldname, $fieldvalue);
    }
    printf("\n");
    $row = ifx_fetch_row($rid, "NEXT");
}
ifx_free_result ($rid);
ifx_close($conn);
?>
C:\work>php select.php
code = AK sname = Alaska
code = HI sname = Hawaii
code = CA sname = California

C:\work>
```

For more information and examples regarding the PHP_INFORMIX extension, refer to:

http://www.php.net/manual/en/book.ifx.php

The IBM_DB2 extension

This PHP extension provides access to IBM Data Servers, including IBM Informix and IBM DB2.

In the same way as the PDO driver, PDO_IBM, the IBM_DB2 extension requires the IBM CLI driver to communicate with the database server. The IBM CLI driver is included as part of the IBM Data Server Driver for ODBC and CLI package.

This PHP extension provides functions such as `db2_connect()`, `db2_exec()`, and `db2_server_info()` that you can use to perform typical tasks against an Informix database server.

Example 9-23 demonstrates how to use the `db2_connect()` and `db2_server_info()` functions to retrieve metadata information from the database.

Example 9-23 The connect_ibm.php script

```
C:\work>type "C:\Documents and Settings\Administrator\db2cli.ini"

[dsc_dsn]
Protocol=TCPIP
Port=9089
Hostname=kodiak
Database=stores_demo

C:\work>cat connect_ibm.php
<?php
$conn = db2_connect('dsc_dsn', 'informix','password');

if ($conn) {
    echo "Connection succeeded.\n";
    $server = db2_server_info( $conn );
    printf ("Database name     = %s\n", $server->DBMS_NAME);
    printf ("Datbase version   = %s\n", $server->DBMS_VER);
    db2_close($conn);
}
else {
    echo "Connection failed.";
}
?>
C:\work>php connect_ibm.php
Connection succeeded.
Database name     = IDS/NT64
Datbase version   = 11.50.0000

C:\work>
```

The IBM_DB2 extension can use the connection details in the `db2cli.ini` configuration file. Refer to the 3.2.2, "IBM Data Server Driver for ODBC and CLI" on page 70 for more information regarding the `db2cli.ini` configuration file.

You can use several functions to select data from the database. Example 9-24 shows how to use the `db2_fecth_object()` function to retrieve the first three rows of the `state` table as a PHP object.

Example 9-24 The select_ibm.php script

```
C:\work>cat select_ibm.php
<?php
$conn = db2_connect('dsc_dsn', 'informix','password');

if ($conn) {
    echo "Connection succeeded.\n";

    $stmt = db2_exec($conn, "SELECT FIRST 3 code,sname FROM state");
    while ($row = db2_fetch_object($stmt)) {
        printf ("%s, %s\n", $row->code,$row->sname);
    }
    db2_close($conn);
}
else {
    echo "Connection failed.";
}
?>
C:\work>php select_ibm.php
Connection succeeded.
AK, Alaska
HI, Hawaii
CA, California

C:\work>
```

You can find a full description of the IBM_DB2 extension at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.apdv.php.doc/doc/t0023132.htm

9.3.5 Exception handling

Error handling or exception handling is important for all applications, including web-based applications. Consider an Informix PHP application that shows the following error message while the user is using the application:

```
Warning: odbc_connect(): SQL error: [unixODBC][Informix][Informix
ODBCDriver][Informix]User (informix) password “123456” is not able to
connect for the database server, Server is down, SQL state 28000 in
SQLConnect in /usr/local/apache2/htdocs/odbc/error/error.php on line 4
```

In this case, fail in exception handling creates a security exposure. In the this section, we describe many aspects related to exception handling using Informix PDO.

PHP 5 has introduced, as a part of the new object-oriented programming interface, the exception handling mechanism that is already used for other programming languages. We strongly suggest that you consider exceptions for the usage of Informix PDO. Additionally, you can advance this interface by creating, throwing, and catching your own exceptions.

In comparison with the procedural-oriented interface, Informix PDO defines the following methods of expressing an error in a database environment:

- Exceptions raised internally by the PDO based on an error condition
- Database-generated errors, which can be captured and handled by calling the PDO class `errorInfo()` or `errorCode()` function

Example 9-25 shows two exceptions raised by Informix PDO functions. One exception is a connection request to the database that failed because the specified database does not exist. The other exception is an attempt to start a transaction twice.

Example 9-25 Without exception handling sample

```
<?php
$dbc = new PDO("informix:; database=stores; server=ol_svr_custom;","informix",
"123456");
?>

Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE=HY000,
SQLDriverConnect: -329 [Informix][Informix ODBC Driver][Informix]Database not
found or no system permission.' in C:\Program
Files\OpenAdmin\Apache_2.2.4\htdocs\excep1.php:2 Stack trace: #0 C:\Program
Files\OpenAdmin\Apache_2.2.4\htdocs\excep1.php(2): PDO->__construct('informix:;
data...', 'informix', '123456') #1 {main} thrown in C:\Program
Files\OpenAdmin\Apache_2.2.4\htdocs\excep1.php  on line 2
```

```
<?php
$dbc = new PDO("informix:; database=stores_demo;
server=ol_svr_custom;","informix", "123456");
$dbc->beginTransaction();
$dbc->beginTransaction();
?>
Fatal error: Uncaught exception 'PDOException' with message 'There is already
an active transaction' in C:\Program
Files\OpenAdmin\Apache_2.2.4\htdocs\excep1.php:4 Stack trace: #0 C:\Program
Files\OpenAdmin\Apache_2.2.4\htdocs\excep1.php(4): PDO->beginTransaction() #1
{main} thrown in C:\Program Files\OpenAdmin\Apache_2.2.4\htdocs\excep1.php  on
line 4
```

If these exceptions are not caught and processed, the application terminates. Example 9-26 shows the use of the basic exception handler provided by PHP 5 to cover these errors. After an exception is caught, the action that is taken depends on where the error originates. For example, if the error occurs in connecting to the database phase and the application cannot continue, the action is to generate an “out of order” web page with contact details. If the error is a minor database error, logging the error and retrying the activity is an appropriate action.

Example 9-26 Simple exception handle code

```
<?php
try
{
$dbc = new PDO("informix:; database=stores_demo;
server=ol_svr_custom;","informix", "123456");
$dbc->beginTransaction();
$dbc->beginTransaction();
}
catch (PDOException $e )
{
printf("Error: %s \n",$e->getMessage());
}
?>

Output:
Error: There is already an active transaction.
```

The error information that is generated when executing the SQL statements in the database server is different from the exceptions that are generated by the Informix PDO extension. You can use the Informix PDO `errorInfo()` function to capture the status of the last executed SQL statement in the database. This function returns an array with the following elements:

- The SQLSTATE
- The SQLCODE
- The error message

For the details about the meaning of the codes, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.esqlc.doc/sii-11-40709.htm#sii-11-40709

The `errorCode()` function is available to retrieve the SQL statement status. This function returns only the SQLSTATE information. Example 9-27 shows how to use the `errorInfo()` function and its output.

Example 9-27 ErrorInfo() sample

```
<?php
$dbc = new PDO("informix:; database=stores_demo;
server=ol_svr_custom;","informix", "123456");
$stmt=$dbc->query('SELECT * FROM nonexistingtable ');
/*
question the error code
output of the error Routines
*/

$error=$dbc->errorInfo();
print_r($error);
if (!$error[1])
$row=$stmt->fetch(PDO::FETCH_ASSOC);
?>

Output:
Array
(
[0] => 42S02
[1] => -206
[2] => [Informix][Informix ODBC Driver][Informix]The specified table
(nonexistingtable) is not in the database. (SQLPrepare[-206] at at
ext\pdo_informix\informix_driver.c:118)
)
```

In addition to using the generic exceptions provided by the PHP 5, you can extend the exception class of your own exceptions. For example, you can define different severities for SQL errors. Critical database errors are, for example tables that do not exist or the database connections, cannot be established. The application cannot continue with these critical errors. Non-critical errors, such as locking errors, can be handled by a retry.

Example 9-28 demonstrates how to define your own exceptions with PHP 5 and Informix PDO. This example extends the standard exception class with two new database exception classes and, depending on the severity, different actions are taken.

Example 9-28 Custom exceptions

```
<?php
/*
own Exception classes for minor and major errors
*/
class CriticalDatabaseErrors extends Exception
{
public function __construct($message, $code = 0) {
parent::__construct($message, $code);
}
}
class NonCriticalDatabaseErrors extends Exception
{
public function __construct($message, $code = 0) {
parent::__construct($message, $code);
}
}
try
{
$dbc =  new PDO("informix:host=9.14.23.34; database=stores_demo; server=ol_svr_custom;
protocol=onsoctcp ", "informix", "123456");
$stmt=$dbc->exec("SET ISOLATION REPEATABLE READ");
$stmt=$dbc->query('SELECT * FROM carr');
$error=$dbc->errorInfo();
if (!$error[1]) {
do
{
$row=$stmt->fetch(PDO::FETCH_ASSOC);
$error=$dbc->errorInfo();
if ($error[1]) throw new NonCriticalDatabaseErrors($error[1]);
}
while($row) ;
}
else {
throw new CriticalDatabaseErrors($error[2]);
}
}
catch ( CriticalDatabaseErrors $cde )
{
printf("<pre>CritError!: %s \n</pre>",$cde->getMessage()) ;
}
catch ( NonCriticalDatabaseErrors $ncde )
{
```

```
printf("<pre>NonCritError!: %s \n</pre>",$ncde->getMessage()) ;
} catch (PDOExecption $ncde) {
printf("<pre>Error!: %s \n</pre>",$e->getMessage()) ;
exit;
}
?>
Output:
CritError!: [Informix][Informix ODBC Driver][Informix]The specified table (carr) is not
in the database. (SQLPrepare[-206] at ext\pdo_informix\informix_driver.c:131)
```

9.3.6 Troubleshooting

The following common errors can occur in setting the database connectivity:

- Missing environment variable setting

  Setting the environment variable INFORMIXDIR is required for starting the Apache. If this variable is not set properly, the database connection also fails.

  Example 9-29 shows the error message when you have improper INFORMIXDIR set. The message occurs at the first line of any PHP program which is generally the connection string. If you get this message, verify that the INFORMIXDIR setting is correct.

  Example 9-29 Unspecified error

  ```
  Error!: SQLSTATE=HY000, SQLDriverConnect: -23101
  [Informix][Informix ODBC Driver][Informix]Unspecified System Error =
  -23101.
  ```

- Mismatched settings

  Another important item affecting the connectivity is the setup of the Informix runtime environment. Informix Connect or Client SDK provides the connectivity at run time. If the settings between the environment variables and the `sqlhosts` do not match, you receive messages similar to the one shown in Example 9-30.

  Example 9-30 Wrong connection information

  ```
  /usr/local/bin/php pdoconnect.php
  Error!: SQLSTATE=HY000, SQLDriverConnect: -25555
  [Informix] [Informix ODBC Driver][Informix]Server ol_svr_custom is not
  listed as a dbserver name in sqlhosts.
  ```

10

User-defined routines

In this chapter, we how to create and use user-defined routines (UDRs). A UDR is a routine that you create that can be invoked in an SQL statement by the database server or from another UDR. A *routine* is a collection of program statements that perform a particular task. By understanding UDRs, you can take the next step and extend the database server, either a little bit or in steps that lead to something bigger, such as a bladelet or DataBlade module.

This chapter includes the following topics:

- An overview of UDRs and database extensions
- Developing UDRs
- DataBlades and bladelets

Extending the database server requires an understanding of the components that are required to implement the extension. An *extension* can be as simple as one UDR or as complex as a new data type with many supporting routines. We first discuss UDRs and user-defined types (UDTs) to provide the scope for what is needed. Then we provide examples.

10.1 An overview of UDRs and database extensions

IBM Informix database servers have the built-in ability to retrieve, store, manipulate, and sort a number of standard data types, some of which are unique to Informix servers. A developer can choose to modify or extend the built-in data types and to modify or extend how various operations work with the resulting new data type extensions.

You can extend the data types using the following methods:

- Extend operations that are used to process built-in data types.
- Create complex data types based on built-in data types.
- Create UDTs, both distinct and opaque data types.
- Create new operations to process extended data types.

Database extensions and extended data types allow the developer to make customized routines and functions transparent, because you have the capability to actually build it into the database server. The transparency is visible as:

- Better performance through optimized routines, faster queries, and reduced network traffic.
- Simpler applications in streamlined code and easy to upgrade applications.
- Transaction control with automatic recovery, backup, and rollback capability that is provided by the server.
- Scalability because database extensions scale automatically with the database size and user count.

In the context of discussing UDRs and database extensions, we also mention DataBlade modules. A *DataBlade module* is a software package that extends the functionality of the IBM Informix database server. Each package includes SQL statements and supporting code written in an external language or in Stored Procedure Language. A DataBlade module enables the same level of support for new data types as the database server provides for built-in data types. DataBlade modules can also use SQL queries or the DataBlade API to access data types and routines in other DataBlade modules.

Note: Extended Parallel Server and Standard Engine support stored procedures but not UDRs.

10.1.1 Considerations for UDRs

A UDR can be written using the IBM Informix Stored Procedure Language (SPL) or with an external language. A routine written with SPL is simple to use and easy to implement. An SPL-based UDR has flow-control extensions (conditional clauses and while loops) and works with SQL statements. After the routine is created and is ready to use, the database server parses, optimizes, and stores it in system catalog tables so that it is ready to execute.

The system catalog tables are used to keep track of the information the database server uses to manage the database server. UDR-related information is stored in a small group of catalog tables that provide information about the UDR layouts. UDRs include the following common catalog tables:

- sysprocedures table

  Used to track the name and owner and to indicate whether the UDR is a user-defined function or a user-defined procedure (functions return values but procedures do not).
- sysprocauth table

  Tracks who can execute the procedure.
- sysprocbody table

  Contains the actual code for the SPL routines.

UDRs can also be written with an external language. The body of an external-language routine allows language-based operations such as flow control and looping, while also using special Informix library calls to access the database server.

The database server stores information for external-language routines in the following system catalog tables:

- sysprocedures table

  The information kept in this table is same as the SPL UDR.
- sysroutinelangs table

  This table tracks the language information.
- syslangauth table

  This table tracks the users of the server who can use the particular external language.

You need to use an appropriate compiler to parse and compile an external-language routine into an executable format. We discuss this with specific examples of the language and API methods that are available for writing these extensions.

The external languages that can be used are:

- C

  To execute SQL statements in C UDRs, you must use the DataBlade API, and you cannot use ESQL/C. To write routines in C, you need a C compiler. YOu can find additional information about writing UDRs in C in the *IBM Informix DataBlade API Programmer's Guide, Version 11.50,* SC23-9429, and *IBM Informix DataBlade API Function Reference, Version 11.50,* SC23-9428.

- Java

  To write Java routines, you must use IBM Informix database server with J/Foundation and with the Java Development Kit (JDK) to compile your Java routines. You can find additional information about how to write Java UDRs in *J/Foundation Developer's Guide, Version 10.0,* G251-2291.

10.1.2 About UDRs

Table 10-1 lists the UDR types that are supported by SPL, C, and Java in IBM Informix database server.

Table 10-1 Supported UDR tasks by languages

UDR task	SPL routines	C routines	Java routines
Cast function	Yes	Yes	Yes
Cost function	No	Yes	No
End-user routine	Yes	Yes	Yes
Iterator function	No	Yes	Yes
Negator function	Yes	Yes	Yes
Opaque data type support	No	Yes	Yes
Parallelizable UDR	No	Yes	Yes
Statistics function	No	Yes	Yes
Selectivity function	No	Yes	No
User-defined aggregate	Yes	Yes	Some
Operator function	Yes	Yes	Yes
Operator-Class function	Yes	Yes	Yes

The following terms are used in this table:

- *Cast*: A routine to convert from one data type to another. Built-in data types have automatic casts between most data types. For UDTs, casting must be defined from scratch.
- *Cost function*: A routine that informs the optimizer of cost factors for execution of a particular UDR.
- *End-user function*: A routine that is used to encapsulate multiple SQL statements into one function.
- *Iterator function*: A routine that is designed to return to its calling SQL statement several times, returning a value each time.
- *Negator function*: A routine used for a `NOT` condition that involves a Boolean UDR.
- *Opaque data type support*: When you create a new data type, you must provide several basic support functions for your UDT. The following functions are required:
  - Text input and output routines
  - Binary send and receive routines
  - Text import and export routines
  - Binary import and export routines
- *Parallel UDR*: A routine that can run in parallel within parallel queries.
- *Statistics function*: A routine to create distribution statistics for a UDT.
- *Selectivity function*: A routine to determine the percentage of rows for which a Boolean UDR is expected to return true.
- *User-defined aggregate*: A SQL invoked routine that takes values selected and returns information about those rows (a summarizing method).
- *Operator function*: A routine that is used within expressions with a symbol, such as “+,-,<,>,=”. Built-in data type operators cannot be extended. All UDTs require some operators to function within an SQL context.
- *Operator-Class function*: A set of operators that the server associates with how to build an access method (that is, an index), how to arrange values in the access method, how to select values based on operator function, and ways to allow the query optimizer to consider using the access method to return results for a query.

For more details about any of these functions and for details about functionality that we do not discuss here, see *IBM Informix User-Defined Routines and Data Types Developer's Guide, Version 11.50,* SC23-9438.

Invoking a UDR

You can invoke a UDR implicitly or explicitly. *Implicit invocation* is the result of an operator function, an implicit cast, or some type of query processing. For this book, we mainly discuss explicit invocation. You can use either EXECUTE PROCEDURE or EXECUTE FUNCTION statements to run a UDR.

When the database server executes an SQL statement that contains a UDR, it loads the UDR executable code as a shared-object into memory. It determines which shared-object file to load from the `externalname` column of the row in the `sysprocedures` system catalog table that describes the UDR. The `sysprocedures` entry is created when you register the UDR, as a result of the CREATE FUNCTION or CREATE PROCEDURE statement.

In more general terms, when you invoke a UDR, the database server parses the statement into syntactic parts, call the system catalog to resolve the routine parts, generate a query plan, and then execute the query. If the query involves more than one database, each database requires that all the UDRs and UDTs must be registered in the participating databases.

10.1.3 Considerations for extending data types

A significant aspect of UDR is the support for an extended data type. IBM Informix database servers have several built-in data types. Why do we need more?

An *extended data type* is a new data type that has different core properties, new functionality, and new operator methods that go beyond a basic data type.

As an example, consider a `datetime` data type. Imagine, as a programmer if you had to add “a day and a half” to a specific datetime event. Intuitively, we recognize that the value “‘day and a half” is an interval of time, but the database serve does not recognize this construct as an interval. We need to convert it manually to a usable interval, and then add it to a datetime value.

With a UDR, an unconventional interval name such as this can be passed through a function, which translates it to an “normal” interval automatically, applies the addition operation, and returns a `datetime` value with the result. To do this, we apply a functional behavior change, which extends the data type. The result of processes such as this can simplify a repeating data task and make the development task easier.

A quick review of the data type hierarchy is useful. Figure 10-1 shows a summary of these data types.

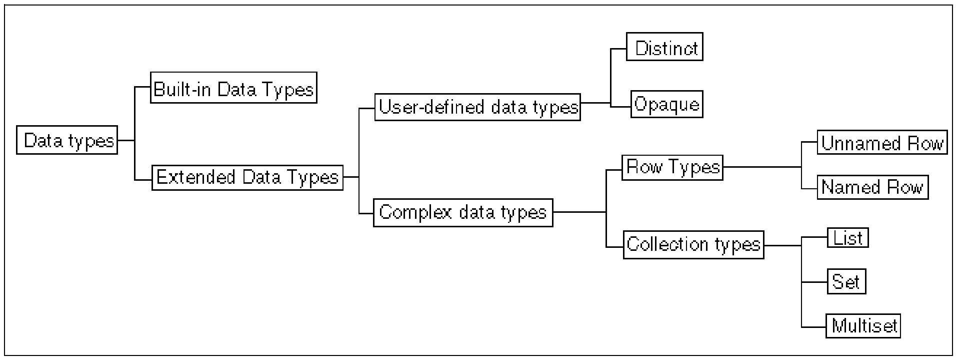


Figure 10-1 Hierarchy of data types in Informix servers

The extended data types can be either UDR types or complex data types. We are especially interested in UDTs, which can be described as follows:

- *Distinct*: Internally, this data type is stored the same as a source data type, but it is overlaid with different casts and functions defined beyond the basic source type. The server sees distinct types as different from the source type. It is necessary to tell the server:
  - Source data type information and how the internal structure is defined.
  - The functions of how this data type interacts with its internal structure.
  - The operations that are valid with this distinct data type.
  - Any secondary access methods on how to handle this type.
  - Cast functions to move data in and out of the distinct type are automatic.
- *Opaque*: A fundamental, user-defined data type. It cannot be broken into smaller pieces, although it can serve as the building block for other data types. The internal structure of the opaque data type is invisible to the server. When you define and use an opaque type, the developer must provide all of the following:
  - How the internal structure is defined.
  - The functions that enable routines to interact with its internal structure.
  - The operations that are valid with this distinct data type.
  - Any secondary access methods on how to handle this type.
  - Cast functions to move data in and out of the distinct type need to be provided.

10.2 Developing UDRs

To develop a UDR, plan it first and then write the routine. We follow the recommendations given in the *UDR and Data Types Developers Guide* for planning our routines:

- Use a sensible name.
- Avoid modal arguments (an argument in the function determines how the function will work).
- Always declare routine parameter data types.
- Declare the type that is returned when the routine is returning a value

The source for an external routine resides in a separate text file. To prepare UDR source code:

- You will compile the C-language UDR and store the executable version in a shared-object (`.so` or `.o` on UNIX) file.
- Compile the Java-language UDR and store the executable version in a `.jar` file.

You must install shared object files and `.jar` files on all database servers that need to run the UDRs, including database servers involved in Enterprise Replication (ER) and High-Availability Data Replication (HDR). The shared object files and `.jar` files need to be installed under the same absolute path name.

Tip: Although not required, use the DataBlade Developer's Kit (DBDK) to help write UDRs is advantageous. DBDK books and software can help enforce standards that facilitate migration between different versions of the database server. Because external-language routines are external to the database, be aware that you must compile the UDR source code and store it where the database server can access it when the routine is invoked.

For information about C UDRs, refer to *IBM Informix DataBlade API Programmer's Guide, Version 11.50*, SC23-9429 and *IBM Informix DataBlade API Function Reference, Version 11.50*, SC23-9428. For information about Java UDRs, refer to the *J/Foundation Developer's Guide, Version 10.0,* G251-2291.

10.2.1 UDR examples in SQL

You can use Informix Stored Procedure Language (SPL) statements to write routines and then store these SPL routines in the database. SPL extends SQL and helps to reduce SQL coding by minimizing the visible code in large SQL operations. It has the advantage that it is run as a server-side routine, the executable code stays inside the engine, and it is optimized only as needed. The end result is lower application startup costs and better performance. As an SQL extension, SPL can do flow control, such as looping and branching. SPL routines can also execute routines written in C or other external languages, and other UDR routines can execute SPL routines.

In this section, we provide examples of UDRs written in SQL. As we do so, consider the following rules for SQL UDRs:

- If you use any parameters, they must be declared as built-in or user-defined data types.
- An SPL routine does not have access to the user state of its execution sequence. If the routine is going to be called more than once and if you want to retain information about previous executions in the transaction, use an SPL routine that states WITH RESUME as a part of the RETURN statement for multiple executions of the same SPL routine within the same routine sequence.
- When an SPL routine is executed, the parameters (also known as the *dependency list*) is checked. If it is determined that an item in the dependency list needs reoptimization, optimization occurs at this point. If an item needed in the execution of the SQL statement is missing (for example, a column or table is dropped), an error is returned.
- UDRs can be *overloaded*, meaning that a function can have more than one way to operate, depending on the list of data types that are provided as parameters. There is a precedence hierarchy to decide how the parameter list is executed. A precedence hierarchy to decide the execution sequence of the parameters can be important if you have more than one UDR with the same name but a different parameter list. For more information, see *IBM Informix User-Defined Routines and Data Types Developer's Guide, Version 11.50,* SC23-9438.

Using a stored procedure method for a UDR is simple, as long as you can recognize the incoming and outgoing parameters properly. The incoming parameters, which are provided in the function definition, must have data types defined when the parameter is first created and declared. The parameter values, which are to be returned from the function, also should have declared data types. By following this general rule, you can avoid many of the initial problems you can get with SPL UDRs.

For the routines with SPL, we use the Informix `stores` sample database, defined as `stores@demo_on`.

One variable in, one result out

For this example, we want a function that provides a count of all the orders that are received in a numerical month (N) from the orders table of the stores database. This example demonstrates what happens when selecting an aggregating value, once.

Example 10-1 shows the function code and ways to invoke the function to get output.

Example 10-1 CREATE FUNCTION new_orders (month_num INT)

```
CREATE FUNCTION new_orders ( month_num INT )
RETURNING  INT ;
DEFINE nrows INT;
SELECT COUNT(order_date) INTO nrows FROM orders
WHERE month(order_date)=month_num;
RETURN nrows;
END FUNCTION;

# -- Execute it as a function:
EXECUTE FUNCTION new_orders(5);

(expression)
7

# -- Execute it as a select statement.
# -- Note that we have to force “first 1” as a criteria so we can assure we are
# -- only getting one value. Otherwise we would get an error.

SELECT FIRST 1 new_orders(5) FROM orders;

(expression)
7
```

One variable in, several results out

Example 10-2 performs three SELECT statements, and each statement does an aggregation, returning one value from each select. It returns three values and labels each value in the returned result. This example shows how to aggregate three separately selected value returns in one call and still not use a cursor.

Example 10-2 Selecting three and returning three

```
CREATE FUNCTION new_orders ( month_num INT )
 RETURNING  INT as TotalOrders,INT as ShippedOrders,INT as Backorders;
DEFINE nrows INT;
DEFINE mrows INT;
DEFINE b_rows INT;
```

```
SELECT COUNT(ship_date) INTO mrows FROM orders
   WHERE month(ship_date)=month_num;
SELECT COUNT(order_date) INTO nrows FROM orders
   WHERE MONTH(order_date)=month_num;
SELECT COUNT(order_date) INTO b_rows FROM orders
   WHERE MONTH(order_date)=month_num AND ship_date IS NULL;

 RETURN nrows,mrows,b_rows;
END FUNCTION;

# -- Since we are returning three values with labels, calling the execute
# -- function is the most appropriate way to make our SQL call.

EXECUTE FUNCTION new_orders(5);

totalorders shippedorders  backorders

          7             3           1
```

Using the WITH RESUME clause to return a cursor result

Example 10-3 collects better details for the new orders. To return a detailed listing of orders made, it uses the WITH RESUME clause in the function. In this case, we assume (and expect) more than one row is returned. So, the WITH RESUME clause is needed.

Example 10-3 A WITH RESUME ROUTINE that uses a cursor

```
CREATE FUNCTION new_orders ( month_num INT )
 RETURNING  INT as Num, INT as ord_num;
DEFINE  ord_num  INT;
DEFINE Num INT;
LET Num=1;
FOREACH cursor1 FOR
  SELECT order_num INTO  ord_num FROM orders
     WHERE month(order_date)=month_num
  RETURN Num, ord_num WITH RESUME;
  LET Num=Num+1;
 END FOREACH;
END FUNCTION;

EXECUTE FUNCTION new_orders(5);

------num     ord_num-----

         1        1001
         2        1002
         3        1003
         4        1004
         5        1005
         6        1006
         7        1007
```

Multi-table select using a cursor

Example 10-4 performs a multi-table join with a summary expression return and a group by. We ask for all the orders placed in a specific month, the name of the person placing the order, and the total amount for each order.

Example 10-4 A multi-table select using a cursor

```
CREATE FUNCTION new_orders ( month_num INT )
 RETURNING  INT as Ord_Num, char(15) as ord_fname,
char(15) as ord_lname, money(8,2) as Amt;
DEFINE  ord_num  INT;
DEFINE ord_fname char(15);
DEFINE  ord_lname char(15);
DEFINE Amt  money(8,2);
FOREACH cursor1 FOR
 SELECT o.order_num,  c.fname,c.lname,sum(i.quantity*i.total_price)
 INTO  ord_num, ord_fname, ord_lname, Amt
FROM orders o, customer c, items i
WHERE month(order_date)=month_num
AND o.customer_num=c.customer_num
AND i.order_num=o.order_num
group by o.order_num,c.fname,c.lname
  RETURN ord_num, ord_fname, ord_lname, Amt WITH RESUME;
 END FOREACH;
END FUNCTION;
```

10.2.2 UDRs in Java

You must distinguish between JDBC and Java Virtual Machine (JVM) applications. You can use JDBC to write stand-alone applications. If you want to connect with to databases that support Java, you typically write stand-alone JDBC applications, because these applications require specific driver methods to communicate with other database servers. When you write a Java UDR, you must use the IBM Informix JDBC Driver, which is based on the JDBC 2.0 API. The generated code is processed by a JVM that runs as a process inside IBM Informix Server. The generated code (`.jar` file) is stored in an sbspace and can also refer to a `.jar` file in a storage location outside of the server.

Java allows you to create UDRs, cast support functions, aggregates, and opaque data type support routines. However, Java routines cannot handle row or collection data types.

For ordinary UDR with IBM Informix database servers, you can use the Java Developers Kit (version 1.5 at this writing). A JVM comes pre-installed with the IBM Informix database server (with J/Foundation). To confirm it is present, make sure that you have an existing directory path to the `INFORMIXDIR/extend/kraratoa` directory.

Java-based UDR, when developed, is placed into a Java archive (`.jar`) file. The `.jar` file is stored inside an sbspace, or it can have additional supporting files on the file system. If the `.jar` file is large or perhaps proprietary, you can leave it on the file system. Store smaller `.jar` files or files that you might want to update frequently in the sbspace.

Configuration

Make sure that you use the JVM system that came with your server engine, and make sure the `onconfig` file and environment variables are set to accurate working paths. The environment settings for our testing setup had the following variables in the environment:

```
JRE_HOME=/usr/lib/jvm/java/jre
JAVA_BINDIR=/usr/lib/jvm/java/bin
JAVA_HOME=/usr/lib/jvm/java
SDK_HOME=/usr/lib/jvm/java
JJDK_HOME=/usr/lib/jvm/java
JAVA_ROOT=/usr/lib/jvm/java
```

The `JAVA_ROOT` environment variable is mostly determined by the developer. It is the path to the file system directory where you are developing your `.jar` files:

```
CLASSPATH=location_for_your_java_classes:.:.:.
```

The `INFORMIXDIR/etc/ONCONFIG` file also has a small group of parameters that must have verified settings. The following parameters are standard:

```
JVPJAVAHOME     $INFORMIXDIR/extend/krakatoa/jre
JVPHOME         $INFORMIXDIR/extend/krakatoa
JVPPROPFILE     $INFORMIXDIR/extend/krakatoa/.jvpprops
JVPJAVALIB      /bin
JVPJAVAVM       jvm
```

The `JVPJAVALIB` and `JVPJAVAVM` parameters in the `onconfig` file are important for development. The following parameter specifies the name of the log file to which Java problems are written (make sure the path exists):

```
JVPLOGFILE      $INFORMIXDIR/jvp.log
```

If you are going to use external `.jar` files, you must add them to `JVPCLASSPATH`:

```
JVPCLASSPATH  $INFORMIXDIR/extend/krakatoa/krakatoa.jar:$INFORMIXDIR/extend/krakatoa/jdbc.jar
```

Note: The paths for `.jar` files that are added to JVPCLASSPATH are visible only after you add them to the `onconfig` file and restart the database engine.

Routine examples in Java

In the examples that follow, we create our Java routines in `/work/`, which defines the UNIX directory that we use for our working CLASSPATH. You must change the directory path in some of the code expressions if you use a different directory. After you compile the code, the source code is not used for execution, but keep it in a safe place in case you want to improve it later.

A function extension

This example illustrates a simple way to extend functionality. This routine provides an SQL function to "multiply a value times 10." The developing procedure is as follows:

1. Create a simple class file named `Times.java` in the working JVPCLASSPATH directory. Example 10-5 shows the source code.

   Example 10-5 A Java function to multiply a value times 10

   ```
   /*Times.java */
   public class Times {
     public static int TimesTen(int x) {
       return x * 10;
     }
   }
   ```

2. At a command prompt, compile the file using the following command:

   ```
   javac Times.java
   ```

3. Compress it into a `.jar` file:

   ```
   jar cvf Times.jar Times.class
   ```

   As this runs, it shows output similar to the following:

   ```
   added manifest
   adding: Times.class(in = 245) (out= 187)(deflated 23%)
   ```

4. Register the routine:

   Start `dbaccess` to connect to the database and run the following SQL to register and add the `.jar` file into our sbspace:

   ```
   execute procedure sqlj.install_jar ("file://work/Times.jar" , "Times_jar");
   Routine executed.
   ```

5. With the `.jar` file in storage and accessible, from `dbaccess`, we create a function that calls our routine in the `.jar` file. See Example 10-6.

   Example 10-6 Calling routine

   ```
   create function times_ten(value int) returning int
     with (class = "jvp")
     external name "Times_jar:Times.TimesTen"
     language JAVA;
   ```

6. The procedure is ready to run. Test it in `dbaccess`:

   ```
   EXECUTE function times_ten(13)

   (expression)
   130
   ```

   For an alternate method, try:

   ```
   SELECT ship_charge, times_ten(ship_charge) X10 FROM orders;
   ```

A Java UDR with two input parameters

This example demonstrates how to create a Java routine with multiple input parameters. Java language and IBM Informix servers express decimals in different ways. This example creates a new sales tax function, `Salestax`, that includes a new Java function and the existing `Times_ten` function created in “A function extension” on page 342.

We need to include an external standard Java library that has a math class. You might find that multiplying decimal values in Java is different than what you might be used to. In addition, Java uses a different naming convention than Informix data types. If you get the class wrong or the library is incorrect, things simply do not work, and you will get errors.

We use the following procedure to create the `salestax` function:

1. Create a Java class called `Tax.java`, using an editor such as **`vi`**. Example 10-7 shows the source code.

   Example 10-7 The Tax.java class

   ```
   /*Tax.java*/
   import java.math.BigDecimal;
   public class Tax {
   public static BigDecimal salestax(BigDecimal x,BigDecimal xtax )
   {
       BigDecimal ratePlusOne = xtax.add(BigDecimal.valueOf(1));
       BigDecimal afterTax = x.multiply(ratePlusOne);
       afterTax = afterTax.setScale(2, BigDecimal.ROUND_HALF_UP);
     return (afterTax);
     }
   }
   ```

2. Compile the file. At a command prompt, run the following command:

   ```
   javac Tax.java
   ```

3. Compress the file, and add it to the existing `.jar` file (named `Times.jar`).

   ```
   jar cvf Times.jar Tax.class
   ```

   As this runs, it shows a line similar to the following:

   ```
   added manifest
   adding: Tax.class(in = 546) (out= 318)(deflated 41%)
   ```

4. Add the updated `.jar` file to sbspace.

   For this example, we want to update an existing function that already exists in the sbspace `.jar` file. Because we cannot replace an existing function in a sbspace `.jar` file directly, update an sbspace `.jar` function, we must replace the entire `.jar` file. To do this, we must remove (drop) the sbspace `.jar` file, then replace it with an updated version from our working path.

   A `.jar` file in an sbspace must be empty to drop the `.jar` file. Drop all the UDRs in the `.jar` file to empty it. Otherwise, you receive the following error message:

   "`Invalid jar removal. All dependent UDRs not dropped`".

   Our `Times_jar` file exists in the database. To drop the function, we start `dbaccess`, connect to the database, then run the following command:

   ```
   DROP FUNCTION times_ten;
   ```

   Now, we can remove the `.jar` file:

   ```
   EXECUTE PROCEDURE sqlj.remove_jar ("Times_jar");
   ```

5. Add the updated `.jar` file into sbspace. Run the following SQL:

   ```
   EXECUTE PROCEDURE sqlj.install_jar ("file://work/Times.jar" , "Times_jar");
   ```

6. The updated `.jar` file is now in sbspace storage. Re-create the dropped `Times_ten` function:

   ```
   create function times_ten(value int) returning int
     with (class = "jvp")
     external name "Times_jar:Times.TimesTen"
     language JAVA;
   ```

7. Add the new function to call on the same `.jar` file:

   ```
   create function salestax(value decimal,xtax decimal)
     returning decimal(8,2)
     with (class = "jvp")
     external name "Times_jar:Tax.salestax"
     language JAVA;
   ```

8. Now, can test the expanded function from dbaccess:

   ```
   EXECUTE FUNCTION salestax(250.00,.065);

   (expression)
   266.25
   ```

   As an alternative test using SQL, try:

   ```
   SELECT o.order_num,
          salestax (sum(i.quantity*i.total_price), .065) Amt_w_Tax
   FROM orders o,  items i WHERE month(order_date)=5
   AND i.order_num=o.order_num group by o.order_num
   ```

Creating a routine that uses external Java APIs

This example demonstrates how to create a Java UDR that requires support from one or more Java APIs that lie outside the database server. The import references in the code indicate that an external Java API exists to help support the UDR. The references on the import list remain outside the database server. The JVM runs using the `.jar` file calls in the sbspace storage location and sends calls to the external Java functions based on the import reference in the `.jar` file.

Note that the import reference does not have a full directory path. Any Java API or `.jar` file that is not residing in an sbspace is external to the instance. To help the server instance find external `.jar` files, you must supply all JAR API path locations in the `onconfig` file.

Note: The `onconfig` file must be updated so the JVM knows the directory path for any supporting APIs. The full path location for the supporting Java file specified on an import list must be included in JVPCLASSPATH.

For our example, JVPCLASSPATH is set to:

```
/usr3/11.50/extend/krakatoa/krakatoa_g.jar:/usr3/11.50/extend/krakatoa/jdbc_g.j
ar:/usr3/11.50/extend/krakatoa/jre/lib/rt.jar:/work/mailapi.jar:/work/activatio
n.jar:/work/smtp.jar
```

For our example to work, the engine needs the Java mail API classes. The files that are required are `mailapi.jar`, `activation.jar`, and `smtp.jar`, which are available at:

http://java.sun.com/products/javamail/downloads/index.html

To create the `sendmail` routine:

1. Install the downloaded `.jar` files, and add `.jar` files with full path to JVPCLASSPATH.

   In this example, the paths are `/work/mailapi.jar`, `/work/activation.jar`, and `/work/smtp.jar`.

2. Create the file `MailClient.java` as shown in Example 10-8. You have to update the ***italicized*** references in the example with your own information.

Example 10-8 An SQL based sendmail() UDR

```
---- MailClient.java ---
import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;
import java.util.Properties;
public class MailClient
 {
     public static void send(String to, String text)
     {
         try
         {
             MailClient client = new MailClient();

             Properties props = System.getProperties();
             props.put("mail.smtp.host", "smtp.server.com");

             Session session = Session.getDefaultInstance(props, null);

             Message message = new MimeMessage(session);
             message.setFrom(new InternetAddress("Name_showing@fromfield"));
message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
             message.setSubject("Message from the database");
             message.setText(text);

             Transport.send(message);
         }
         catch(Exception e)
         {
             e.printStackTrace(System.out);
```

```
            }
        }
    }
---- MailClient.java ---
```

3. Compile and compress your file:

   ```
   javac MailClient.java
   jar cvf MailClient.jar MailClient.class
   ```

4. Install the `.jar` file into the sbspace:

   ```
   EXECUTE PROCEDURE sqlj.install_jar ("file://work/MailClient.jar" ,
   "MailClient_jar");
   ```

   Make sure JVPCLASSPATH is set as indicated in step 1.

5. Create the `sendmail` procedure from dbaccess:

   ```
   CREATE PROCEDURE sendemail(to LVARCHAR, message LVARCHAR)
     WITH (class = "jvp")
     EXTERNAL NAME "MailClient_jar:MailClient.send"
     LANGUAGE JAVA;
   ```

6. Test the procedure:

   ```
   EXECUTE PROCEDURE sendemail('dba@mybiz.com','Error deleting from table');
   ```

Troubleshooting tips

Sometimes you might have trouble getting a Java UDR to run. The following points of exposure for errors are possible:

- At the time of a Java compile

  If you get an error here, the issue relates to a Java language problem, most likely resulting from syntax or a Java method. Consult a Java Programming Language Guide for assistance.

- At the `.jar` installation point or later

  Check the `jvp.log` at the path in the `onconfig` file specified by the JVPLOG parameter. You should see no errors at the time of install, and no errors at run time. When you start the server instance the JVM starts, `.jar` files load into the process memory as needed. The MSGPATH file (`online.log`) often reveals the success or failure of JVM and `.jar` file loading. If the files cannot load, they cannot run.

- UDR runtime errors

  If there are errors, study the Java error messages returned, and determine the cause for whatever did not resolve.

Troubleshooting can be a trial and error approach. When you have an understanding of how the provided UDR examples work, do them over again, and break something in the Java code. Go through the example with slightly broken syntax and review the results. For example, in the BigDecimal code example, change BigDecimal to Float or Double, and work through the example again.

10.2.3 UDRs in C

The C programming language allows you to create UDRs, cast support functions, aggregates, and opaque data type support routines. C can also handle row and collection data types. Working with C UDR adds an extra layer of difficulty because these routines have to be compiled using a compiler, specific to the system and operating system where you have products installed. Compiled C is not stored in sbspace, but rather it is stored outside the server.

In addition to compiler difficulties, it is a good idea to develop and test UDRs on a development server and not in a production environment. C UDR code runs as a database server process that works closely with internal structures. The routine should not do anything that would negatively affect the database server. A poorly designed C UDR is likely to crash the server. If you plan to work with C UDRs, the following resources are good reference books:

- *IBM Informix User-Defined Routines and Data Types Developer's Guide, Version 11.50*, SC23-9438
- *IBM Informix DataBlade API Programmer's Guide, Version 11.50*, SC23-9429
- *IBM Informix DataBlade API Function Reference, Version 11.50,* SC23-9428

If you are interested in using C++ Datablade modules, the IBM Informix Developer Zone that provides the latest recommendations on C++ programming options at:

http://www.ibm.com/software/data/developer/informix

Routine examples in C

In this section, we undertake a few examples written in C. With Java, we have portability that extends across platforms, but some of our functionality is limited. C is flexible because it can extend the database by way of data types, new functions, and new operators. For each database across an enterprise, the extensibility that you provide through C must be specifically compiled with each operating system. For the examples in this section, we use a Solaris system.

A simple function using C

For our first example, we start with a function similar to what we did in our Java example, a multiplying function. For each of our C examples, we must include the DataBlade API files (referenced in the example as `mi.h` and `milib.h`). The

DataBlade APIs provide the interface to allow C language calls to interface with IBM Informix database SQL calls. To prevent a conflict with our `times_ten` Java UDR function, we name this example `times_five`.

To create this routine:

1. Create the text file that holds the C code. Example 10-9 shows the source file `times.c`.

   Example 10-9 A C UDR that multiplies times five

   ```
   #include <stdlib.h>
   #include "mi.h"
   #include "milib.h"

   mi_integer* TimesFive(mi_integer value);

   mi_integer* TimesFive(mi_integer value)
   {
      return (mi_integer *) (value * 5);
   }
   ```

2. Compile and link the C code.

   The compile command (**`cc`** or **`gcc`**) depends on the operating system. The first way to identify your compiler is to examine the text of `man cc` (on UNIX), or check the command for your operating system compiler in the `INFORMIXDIR/release/en_us/0333/ids_machine_notes_vers.txt` file.

   At a minimum, your compile line for UDR preparation usually includes:

   ```
   -DMI_SERVBUILD -KPIC -I$INFORMIXDIR/incl/public -I/$INFORMIXDIR/incl
   ```

   Where

   – `-DMI_SERVBUILD` is the flag to indicate that this is a server-oriented C UDR application which uses the DataBlade API (required).

   – `-KPIC` is the flag to indicate that the symbol table is dynamic (for UNIX and Linux only).

   – `-I$INFORMIXDIR/incl/public` and `-I/$INFORMIXDIR/incl` are the location of the `mi_ libs`.

   The compile command for our code sample (on Solaris) is:

   ```
   cc  -DMI_SERVBUILD -KPIC -I$INFORMIXDIR/incl/public -I/$INFORMIXDIR/incl -o
   times.o -c times.c
   ```

   Run the link command to put the compiled object file into our Blade library file:

   ```
   ld -dy -G -Bsymbolic -o times.bld times.o
   ```

3. Set the `bld` file permissions as user `informix`:

   ```
   chmod 555 ./times.bld
   ```

4. In dbaccess, after connecting to the database, create the `times_five` function:

   ```
   CREATE FUNCTION times_five(value int) RETURNING int
     WITH (handlesnulls)
     EXTERNAL NAME "/work/times.bld(TimesFive)"
     LANGUAGE C;
   ```

   Adjust the external work path as needed.

5. Test the routine:

   ```
   execute function times_five(20);

   (expression)
   100
   ```

In Java UDR, the `.jar` file serves as a library (collection-repository) for all of the compiled routines. In C, a collection of compiled routines is stored in a shared library (`.so` or `.o`) file. On Windows, a shared object file has a `.dll` extension (dynamic link library).

Creating a C routine using large object column

The next example explores the use of working with a large object column reference. Without a text search DataBlade, searching a character large object (CLOB) file for a particular value can be a laborious SQL task. This example shows how to access a CLOB file, copy the CLOB contents into an LVARCHAR, and search for a specific text item, while using a simple function. We create a C file that handles two parameters. The first parameter is a CLOB column reference, the second parameter is the LVARCHAR text value for our search. It returns a count for the number of successful finds.

To implement this search routine:

1. Create the CLOB column search routine source file `un.c` as shown in Example 10-10.

   Example 10-10 A UDR for searching a CLOB

   ```
   #include <ifxgls.h>
   #include <mi.h>
   #include <milib.h>
   #include <fcntl.h>
   #include <stdio.h>
   #include <stdlib.h>

   typedef unsigned char byte;
   mi_integer contains(MI_LO_HANDLE* loptr, mi_lvarchar* pattern, MI_FPARAM* fp);
   ```

```
mi_integer contains(MI_LO_HANDLE* loptr, mi_lvarchar* value, MI_FPARAM* fp)
{
mi_unsigned_integer crc = (mi_fp_argisnull(fp, 1) == MI_TRUE) ? 0 :
(mi_unsigned_integer)1;
 MI_CONNECTION *conn;
 MI_LO_SPEC  *lo_spec = NULL;
 MI_LO_FD  lo_fd;
 MI_LO_STAT  *lo_stat = NULL;
 char   *buff = NULL;
 char   *pattern = NULL;
 mi_integer  buffsize = 4096;
 mi_integer  found=0;
 mi_integer  result=0;

 pattern = mi_lvarchar_to_string(value);

 if ((conn = mi_open(NULL, NULL, NULL)) == NULL) return (mi_integer) -1;
 if ((buff = (char*)mi_alloc(buffsize)) == NULL)  return (mi_integer) -2;
 if ((lo_fd = mi_lo_open(conn, loptr, MI_LO_RDONLY)) == MI_ERROR)
  return (mi_integer) -3;

 do {
  if ((result = mi_lo_read(conn, lo_fd, buff, buffsize)) == MI_ERROR)
   break;
  if (result == 0)
   break;
  if (strstr(buff,pattern)!=NULL)
    {
     found=1;
    break;
    }
  if (result < buffsize)
   break;
  } while(1);
 mi_lo_close(conn,lo_fd);
 if (buff)
  mi_free(buff);
 return (mi_integer)found;
}
```

2. Compile and link the C routine. We use the following Solaris compile line:

   ```
   cc   -DMI_SERVBUILD -KPIC -I$INFORMIXDIR/incl/public
   -I$INFORMIXDIR/incl/public -I$INFORMIXDIR/incl/esql -I/$INFORMIXDIR/incl -o
   un.o -c un.c
   ```

   Here is the link line:

   ```
   ld -dy -G -Bsymbolic -o un.bld un.o
   ```

3. Set the `bld` file permissions as user `informix`:

   ```
   chmod 555 ./un.bld
   ```

4. Create the function in dbaccess.

```
create function contains(clob,lvarchar)
returns integer
external name '/work/un.bld(contains)'
language C;
```

5. Set up for testing. Create a table and populate it with our c file.

```
CREATE TABLE tclob (c1 INT, c2 CLOB);
INSERT INTO tclob VALUES (1,filetoclob('un.c','server'));
```

6. Test the routine:

```
SELECT c2 FROM tclob;
SELECT CONTAINS(c2,'pattern') FROM tclob;
SELECT CONTAINS(c2,'nopattern') FROM tclob;
SELECT c1 FROM tclob WHERE CONTAINS(c2,'buff')=1;
```

Troubleshooting tips

To track down the cause of problems with C UDRs, the most affective approach is to use a debugger. To debug your UDR, use a debugger that can attach to the active server process and access the symbol tables of the dynamically loaded shared object files. On UNIX and Linux, the `debugger` and `dbx` utilities meet these criteria. To start a debugger, enter the following command at the shell prompt, in which *`pid`* is the process identifier of the CPU or virtual processor:

```
debugger -pid
```

This command starts the debugger on the server virtual-processor process without starting a new instance of the virtual processor. For more information about available debugger commands, see the debugger manual page, and learn more in the information center:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.dapip.doc/sii111026637.htm

10.3 DataBlades and bladelets

Your initial collection of UDRs increases over time. It might not be long until you have UDRs, special functions, and stored procedure UDRs that reduce work complexity and provide great functionality. If you choose not to write your own DataBlade, you can still choose from a nice selection of DataBlade modules. Any DataBlade module you use provides functions that might increase the usefulness of business data dramatically or that might perhaps generate income for you—as a software developer that develops and sells licensed DataBlades.

IBM Informix has several books on the subject of planning, designing, and implementing DataBlades. With a taste of the programming examples presented earlier in this chapter, you might be ready to take the next step and program a DataBlade in C or Java. From a technical perspective, *IBM Informix DataBlade Developers Kit User's Guide, Version 4.20,* G229-6366 is specifically oriented to the development of DataBlade modules. It provides detailed help for programming DataBlade modules in Java and C.

If you want to work with a DataBlade before you decide whether you have a need to write your own, the sections that follow provide an overview of the DataBlades that are available with IBM Informix Server.

10.3.1 Configuration

Configuration for a Datablade module requires the following steps:

1. Prepare the database serve.
2. Install the DataBlade.
3. Register the DataBlade.

Conceptually, the process is the same, regardless of the operating system and hardware. Small differences exist in interface or command lines, which we will point out.

Prepare the database

Database preparation involves setting up environment variables and making sure the database you are going to use with a DataBlade is set to a logged database in advance.

The environment settings required are as follows:

- On UNIX and Linux: LD_LIBRARY_PATH, INFORMIXSERVER, and ONCONFIG
- On Windows: INFORMIXSERVER and ONCONFIG

When you get to the step for BladeManager usage, you also require an environment setting for LD_LIBRARY_PATH.

Although a logged database is not required for every DataBlades, a logged database can help avoid concurrency problems and thus is recommended. Also, while it is the case that not all DataBlade modules require a logged database, some of them do require logged sbspace. Keep this in mind as you install and configure the DataBlade that you select.

Installing the DataBlade

Any DataBlades that are installed with IBM Informix Servers are installed in separate subdirectories under the `INFORMIXDIR/extend` directory. Several subdirectories for DataBlades are established when the IBM Informix Server is installed. There might be some variation to the list, based on your exact version and operating system.

Example 10-11 shows the subdirectory listing for 11.50.UC6 on Linux with J/Foundation.

Example 10-11 A sample INFORMIXDIR/extend subdirectory

```
opt/IBM/informix/extend:> ls

binaryudt.1.0    ifxmngr           LLD.1.20.UC2   spatial.8.21.UC3
bts.2.00           ifxrltree.2.00    mqblade.2.0      web.4.13.UC4
ifxbuiltins.1.1    krakatoa          Node.2.0           wfs.1.00.UC1
```

If you do not see the DataBlade directory reference for the one you want, you must acquire it by way of a download or CD.

Installation on UNIX is simply a matter of uncompressing the new DataBlade module into a temporary directory. After the files are expanded, run the installation script, `./install`. The `./install` script creates a new module directory under the `INFORMIXDIR/extend` directory. Some file expansions can result in more than one new module. If there is more than one module, you must run the `./install` script for each one.

Installation on Windows systems requires that you go to the installation location and run **`setup.exe`**. Select the **Typical** installation option. The other dialog verification is the INFORMIXDIR location. With those items confirmed, the software is installed. When the installation is complete, the module directory is installed in the `INFORMIXDIR\extend` directory.

Registering the DataBlade

With a DataBlade directory in place, the active database server is not aware of the DataBlade directories until the software is registered in the database. **`Registration`** is the process of executing the SQL statements that create the DataBlade module database objects and identify the DataBlade module shared object file or dynamic link library to the database server.

With the release of 11.50.XC4, IBM Informix provides the following distinct methods for DataBlade registration:

- BladeManager

  Available with the first release of datablades, the BladeManager is an interface that automates the registration process by performing a series of SQL steps in the database engine.

- **`sysbldprepare()`**

  This command is an Informix function for DataBlade registration. At its simplest, you can run the command inside dbaccess with your target database open to install a DataBlade. It has a few restrictions that are described in *IBM Informix DataBlade Module Installation and Registration Guide, Version 4.20,* G229-6368. To register the `bts` DataBlade with this interface, use the following command:

  ```
  EXECUTE FUNCTION sysbldprepare('bts.*','create');
  ```

To register a DataBlade module using BladeManager:

1. Start BladeManager:
   - On UNIX or at the MS-DOS prompt, the BladeManager is started with the **`blademgr`** command.
   - To start BladeManager on a Windows system, select **Start** ∅ **Programs** ∅ **Informix program group name** ∅ **BladeManager** or double-click the BladeManager icon in the Informix program group.

   If the BladeManager fails to start, it is either not installed or you do not have the environment variables set, as noted in the previous section.
2. Set confirmations.

   If you want an automatic confirmation after each step, turn the prompt on:

   ```
   set confirm on
   ```

   Commands run when you press the carriage return.
3. Connect to an Informix instance:

   ```
   show servers
   set server demo_on < Use you own server name for demo_on>
   ```

   If you want to connect as a different user, try:

   ```
   set user <username>
   ```

   At the password prompt, enter your password. Validation does not occur until connection, on the next step.

To connect to a database, run one of the following commands:

```
list stores
register module_name database_name
unregister module_name database_name
```

The *module_name* represents the name of the DataBlade module directory. These names typically follow the form of the DataBlade module name followed by the version number.

When BladeManager registers a DataBlade module, it executes a series of SQL CREATE statements to register each database object in the module. You must have resource permissions on the database to register the DataBlade. In addition, if your server has implemented the ONCONFIG EXTEND role, you must be granted the EXTEND role by user `informix`.

If the registration of a module fails, BladeManager returns the database to its prior state. To see the SQL statements that failed, look at the corresponding log file and check the procedure in *Appendix A. Troubleshooting Registration Problems* of *IBM Informix DataBlade Module Installation and Registration Guide, Version 4.20*, G229-6368, for possible solutions.

Occasionally, DataBlade modules have more than one interface. If there are additional modules, there are also dependencies. You have to make sure that each of the interfaces are registered correctly in order for the DataBlade to work. BladeManager automatically checks for dependencies and registers any dependencies it might need. If the BladeManager cannot do the registration, it will prompt you to do so manually.

Important: BladeManager does not verify the integrity of DataBlade modules that have additional interfaces, nor does it not check for the presence of required database objects.

Datablade modules written in the Java language can only be registered in IBM Informix Servers with J/Foundation database servers.

10.3.2 IBM Informix provided DataBlades

Table 10-2 the DataBlades that are available with IBM Informix database servers.

Table 10-2 Informix provided DataBlades

IBM Informix DataBlade module name	Description	Special notes
Large Object Locator	A foundation Datablade module for large objects management that can be used by other modules that create or store large-object data.	Available on standard install (LLD.1.20.UC2)
MQ DataBlade	Allows IBM Informix database applications to communicate with other MQSeries® applications with MQ messaging.	Available on standard install (mqblade.2.0)
Binary DataBlade	This module includes binary data types to store binary-encoded strings that can be indexed for quick retrieval.	Available on standard install. (binaryudt.1.0)
Basic Text Search	Permits text search of words and phrases in an unstructured document repository stored in a column of a table.	Available on standard install. (bts.2.00)
Node DataBlade	This module is for the hierarchical data type, to represent hierarchical data within a relational database.	Available on standard install. (Node.2.0)
Web Feature Service	This module is an add-on to allow Open Geospatial Consortium (OGC) web feature service as a presentation layer for the Spatial and Geodetic DataBlade modules	Available on standard install. (wfs.1.00.UC1)
J/Foundation krakatoa	A library of classes and interfaces that allow programmers to create and execute Java UDRs that access Informix database servers	Available as a part of Informix Server with J/Foundation
ifxbuilt-ins	This is not really a DataBlade, but it sets up definitions and functions for the standard data types offered in the informix server.	Available on standard install (ifxbuiltins.1.1)
ifxmngr.2.00	This is the API that supports the BladeManager.	Available on standard UNIX install

IBM Informix DataBlade module name	Description	Special notes
ifxrltree.2.00	This is a foundational, multidimensional, index called “Region tree” (R-tree) (also known as Range Tree). This blade is needed for both spatial and time related data management.	Available on standard install
Image Foundation	This module is a foundation DataBlade which provides a base on which new or specialized image types and image processing technologies can be added or changed. Because the foundation is open, secure, and scalable, it provides a clear path toward reusing and repurposing valuable image assets.	No charge download
Excalibur Text Search	This module enables provides extensive text-searching capabilities; It supports full-text indexing, including extensive fuzzy-search logic for indexing scanned text. Can search document types including: ASCII, Word, Excel, HTML, PowerPoint, WordPerfect, and PDF. Includes an adaptive pattern recognition process (APRP) and capabilities such as multiple stop-word lists, proximity searching and synonym lists.	License fee applies
Geodetic	This blade supports global space- and time based queries. It is designed to treat earth as a globe rather than a flat plane. Supports client-side Geographic Information Systems (GIS) software.	License fee applies
Spatial	This blade transforms locations and traditional 2-d map data into useful information. Uses SQL-based spatial data types and functions that can be used directly through standard SQL queries or with client-side Geographic Information Systems (GIS) software.	No charge download

IBM Informix DataBlade module name	Description	Special notes
TimeSeries	This module supports data for managing time-series and temporal data. A “time series” is any set of data that is accessed in sequence by time and can be processed and analyzed in a chronological order.	License fee applies.
Video Foundation	This blade allows you to incorporate video servers, external control devices, compression codes, and cataloging tools to manage video content and metadata or information about the content. Allows metadata elements in the database, while allowing video content to be maintained on disk, video tape, video server, or other external storage devices.	License fee applies.
Web	This module supports most web server APIs and has a web client application to build and run SQL queries to work with your database. Enables customized web applications. Allows you to track persistent session variables between AppPages.	License fee applies

10.3.3 Developing a bladelet routine

We can define a *bladelet* as a small, unofficial DataBlade module. It is meant to be useful (and complete with source code) from the time you set it up, but it becomes your own application (with no support or warranty). If you have tried out our UDR development examples, you have a bladelet.

As you might have observed from having to drop and re-create the JAR API in the server in our earlier example, you can understand that if you have a large number of UDRs and have gone to the trouble of creating user-defined data types, the whole package of tasks that are required to set up, change, or update a DataBlade object inside the server might not be a convenient task.

On a large scale, if you have API dependencies, dozens of UDRs, and other DataBlade related objects, you will want to move them and install them as a package. IBM Informix provides a Windows-based interface for this, which has the ability to do the package preparation work for you. The package preparation interface, called *BladeSmith*, allows you to populate a properties definition dictionary.

When the properties are all defined and you proceed, BladeSmith creates, assembles, and arranges a directory structure with a complete tree layout of all the components that are required and handled in a DataBlade registration process. The resulting directory tree layout and assembly pieces provide a prepared package that is ready to ship to an operating system of your choice. Likewise, any language source code that is output for the preparation task is parcelled out for the appropriate component nodes also.

The dispensation for the BladeSmith file components and directory structure is laid out based on the type of component node, as described in Table 10-3.

Table 10-3 BladeSmith file package creation layout

Component node	What is generated
Source	All source code in the coding languages you use for your DataBlade module objects
Client	Client code (ActiveX or Java)
Server	Server code in the coding language you specified for BladeSmith
Individual language	Source code for the represented language (C, Java, or SPL)

For further information about creating datablade objects using BladeSmith generating files, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.dbdk.doc/sii-smith-27272.htm

For further information or examples of existing bladelets and downloads developed by users, refer to:

http://www.ibm.com/developerworks/data/zones/informix/library/samples/db_downloads.html

There is also a downloads page that features bladelets at the International Informix Users Group site called *"ORDBMS - Object-Relational Database Extensibility, DataBlades"* at:

http://www.iiug.org/software/index_ORDBMS.html

11

Working with Ruby on Rails

This chapter contains information about how to develop applications against an IBM Informix database using the Ruby programming language and the Ruby on Rails web development framework.

In this chapter, we discuss the following topics:

- ► A brief overview of Ruby on Rails
- ► Setup and configuration
- ► Database operations
- ► Using the Rails Adapter with Ruby Informix
- ► Using the Rails Adapter with IBM_DB

11.1 A brief overview of Ruby on Rails

Ruby is a open source programming scripting language with a focus on simplicity and productivity. Ruby is similar to other scripting languages like Perl or Python with the difference that Ruby is an *object-oriented language*. Ruby is both technology- and platform-independent. You can find implementations of the Ruby run time on C, Java, and even .NET, making Ruby a useful option for any scripting need.

For more information about the Ruby language, refer to:

http://www.ruby-lang.org/en/about/

Rails is an open source Ruby framework for developing database-backed web applications. Rails expands the object-orientated core design of Ruby, helping developers to build websites and applications with minimum coding efforts.

Rails is based on two key principles:

- *Convention over configuration* (CoC), where developers need to focus only on the exceptions to the conventions. Every other aspect of the application, from design to implementation, is done automatically by the defined conventions.

  The conventions define rules such as use a plural for the table names. For example, if the application uses an entity called *Book*, the table that stores this entity must be called *Books*, and if it stores details about a **`Person`**, the table must be called *People*.
- *Don't Repeat Yourself* (DRY). Information is located in one place only. Database object definitions, documentation, and configuration scripts all are kept in one location and are consulted when information about them is required.

For more information, refer to:

http://rubyonrails.org/documentation

11.1.1 Architecture of Ruby on Rails

Ruby on Rails is built with the Model, View, Controller (MVC) architecture that is typically used in web-based GUI programming. This architecture has three main concepts:

- Model

  The business logic of the system, which encompasses the persistence layer because it interacts with a database back end.

- View

  The GUI interface of the model that is visible to the user. One Model can have many views.

- Controller

  The action taken by the user using the view. The controller takes inputs from the user through the view and executes the business logic encapsulated in the model.

The Rails framework provides a set of utilities and components that are designed to facilitate the development of web applications:

- *Rake* is a build tool that is bundled with the Ruby programming language. It is the equivalent to the **make** command on UNIX.
- *WEBrick* is the web server that is bundled with Ruby on Rails.
- *ActiveRecord* is the object-relational mapper of Rails and provides for persistence. It presents the database table as a class, which in Rails is called *model*.
- *Action Controller* is the component that manages the controllers in a Rails application. It also processes and dispatches incoming requests.
- *Action View* manages the views in a Rails application.

11.1.2 Ruby Driver and Rails Adapter

IBM Informix supports database access for client applications written in the Ruby programming language and web application development with the Rail framework.

Ruby Driver

To use an database with Ruby, the application requires a Ruby driver. This driver provides the layer that connect the Ruby run time with the database server.

The following drivers allow Ruby to connect to an IBM Informix database:

- The Ruby Informix driver is an open source project supported by the open source community. It allows Ruby to connect to any IBM Informix database server. The Ruby driver is developed using the IBM Informix ESQL/C language that provides full support for all the Informix database features and data types. Because it uses the Informix Client Software Development Kit (Client SDK) libraries,
- Ruby Driver for IBM Data Servers driver (IBM_DB) is provided, supported, and developed by IBM as an open source project. The Ruby driver is bundled together with the Rails Adapter in the Rails Adapter/Driver for IBM Data Servers package.

Rails Adapter

A Rails Adapter is a Ruby script that allows you to use a specific Ruby driver within the Rails framework. It provides the required Ruby objects, for example the ActiveRecord object, that enable the full use of the Ruby driver inside the Rails framework.

IBM Informix supports the following Rails adapters:

- `informix_adapter.rb`, used in conjunction with the Ruby Informix driver. Requires Client SDK libraries for the communication with the Informix database server.
- `ibm_db_adapter.rb`, used with the Ruby Driver for IBM Data Servers. Requires the IBM Data Server Driver for ODBC and CLI package.

Both adapters are available from the Ruby repository as Ruby *gems*. Ruby gems are self-contained packages that contain all the libraries, source files, and scripts needed for the Ruby component.

11.2 Setup and configuration

This section describes how to set up and configure both Ruby drivers and Rails adapters for use with an IBM Informix database.

11.2.1 Ruby Informix driver

The Ruby Informix driver is available for download at the SourceForge website at:

http://rubyforge.org/projects/ruby-informix

You can use the Ruby `gem` utility to download the Ruby Informix driver automatically from the Ruby repository and install it in the Ruby environment.

Run the following command from your Ruby session to install the Ruby Informix driver:

```
gem install ruby-informix
```

Because the driver shared library, `informixc.so`, is built during the installation process, the environment should contain the correct settings for compiling ESQL/C applications. Refer to Chapter 4, "Working with ESQL/C" on page 125 for more information about ESQL/C settings.

You can find more information about the installation process in the README file inside the `gem` directory. See Example 11-1 for the list of files that are included with the Ruby Informix driver.

Example 11-1 The gem directory

```
Directory of C:\work\Ruby187\lib\ruby\gems\1.8\gems\ruby-informix-0.7.3

03/07/2010  19:34    <DIR>          .
03/07/2010  19:34    <DIR>          ..
03/07/2010  19:19             8,383 Changelog
03/07/2010  19:19             1,470 COPYRIGHT
03/07/2010  19:29    <DIR>          ext
03/07/2010  19:35    <DIR>          lib
03/07/2010  19:19             4,500 README
03/07/2010  19:19    <DIR>          test
               3 File(s)         14,353 bytes
               5 Dir(s)  76,168,769,536 bytes free
```

Configuration

The Ruby Informix uses the same connectivity information as other Client SDK components. It uses the INFORMIXDIR environment variable to locate the libraries and resources such as error message files or configuration files.

By default, the Ruby driver connects to the database server specified in the INFORMIXSERVER environment variable. Same as the other Client SDK components, the information regarding the INFORMIXSERVER value is stored on the `sqlhosts` file or the Windows registry. For more information, refer to "Connectivity on UNIX" on page 27.

The shared library search path variables, for example, LD_LIBRARY_PATH or SHLIB_PATH, must contain the `$INFORMIX/libl` and `$INFORMIX/lib/esql` directories. Otherwise, the Ruby driver might fail to load the ESQL/C libraries that it requires for work.

Data types

The Ruby Informix driver provides the data types to be used against an IBM Informix database. The driver provides specific types such as `Informix::IntervalYTM` or `Informix::Slob` to handle specific Informix types.

Table 11-1 shows the data type mapping between the Ruby Informix driver and the Informix database.

Table 11-1 Ruby Informix data type mapping

Informix data type	Ruby data type
SMALLINT, INT, INT8, FLOAT, SERIAL and SERIAL8	Numeric
CHAR, NCHAR, VARCHAR, NVARCHAR	String
DATE	Date
DATETIME	TIME
INTERVAL	Informix::IntervalYTM, Informix::IntervalDTS
DECIMAL, MONEY	BigDecimal
BOOL	TrueClass, FalseClass
BYTE, TEXT	StringIO, String
CLOB, BLOB	Informix::Slob

Verifying connectivity

Ruby includes an interactive shell called *irb* that you can use to run simple Ruby statements. The `irb` is located in the `bin` directory of the Ruby installation.

The driver name used inside the Ruby scripts to reference the Ruby Informix driver is *`informix`*.

To test whether the Ruby driver can connect to a database, you must load the Ruby driver and then create a connection using the Ruby Informix object.

Example 11-2 demonstrates how to load the Ruby Informix driver and connect to an IBM Informix database. The fist command, `require 'informix'`, tells the Ruby run time to load the Ruby Informix driver. After that, it creates a Ruby Informix connection object and prints the database version information.

Example 11-2 Testing connection with Ruby informix

```
C:\work>irb
irb(main):001:0> require 'informix'
=> true
irb(main):002:0> db=Informix.connect('stores_demo','informix','password')
=> #<Informix::Database:0x8176110>
irb(main):003:0> puts db.version
IBM Informix Dynamic Server Version 11.50.FC6
=> nil
irb(main):004:0>
```

11.2.2 Data Server Ruby driver

The Data Server Ruby driver uses the Data Server CLI driver to connect the Informix database. It uses the DRDA protocol so the version of the IBM Informix database server must be 11.10 or 11.50.

The driver is included with IBM Data Server Driver and is also available to download directly from the RubyForge website:

http://rubyforge.org/projects/rubyibm/

You can install the complete package using the Ruby `gem` utility by running the following command from a Ruby session:

```
gem install ibm_db
```

For information about the build and setup process for Data Server Driver, consult the README file in the driver directory.

Example 11-3 shows the Data Server Ruby driver directory.

Example 11-3 The ibm_db gem directory

```
Directory of C:\work\Ruby187\lib\ruby\gems\1.8\gems\ibm_db-0.10.0-x86-mswin32

03/07/2010  17:30    <DIR>          .
03/07/2010  17:30    <DIR>          ..
03/07/2010  17:30             6,063 CHANGES
03/07/2010  17:30    <DIR>          ext
03/07/2010  17:30             1,656 init.rb
03/07/2010  18:06    <DIR>          lib
```

```
03/07/2010  17:30             1,088 LICENSE
03/07/2010  17:30               299 MANIFEST
03/07/2010  17:30            13,402 README
03/07/2010  17:30    <DIR>          test
               5 File(s)         22,508 bytes
               5 Dir(s)  76,167,290,880 bytes free
```

You can find additional information regarding the Ruby driver for IBM Data Servers at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.ruby.doc/doc/t0052765.html

On some platforms, such as Windows operating system platforms, the shared library for the Ruby driver is already included in the package. Therefore, there is no need for a C build environment.

Example 11-4 shows the Ruby driver directory from Data Server Client on a Windows system after the installation.

Example 11-4 Windows system Data Server Ruby directory

```
C:\work>dir "C:\Program Files\IBM\IBM DATA SERVER DRIVER\ruby"
 Volume in drive C is W2003
 Volume Serial Number is 50DA-70D7

 Directory of C:\Program Files\IBM\IBM DATA SERVER DRIVER\ruby

17/05/2010  02:16    <DIR>          .
17/05/2010  02:16    <DIR>          ..
30/05/2009  11:09           198,144 ibm_db-0.10.0-mswin32.gem
               1 File(s)        198,144 bytes
               2 Dir(s)  76,167,372,800 bytes free

C:\work>
```

Configuration

The Data Server Ruby driver uses the Data Server ODBC/CLI driver for the connection to the database.

The configuration details are the same as with the ODCB/CLI driver. These details are usually kept in the `db2profile.ini` file. Refer to 2.2.3, “Setting up IBM Data Server drivers” on page 43 for detailed information about Data Server Driver configuration.

Data types

There are no specific Ruby objects to use Informix data types. Data Server Ruby driver supports the same Informix data types as Data Server Driver for CLI.

Verifying connectivity

We use an `irb` session to test the connectivity against an IBM Informix database server.

The reference name of the Data Server Ruby driver is `ibm_db`. You must load this driver before creating the Ruby connection object.

Example 11-5 shows how to load the Ruby driver and how to connect to the Informix server.

Example 11-5 Testing connection with data server Ruby driver

```
C:\work>irb
irb(main):001:0> require 'mswin32/ibm_db'
=> true
irb(main):002:0> db=IBM_DB.connect 'dsc_dsn','informix','password'
=> #<IBM_DB::Connection:0x81fb6d0>
irb(main):003:0>
```

First, load the Ruby driver with `require 'mswin32/ibm_db'`. Then, open the connection using the `IBM_DB::connect` method.

Note: Because we use the Windows version of the Ruby driver, we must prefix the driver name with the `mswin32` directory.

You can also made a dsn-less connection by specifying all the required parameters in the connection string as follows:

```
IBM_DB.connect 'DRIVER={IBM DB2 ODBC DRIVER};DATABASE=stores_demo;
HOSTNAME=kodiak;PORT=9089;PROTOCOL=TCPIP;UID=informix;PWD=password;', '', ''
```

11.2.3 Rails adapters

The Ruby Informix Rails Adapter provides the Ruby ActiveRecord object that makes the use of the Ruby driver to work on the Rails framework possible.

The Rails Adapter for the Ruby Informix driver is a free download that is available at:

http://rubyforge.org/projects/rails-informix/

The Rails Adapter can be installed automatically as a Ruby gem using the `gem` utility. Example 11-6 shows how to install the Ruby Informix adapter.

Example 11-6 Installing the Ruby Informix adapter

```
C:\work>gem install activerecord-informix-adapter -v 1.1.1
Successfully installed activerecord-informix-adapter-1.1.1
1 gem installed
Installing ri documentation for activerecord-informix-adapter-1.1.1...
Installing RDoc documentation for activerecord-informix-adapter-1.1.1...

C:\work>
```

The configuration of the adapter depends on the version of Rails that is installed in the Ruby environment. On versions older than 2.x, you need to copy the `informix_adapter.rb` adapter script file into the `connection_adapters` directory.

Example 11-7 shows the `connection_adapter` directory with both Rails adapters installed.

Example 11-7 The connection _adapter directory

```
Directory of
C:\work\Ruby187\lib\ruby\gems\1.8\gems\activerecord-1.15.6\lib\active_record\co
nnection_adapters

3/07/2010  18:02    <DIR>          .
3/07/2010  18:02    <DIR>          ..
3/07/2010  17:01    <DIR>          abstract
3/07/2010  17:01             5,235 abstract_adapter.rb
3/07/2010  17:01             6,957 db2_adapter.rb
3/07/2010  17:01            27,749 firebird_adapter.rb
3/07/2010  17:01            30,751 frontbase_adapter.rb
3/07/2010  17:30            70,854 ibm_db_adapter.rb
3/07/2010  20:06             9,916 informix_adapter.rb
3/07/2010  17:01            13,774 mysql_adapter.rb
3/07/2010  17:01            11,531 openbase_adapter.rb
3/07/2010  17:01            25,897 oracle_adapter.rb
3/07/2010  17:01            21,513 postgresql_adapter.rb
3/07/2010  17:01            13,162 sqlite_adapter.rb
3/07/2010  17:01            22,087 sqlserver_adapter.rb
3/07/2010  17:01            22,622 sybase_adapter.rb
             13 File(s)        282,048 bytes
              3 Dir(s)  76,166,725,632 bytes free
```

When using versions of Rails older than 2.x, it is also required to include the reference name of the Ruby driver in the RAILS_CONNECTION_ADAPTER parameter in the `ActiveRecord` Ruby script (see Example 11-8).

Example 11-8 The active_record file

```
C:\work\Ruby187\lib\ruby\gems\1.8\gems\activerecord-1.15.6\lib>grep informix
active_record.rb
  RAILS_CONNECTION_ADAPTERS = %w( mysql postgresql sqlite firebird sqlserver
db2 oracle sybase openbase frontbase informix ibm_db )

C:\work\Ruby187\lib\ruby\gems\1.8\gems\activerecord-1.15.6\lib>
```

After creating a Rails project, you must update the project configuration file that contains the database information, `database.yml`, with the connection details of the IBM Informix database. This file is located in the `<project>/config` directory and has three sections:

- Development
- Test
- Production

These sections point to databases on the respective environments. The database connectivity properties include:

- adapter: The Ruby driver used. You do not have to give the complete version.
- database: Database to which to connect.
- username and password: To connect to the Informix server.
- server: The system on which the Informix server is running.
- port: The DRDA port on the Informix server.

Example 11-9 shows a typical `database.yml` file with the details for an IBM Informix database. The Rails application is named `stores7`, and we connect to an Informix server called `demo_on`.

We use the Ruby Informix driver reference name for the adapter `parameter informix`. The *`server`* parameter identifies the Informix server to connect to. The information about the Informix server, such as host and port, is retrieved from the `sqlhosts` file or Windows registry.

Example 11-9 The database.yml file for the Ruby informix Adapter

```
C:\work\stores7\config>type database.yml

development:
  adapter: informix
  database: stores7
  pool: 5
  timeout: 5000
  server: demo_on
  username: informix
  password: password

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: stores7
  username: root
  password:
  host: localhost

production:
  adapter: informix
  database: stores7
  pool: 5
  timeout: 5000
  server: demo_on
  username: informix
  password: password

C:\work\stores7\config>
```

Rails Adapter for IBM Data Server

The Rails Adapter for the IBM Data Server Ruby driver is bundled with the Ruby driver. It is installed when the Ruby driver is installed.

In the same way as with the Ruby Informix Rails Adapter, if the Rails Adapter for IBM Data Server is installed on a version of Rails older than 2.x, the adapter script (`informix_adapter.rb`) must be copied in the `connection_adapter` directory, and the RAILS_CONNECTION_ADAPTER parameter in the `active_record.rb` Ruby script has to be updated.

The database configuration file for a Rails project, `database.yml`, has to be updated with the details for the Informix server.

Example 11-10 shows a `database.yml` file that is used in a Rails project that connects to an IBM Informix database.

Example 11-10 The database.yml file for the Rails Adapter for IBM Data Servers

```
C:\work\stores7\config>type database.yml

development:
  adapter: ibm_db
  database: stores7
  pool: 5
  timeout: 5000
  host: kodiak
  port: 9089
  username: informix
  password: password

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: stores7
  username: root
  password:
  host: localhost

C:\work\stores7\config>
```

The `adapter` parameter is set to the reference name of the Data Server Ruby driver `ibm_db`. It also contains the host name and the port parameter with the details about the DRDA Informix instance. These details are the same as used in the Data Server driver for ODBC/CLI.

For a list of all the parameters, refer to:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.dbclient.ruby.doc/doc/t0052780.html

11.3 Database operations

This section provides examples of using both Ruby drivers to perform basic operations against an IBM Informix database server. It also demonstrates how to use the Rails Adapter to create basic web applications.

Using Ruby Informix driver

In this section, we discuss the basic database operation using the Ruby Informix driver.

Connection to the database

Example 11-11 shows how to connect to an Informix database. The database name, user name, and password are passed as the parameters in the command line.

Example 11-11 The ifx_connect.rb output

```
C:\work>type ifx_connect.rb
# load the informix driver
require 'informix'
# Connect to the database
db = Informix.connect(ARGV[0],ARGV[1],ARGV[2])
# print database information
print "Connected to #{db.version}"
db.close

C:\work>ruby ifx_connect.rb stores_demo informix password
Connected to IBM Informix Dynamic Server Version 11.50.FC6
C:\work>
```

Getting information about a table

Example 11-12 demonstrates how to use the Ruby connection object to retrieve metadata information about the `state` table. It uses the `columns()` method of the Informix connection object to retrieve a Ruby array with all the metadata information.

Example 11-12 The ifx_metadata.rb output

```
C:\work>cat ifx_metadata.rb

# load the informix driver
require 'informix'
# Connect to the database
db = Informix.connect(ARGV[0])
# print database information
print "Connected to #{db.version}\n"
db.columns(ARGV[1]).each {
 |name| name.each {|elem| print " #{elem[0]} #{elem[1]}\n"}
}
db.close
C:\work>ruby ifx_metadata.rb stores_demo state
Connected to IBM Informix Dynamic Server Version 11.50.FC6
 precision 0
 type 0
 scale 0
 length 2
 xid 0
 nullable true
 name code
 stype CHAR
...
```

Executing a simple SQL statement

Using the Ruby driver, you can run an SQL statement directly from the Ruby connection object or through a prepared statement object.

Example 11-13 demonstrates how to run a simple SQL statement using a prepared statement Ruby object. The `execute()` method returns the number of affected rows.

Example 11-13 The ifx_execute.rb output

```
C:\work>cat ifx_execute.rb

# load the informix driver
  require 'informix'
# Connect to the database
db = Informix.connect(ARGV[0])
# print database information
  print "Connected to #{db.version}\n"

# create a prepare object with the SQL passed
stmt = db.prepare(ARGV[1])
# Execute the prepared statement
  rc=stmt.execute()
  print "Result=#{rc}"
  db.close

C:\work>ruby ifx_execute.rb stores_demo "DELETE from STATE where CODE='AR'"
Connected to IBM Informix Dynamic Server Version 11.50.FC6
Result=1
C:\work>
```

If the SQL statement returns only one row, you can use the `execute()` method to retrieve that value (see Example 11-14).

Example 11-14 The ifx_execute.rb output

```
C:\work>ruby ifx_execute.rb stores_demo "SELECT sname FROM state WHERE
code='CA'"
Connected to IBM Informix Dynamic Server Version 11.50.FC6
Result=snameCalifornia
C:\work>
```

Using parameters

Example 11-15 shows how to prepare and execute a parametrized INSERT statement using the Ruby driver.

Example 11-15 The ifx_parameters output

```
C:\work>cat ifx_insert.rb

# load the informix driver
  require 'informix'
# Connect to the database
db = Informix.connect('stores_demo')
# print database information
  print "Connected to #{db.version}\n"
# create a prepare object with the SQL passed
  stmt =  db.prepare('INSERT INTO state(code,sname) VALUES (?,?)')
# Execute the statement using parameters
  rc=stmt.execute(ARGV[0],ARGV[1])
# Execute the prepared statement
  print "Result=#{rc}"
  db.close

C:\work>ruby ifx_insert.rb AR Arizona
Connected to IBM Informix Dynamic Server Version 11.50.FC6
Result=1

C:\work>
```

Selecting data from the database

When selecting multiple records from the database the application must create a cursor object to fetch the selected rows. Example 11-16 illustrates how to return information using an Informix cursor.

Example 11-16 The ifx_cursor.rb output

```
C:\work>cat ifx_cursor.rb

# load the informix driver
  require 'informix'
# Connect to the database
  db = Informix.connect('stores_demo')
# display database information
  print "Connected to #{db.version}\n"
# Create a Cursor object
  cur = db.cursor(ARGV[0])
# Open the cursor and fetch some rows
  cur.open.each {|rows|
   puts rows*" = "
  }.close
# Close connection
  db.close
C:\work>ruby ifx_cursor.rb "SELECT FIRST 3 sname FROM state"
Connected to IBM Informix Dynamic Server Version 11.50.FC6
Alaska
Hawaii
California

C:\work>
```

Using the Ruby Informix IfxSlob class

The `Slob` class is the Ruby interface for handling smart large objects. It provides methods for every action applicable with a smart large object. By using the `Informix::Slob` class, it is possible to perform the same operations against a smart large object as with other programming languages and drivers. The `Slob` methods such as `Seek()` or `Lock()` allow random I/O to individuals part of the large object that can only be achieved using this Ruby driver.

Example 11-17 shows how to select a BLOB from a database table. It retrieves the `catalog_advert` CLOB column from the `catalog` table and prints out the content of the large object and the size of the large object.

Example 11-17 The ifx_blob.rb output

```
C:\work>cat ifx_blob.rb
# load the informix driver
  require 'informix'
# Connect to the database
  db = Informix.connect("stores_demo")
# Creates an Informix Slob object
  slob = Informix::Slob
# Opens a cursor to retrieve
  cur = db.cursor("SELECT catalog_num,advert_descr FROM catalog WHERE 
catalog_num=?")
  cur.open(ARGV[0]).each {|rows|
    slob = rows[1].open
    print "Number   = #{rows[0]}\n"
#   Reads the blob data as a String
    print "Clob data= #{rows[1].read(rows[1].size)}\n"
    print "Clob Size= #{rows[1].size}\n"
# Close the Slob object
    slob.close
  }
# Close database connection
  db.close

C:\work>
C:\work>ruby ifx_blob.rb 10001
Number   = 10001
Clob data= Brown leather. Specify first baseman's or infield/outfield style.
Clob Size= 98

C:\work>ruby ifx_blob.rb 10027
Number   = 10027
Clob data=
Double or triple crankset with choice of chainrings or chunky bacon. For double 
crankset...
Clob Size= 154

C:\work>ruby ifx_blob.rb 10031
Number   = 10031
Clob data= No buckle so no plastic touches your chin. Meets both ANSI and 
Snell...
Clob Size= 123

C:\work>
```

For examples and a full description of all the method implemented by the `Informix::Slob` class, refer to:

http://ruby-informix.rubyforge.org/doc/classes/Informix/Slob.html

Using the Ruby Informix INTERVAL

The Ruby Informix driver provides a specific class to deal with the Informix INTERVAL data type.

Example 11-18 shows how to define and use the `Informix::Interval` class. In this example we create an Interval Year to Month with one year and one month as the value. The code performs a simple arithmetic operation adding the Interval to the current date.

Example 11-18 The ifx_interval.rb output

```
# Creates an Informix Interval object
  minterval = Informix::Interval.year_to_month(1, 1)
  print "Interval     \t=#{minterval}\n"
  today = Date.today
  print "Current date \t=#{today}\n"
  print"Interval+today\t=#{minterval + today}\n"

C:\work>ruby ifx_interval.rb
Interval        =1-01
Current date    =2010-07-04
Interval+today  =2011-08-04

C:\work>
```

You can fine the full documentation about all the methods supported by the Ruby Informix driver at:

http://ruby-informix.rubyforge.org/doc/

Using Data Server Ruby driver

In this section, we discuss the basic database operation using the Data Server Ruby driver.

Connecting to the database

The syntax for opening a connection with an Informix database server using the Data Server Ruby driver differs from the syntax used by the Ruby Informix driver.

Example 11-19 shows a simple Ruby script that creates a connection object and opens the connection. In the example code we also retrieve information about the Informix server using the `IBM_DB::server_info` class.

Example 11-19 The dsc_connect.rb script

```
C:\work>cat dsc_connect.rb

# load the informix driver
  require 'mswin32/ibm_db'
# Connect to the database
  db = IBM_DB.connect(ARGV[0],ARGV[1],ARGV[2])
  info = IBM_DB.server_info(db)
# display database information
  print "Connected to #{info.DBMS_NAME} #{info.DBMS_VER}"


C:\work>ruby dsc_connect.rb testdsc informix password
Connected to IDS/NT64 11.50.0000
C:\work>
```

The Data Server Ruby driver is based on calls to the CLI driver, this means it takes the connection details from the `db2cli.ini` configuration file. Example 11-20 shows the contents of the `db2cli.ini` file that we used for this test.

Example 11-20 The db2cli.ini file

```
C:\work>type "c:\Documents and Settings\Administrator\db2cli.ini"

[dsc_dsn]
Protocol=TCPIP
Port=9089
Hostname=kodiak
Database=stores_demo
PWD=password
UID=informix

C:\work>
```

Executing an SQL statement

Example 11-21 demonstrates how to execute an SQL statement using the `prepare()` and `execute()` methods of the IBM_DB driver. Both methods, `prepare()` and `execute()`, require the `connection` and `statement` objects as parameters.

Example 11-21 The dsc_execute.rb output

```
C:\work>cat dsc_execute.rb

# load the informix driver
  require 'mswin32/ibm_db'
# Connect to the database
  db = IBM_DB.connect(ARGV[0],'','')
  info = IBM_DB.server_info(db)
# display database information
  print "Connected to #{info.DBMS_NAME} #{info.DBMS_VER}\n"
# create a prepare object with the SQL passed
  stmt = IBM_DB.prepare(db,ARGV[1])
# Execute the prepared statement
  rc=IBM_DB.execute(stmt)
  print "Result=#{rc}"

C:\work>ruby dsc_execute.rb dsc_dsn "DELETE from STATE WHERE code='NW'"
Connected to IDS/NT64 11.50.0000
Result=true
C:\work>
```

Parametrized SQL statement

When using parameters for an SQL statement, the application can use the `execute()` method. You can provide the parameters as a second argument for the method call.

Example 11-22 insert a new record into the `state` table with the values passed to the script through the command line.

Example 11-22 The dsc_param.rb output

```
C:\work>cat dsc_param.rb

# load the informix driver
  require 'mswin32/ibm_db'
# Connect to the database
  db = IBM_DB.connect('dsc_dsn','','')
  info = IBM_DB.server_info(db)
# display database information
  print "Connected to #{info.DBMS_NAME} #{info.DBMS_VER}\n"
  sql = 'INSERT INTO state(code, sname) VALUES (?,?)'
```

```
# create a prepare object
  stmt = IBM_DB.prepare(db,sql)
# display Statement and parameters
  print "SQL=#{sql}\n"
# Execute the prepared statement
  rc=IBM_DB.execute(stmt,[ARGV[0],ARGV[1]])
  print "Result=#{rc}"

C:\work>
C:\work>ruby dsc_param.rb "NW" "NewState"
Connected to IDS/NT64 11.50.0000
SQL=INSERT INTO state(code, sname) VALUES (?,?)
Result=true
C:\work>
```

Selecting data

The Data Server Ruby driver has several methods that allow retrieving data from the database, such as *f*etch_array(), fetch_assoc(), and fetch_row().

Example 11-23 shows a simple Ruby script that returns the first two columns of an SQL SELECT statement passed through the command line. The script uses the fecth_array() method to retrieve the rows as an array object.

Example 11-23 Select data using fetch_array()

```
C:\work>cat dsc_fetch.rb

# load the informix driver
  require 'mswin32/ibm_db'
# Connect to the database
  db = IBM_DB.connect(ARGV[0],'','')
  info = IBM_DB.server_info(db)
# display database information
  print "Connected to #{info.DBMS_NAME} #{info.DBMS_VER}\n"
# create a prepare object with the SQL passed
  stmt = IBM_DB.prepare(db,ARGV[1])
# Execute the prepared statement
  IBM_DB.execute(stmt)
  while row = IBM_DB.fetch_array(stmt)
    puts "#{row[0]}:#{row[1]}"
  end


C:\work>ruby dsc_fetch.rb dsc_dsn "SELECT FIRST 3 code,sname FROM state"
Connected to IDS/NT64 11.50.0000
AK:Alaska
HI:Hawaii
CA:California

C:\work>
```

Using smart large objects

The Data Server Ruby driver handles smart large objects as normal data types. It does not support all the smart features that are normally available with other drivers. However, it simplifies the code that is needed to deal with these data types.

Example 11-24 retrieves a CLOB column from the `catalog` table and displays the contents.

Example 11-24 The dsc_blob.rb file

```
C:\work>cat dsc_blob.rb

# load the informix driver
  require 'mswin32/ibm_db'
# Connect to the database
  db = IBM_DB.connect('dsc_dsn','','')
  info = IBM_DB.server_info(db)
# display database information
  print "Connected to #{info.DBMS_NAME} #{info.DBMS_VER}\n"
# create a prepare object with the SQL passed
  sql = "SELECT catalog_num,advert_descr FROM catalog WHERE catalog_num=?"
  stmt = IBM_DB.prepare(db,sql)
# Execute the prepared statement
  IBM_DB.execute(stmt,ARGV)
  while row = IBM_DB.fetch_array(stmt)
    puts "#{row[0]}:#{row[1]}"
  end

C:\work>ruby dsc_blob.rb 10001
Connected to IDS/NT64 11.50.0000
10001:Brown leather. Specify first baseman's or infield/outfield style.
Specify right- or left-handed.

C:\work>
```

You can find additional documentation about the methods for the Ruby for IBM Data Server at:

http://rubyibm.rubyforge.org/docs/driver/2.0.0/doc/

You can also find information in the IBM DB2 Information Center:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.dbclient.ruby.doc/doc/c0052760.html

11.4 Using the Rails Adapter with Ruby Informix

In this section, we demonstrate how to create a basic web application using the Rails framework with the Ruby Informix Adapter.

One of the key concepts of Rails is *Convention over Configuration*, meaning that you must follow the convention rules when designing your database so that the Rails framework can generate code automatically to handle typical operations with the database.

The following convention rules are required by the Rails framework:

- Table name must be a *plural* name for the *entity* it contains. For example, if the table contains information about books, it should be called Books.
- The table must contains an unique primary key column and should be called ID.
- You must create an Informix SQL SEQUENCE for each of the tables used for both Ruby drivers. The name of the sequence must be `tablename_seq`.

You can find additional information about Rails conventions at:

http://guides.rubyonrails.org/

11.4.1 Creating database objects

The following examples use the `orders` and `items` tables from the `stores7` database. Due to the convention rules used on Rails, we must modify the schema of the tables to follow the Rails conventions.

Example 11-25 shows the SQL script we use to change the name of the two tables, create views to incorporate the ID column, and create the SQL sequence required for object reference.

Example 11-25 The setup.sql script

```
-- Orders table
RENAME TABLE orders TO order;
CREATE VIEW orders(
 id,
 order_date,
 customer_num,
 ship_instruct,
 backlog,
 po_num,
 ship_date,
 ship_weigh,
 ship_charge,
 paid_date) AS
 SELECT * FROM order;

CREATE SEQUENCE orders_seq;

-- Items table
RENAME TABLE items TO item;
CREATE VIEW items (
 id,
 order_num,
 stock_num,
 manu_code,
 quantity,
 total_price) AS
 SELECT * FROM item;

CREATE SEQUENCE items_seq;
```

11.4.2 Creating the Rails application

You must create a Rails application using Rails commands before adding any objects or definitions.

Example 11-26 shows the output of the **rail stores** command.

Example 11-26 Output of the rail stores command

```
C:\work>rails stores
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
      create  components
      create  db
      create  doc
      create  lib
      create  lib/tasks
      create  log
      create  public/images
      create  public/javascripts
      create  public/stylesheets
...
```

For more information, refer to:

http://rubyonrails.org/documentation

11.4.3 Modifying the database configuration file

The `database.yml` file must include the connection details of the database.

Example 11-27 shows the database configuration file used by our application.

Example 11-27 The database.yml file

```
C:\work\stores>type config\database.yml
development:
  adapter: informix
  database: stores7
  pool: 5
  timeout: 5000
  server: demo_on
  username: informix
  password: password

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: stores7
  username: root
  password:
  host: localhost

production:
  adapter: informix
  database: stores7
  pool: 5
  timeout: 5000
  server: demo_on
  username: informix
  password: password

C:\work\stores>
```

11.4.4 Creating the Rails model and controllers

To make Rails aware of the database tables, you create model and controller in your application. We use the model and controller Ruby script to create model and controller for the two tables used in our application.

Example 11-28 shows the batch script used to generate the model and controller for the `Items` and `Orders` table.

Example 11-28 The objects.cmd script

```
ruby script\generate model Order
ruby script\generate controller Order
ruby script\generate model Item
ruby script\generate controller Item
```

Example 11-29 shows the output of each of the commands in the `objects.cmd` batch script.

Example 11-29 Output of the objects.cmd script

```
C:\work\stores>ruby script\generate model Order
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/order.rb
      create  test/unit/order_test.rb
      create  test/fixtures/orders.yml
      exists  db/migrate
      create  db/migrate/004_create_orders.rb

C:\work\stores>ruby script\generate controller Order
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/order
      exists  test/functional/
      create  app/controllers/order_controller.rb
      create  test/functional/order_controller_test.rb
      create  app/helpers/order_helper.rb


C:\work\stores>ruby script\generate model Item
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/item.rb
      create  test/unit/item_test.rb
      create  test/fixtures/items.yml
      exists  db/migrate
      create  db/migrate/005_create_items.rb

C:\work\stores>ruby script\generate controller Item
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/item
      exists  test/functional/
      create  app/controllers/item_controller.rb
      create  test/functional/item_controller_test.rb
      create  app/helpers/item_helper.rb

C:\work\stores>
```

After the objects are created, you must modify the controller Ruby file for each object to build the object scaffold. We use Rails 1.2.6 on our examples. Rails 2.x does not support dynamic scaffolding. This means it cannot retrieve the information for the table column dynamically. In this case, the scaffold for the objects must be created manually while creating the model object. We add the instruction `scaffold: object_name` to each of the files for creating scaffold.

Example 11-30 shows the Ruby script file for the `Item` and `Order` controllers.

Example 11-30 Controller script

```
C:\work\stores>cat app/controllers/order_controller.rb
class OrderController < ApplicationController
        scaffold :Order
end

C:\work\stores>cat app/controllers/item_controller.rb
class ItemController < ApplicationController
        scaffold :Item
end

C:\work\stores>
```

For more information about the changes in Rails 2.x, refer to:

http://rubyonrails.org/documentation

11.4.5 Starting the Rails web server

To start the Rails web server use the **`ruby script/server`** command.

Example 11-31 demonstrates how to start the server.

Example 11-31 Rails web server

```
C:\work\stores>ruby script/server

=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2010-07-05 10:40:00] INFO  WEBrick 1.3.1
[2010-07-05 10:40:00] INFO  ruby 1.8.7 (2010-01-10) [i386-mingw32]
[2010-07-05 10:40:00] INFO  WEBrick::HTTPServer#start: pid=1092 port=3000
```

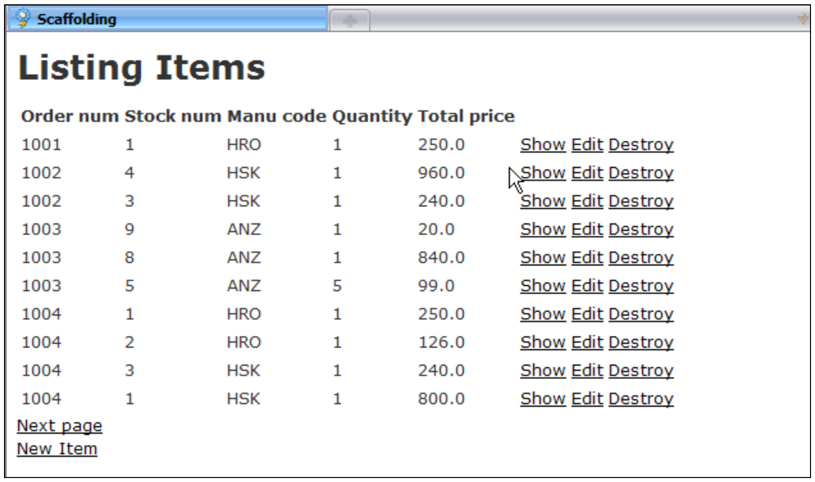
11.4.6 Demonstrating website application

At this point, Rails should have constructed the application for you, and you can browse and change the information for your table.

We can open a web browser and navigate to the local web server to browse the the Order and Item tables:

```
http://127.0.0.1:3000/item
http://127.0.0.1:3000/order
```

Figure 11-1 shows the Listing items page. The page contains links to perform all the typical operations that are associated with a database table (select, insert, update, and delete).



Scaffolding

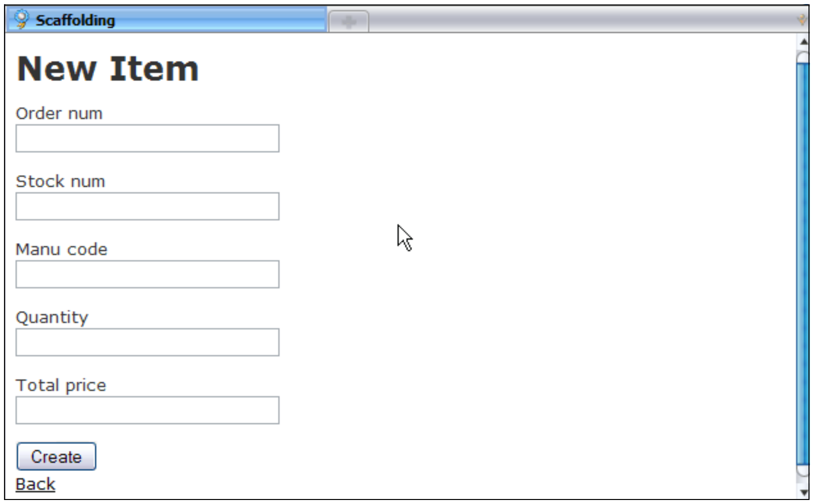
Listing Items

Order num	Stock num	Manu code	Quantity	Total price	
1001	1	HRO	1	250.0	Show Edit Destroy
1002	4	HSK	1	960.0	Show Edit Destroy
1002	3	HSK	1	240.0	Show Edit Destroy
1003	9	ANZ	1	20.0	Show Edit Destroy
1003	8	ANZ	1	840.0	Show Edit Destroy
1003	5	ANZ	5	99.0	Show Edit Destroy
1004	1	HRO	1	250.0	Show Edit Destroy
1004	2	HRO	1	126.0	Show Edit Destroy
1004	3	HSK	1	240.0	Show Edit Destroy
1004	1	HSK	1	800.0	Show Edit Destroy

Next page
New Item

Figure 11-1 Item listing web page

Figure 11-2 shows the New Item page with all the fields from the Item table ready to be used to insert a new record into the table.



Scaffolding

New Item

Order num

Stock num

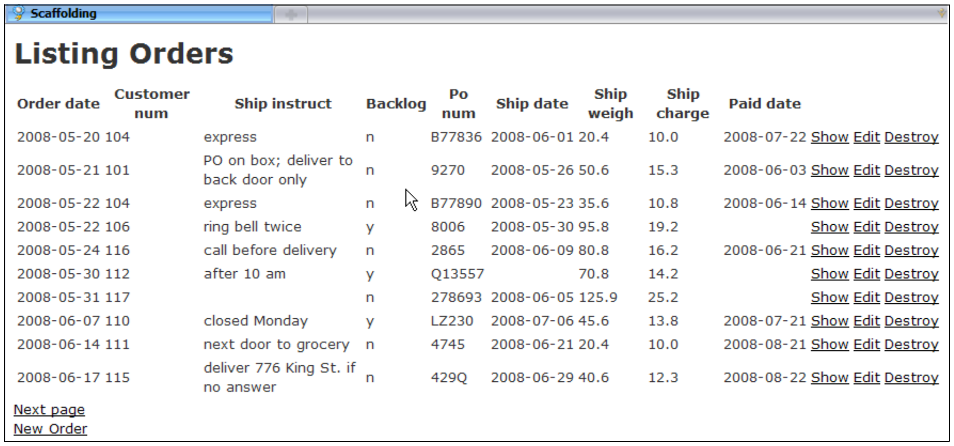
Manu code

Quantity

Total price

Figure 11-2 Item New web page

Figure 11-3 shows the Listing Orders page.



Listing Orders

Order date	Customer num	Ship instruct	Backlog	Po num	Ship date	Ship weigh	Ship charge	Paid date			
2008-05-20	104	express	n	B77836	2008-06-01	20.4	10.0	2008-07-22	Show	Edit	Destroy
2008-05-21	101	PO on box; deliver to back door only	n	9270	2008-05-26	50.6	15.3	2008-06-03	Show	Edit	Destroy
2008-05-22	104	express	n	B77890	2008-05-23	35.6	10.8	2008-06-14	Show	Edit	Destroy
2008-05-22	106	ring bell twice	y	8006	2008-05-30	95.8	19.2		Show	Edit	Destroy
2008-05-24	116	call before delivery	n	2865	2008-06-09	80.8	16.2	2008-06-21	Show	Edit	Destroy
2008-05-30	112	after 10 am	y	Q13557		70.8	14.2		Show	Edit	Destroy
2008-05-31	117		n	278693	2008-06-05	125.9	25.2		Show	Edit	Destroy
2008-06-07	110	closed Monday	y	LZ230	2008-07-06	45.6	13.8	2008-07-21	Show	Edit	Destroy
2008-06-14	111	next door to grocery	n	4745	2008-06-21	20.4	10.0	2008-08-21	Show	Edit	Destroy
2008-06-17	115	deliver 776 King St. if no answer	n	429Q	2008-06-29	40.6	12.3	2008-08-22	Show	Edit	Destroy

Next page
New Order

Figure 11-3 List Order webpage

The web applications development with Ruby on Rails was designed to be an effortless task. With just four commands and a little configuration, we created a website that can handle the common table operations for a database application.

11.5 Using the Rails Adapter with IBM_DB

In this section, we demonstrate how to create a basic web application using the Rails framework with the IBM_DB Adapter. The sample program is a simple telephone directory application for a user to list, add, update, and delete phone entries.

Rails can be used to generate the Data Definition Language (DDL) for the database objects. We show how to create tables with Ruby on Rails.

11.5.1 Creating the Rails application

Use the **rails** command to create the Rails application. We create our application sample in the C:\RailsProjects directory.

Example 11-32 shows how to create a rails application and the **rails** command output of our application.

Example 11-32 Creating a Rails application

```
C:\RailsProjects>rails sample
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
      create  config/initializers
      ...
```

11.5.2 Modifying the database configuration file

Update the database.yml database configuration file with the database connectivity details.

Example 11-33 shows the configuration file for our development database. The name of the adapter, ibm_db, correspond to the Ruby Adapter for IBM Data Servers.

Example 11-33 The database.yml file

```
development:
  adapter: ibm_db
  database: ruby
  username: informix
  password: Ifmx4you
  host: kefka.lenexa.ibm.com
  port: 9089

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:

production:
```

11.5.3 Creating model, control, and view

Ruby on Rails worked on a *Model, Control, View* architecture. You can create model, control, and view components in stages. Here, we show a quick way of using the **scaffold** command to have Rails create the complete application, including all the directories and necessary files.

The telephone table of our application use strings for first name, last name, and phone number. This table layout forms the *model* for our application. We specify this model right on the scaffold command as shown in Example 11-34.

We invoke the command from the root directory of our project with the **ruby script\generate** script. The syntax for the scaffold options is:

```
scaffold <model name> <column name: attributes> .. <column name: attributes>
```

The output shows that Rails creates the necessary models, views and controllers. This command also generates the script to create the tables that are necessary to associate with the model.

Example 11-34 Creating a scaffold

```
C:\RailsProjects\sample>ruby script\generate scaffold phonedir
first_name:string last_name:string phone:string
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/phonedirs
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      create  test/unit/helpers/
      exists  public/stylesheets/
      create  app/views/phonedirs/index.html.erb
...
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/phonedir.rb
      create    test/unit/phonedir_test.rb
      create    test/fixtures/phonedirs.yml
      create    db/migrate
      create    db/migrate/20100704234009_create_phonedirs.rb
```

11.5.4 Migrating the model

Ruby on Rails calls the creation of the database objects backing the model as *migration*. The **scaffold** command created a migrate script for creating the table attached to the model.

Example 11-35 shows the db/migrate/20100704234009_create_phonedirs.rb migration file that is created by the **scaffold** command. This script has both create and drop sections, which means that you can roll back any migration. We created a model by the name phonedir. Ruby on Rails then created the table phonedirs, which is a plural form of the model name, which is the Ruby on Rails naming convention.

Example 11-35 The phonedir migration file

```
class CreatePhonedirs < ActiveRecord::Migration
  def self.up
    create_table :phonedirs do |t|
      t.string :first_name
      t.string :last_name
      t.string :phone

      t.timestamps
    end
  end

  def self.down
    drop_table :phonedirs
  end
end
```

Use the rake utility to migrate the file and create table in the database. The rake utility is a Ruby build script with capabilities similar to the make utility. You can use the rake utility to generate the database schema using a migration file.

Example 11-36 shows how to run the rake script to create the model in the database.

Example 11-36 Creating model

```
C:\RailsProjects\sample>rake db:migrate
(in C:/RailsProjects/sample)
==  CreatePhonedirs: migrating ===============================================
-- create_table(:phonedirs)
   -> 0.0781s
==  CreatePhonedirs: migrated (0.0781s) ======================================
```

Example 11-37 uses the Informix `dbschema` utility to export the table schema to check the tables created Ruby on Rails migration. We started with an empty database and the output shows that two tables were created:

- schema_migrations

  Ruby on Rails uses this table to keep track of the various version of the table, which allows you to roll back to the previous version.

- phonedirs

  In this application table, Ruby added a few columns that we did not specify:

  – id: This serial column is for the primary key required by Ruby on Rails.
  – created_at and updated_at: *Ruby* on Rails uses these optionally.

Example 11-37 The dbschema on database Ruby output

```
% dbschema -d ruby

DBSCHEMA Schema Utility       INFORMIX-SQL Version 11.50.FC7
grant dba to "informix";
...
create table "informix".schema_migrations
  (
    version varchar(255) not null
  );
...

create table "informix".phonedirs
  (
    id serial not null ,
    first_name varchar(255),
    last_name varchar(255),
    phone varchar(255),
    created_at datetime year to fraction(5),
    updated_at datetime year to fraction(5),
    primary key (id)
  );
...
create unique index "informix".unique_schema_migrations on "informix"
    .schema_migrations (version) using btree ;
```

11.5.5 Starting the Rails web server

Example 11-38 shows how to start the Rails WEBrick server from the root directory. The http port number is 3000.

Example 11-38 Starting the WEBrick server

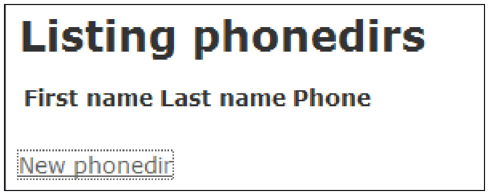
```
C:\RailsProjects\sample>ruby script\server
=> Booting WEBrick
=> Rails 2.3.8 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2010-07-04 17:25:22] INFO  WEBrick 1.3.1
[2010-07-04 17:25:22] INFO  ruby 1.9.1 (2010-01-10) [i386-mingw32]
[2010-07-04 17:25:22] INFO  WEBrick::HTTPServer#start: pid=292 port=3000
```

11.5.6 Checking the application from website

You can start the web application using the local web server address:

`http://localhost:3000/phonedirs`

Figure 11-4 shows the initial screen that our application opened with `http://localhost:3000/phonedirs`.



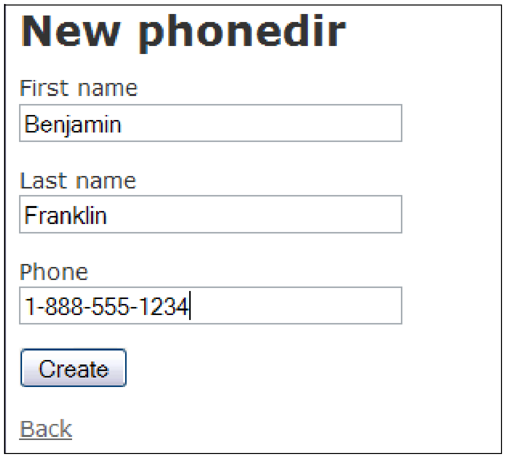
Listing phonedirs

First name Last name Phone

New phonedir

Figure 11-4 Initial screen of the application

We add one phone entry to our directory as shown in Figure 11-5.



New phonedir

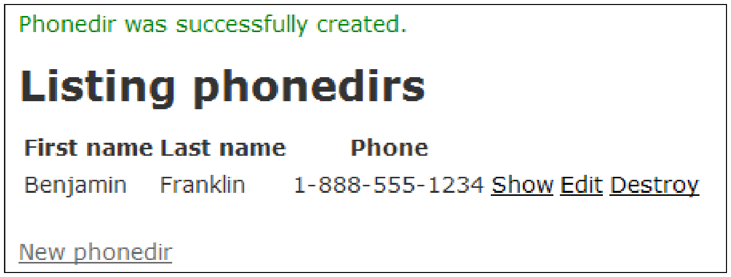
First name
Benjamin

Last name
Franklin

Phone
1-888-555-1234

Figure 11-5 Creating a new phone listing

Figure 11-6 shows that a phone entry is added.



Phonedir was successfully created.

Listing phonedirs

First name	Last name	Phone			
Benjamin	Franklin	1-888-555-1234	Show	Edit	Destroy

Figure 11-6 Listing phonedir after the new entry

For more information regarding the development with Ruby and Ruby on Rails, refer to:

http://rubyonrails.org/documentation

12

Informix 4GL Web Services

In this chapter, we introduce the new Web Services feature of IBM Informix 4GL. This chapter provides an overview and configuring and building Web Services using Informix 4GL.

In this chapter, we discuss the following topics:

- Basic concepts
- Setup and configuration
- Informix 4GL Web Services tools
- Developing a web service with I4GL
- Consuming a web service with I4GL
- Troubleshooting

12.1 Basic concepts

In this section, we provide an introduction of the products and technologies that we discuss in this chapter.

12.1.1 IBM Informix 4GL

Informix 4GL is a programming language developed by IBM for interacting with Informix database servers. It provides a rich environment for easy development of relational database applications. Informix 4GL provides all the component needed to develop character based applications using an Informix database, for example, project management, reports, debugger, and so on.

Informix 4GL supports compilers that can convert the I4GL applications to C language or to generate platform-independent pseudo code that can be executed using a I4GL runner.

For more information about Informix 4GL, refer to the 4GL Reference Manual at:

http://publib.boulder.ibm.com/infocenter/ifxhelp/v0/index.jsp?topic=/com.ibm.tools.doc/4gl.html

12.1.2 Service-oriented architecture and Web Services

Service-oriented architecture (SOA) is an architectural style that provides methods for systems development and integration, allowing applications developed with different technologies or programing languages to exchange data with one another.

This exchange of data is accomplished through the use of Web Services. Web services are saleable functions that can be accessed independent of platforms and programming languages. These functions take a set of inputs and return a set of outputs to accomplish a specific task.

Refer to the *Service Oriented Architecture — SOA* portal for more information:

http://www.ibm.com/software/solutions/soa/

12.1.3 Web Services development

You can create a web service with any web-aware technology. Java is the most common language that is used for Web Services development. However, you can create Web Services with other language and technologies, such as C or .NET

IBM provides the following options for SOA development:

- IBM SOA Sandbox
- IBM Rational® Application Developer
- eKit: Enterprise Architect for SOA

You can develop a web service that requires the use of an IBM Informix database using the following languages:

- Java, using any of the JDBC drivers for an Informix database
- Any .NET language, using any of the Informix .NET providers available
- IBM Informix 4GL

12.1.4 Informix 4GL and Web Services

Starting form version 7.50 of IBM Informix, 4GL developers can manage and create Web Services using the 4GL language.

One of the key benefits of using Informix 4GL is the easy interaction with the Informix database server. The use of SQL statements to access database objects does not require any specific code as with other programming languages, because SQL is embedded in the I4GL language.

The ability to create Web Services directly with I4GL allows the reuse of existing code. Existing solutions that are developed with I4GL can be converted to web solutions without much effort.

With Informix 4GL, you can publish existing I4GL functions as Web Services and use existubg Web Services from any I4GL application.

12.1.5 Components

Informix 4GL uses the Axis2 web service wrapper API to implement the interface that is required to communicate between the web server and the Informix 4GL libraries.

A typical I4GL web service solution includes the following components:

- Apache AXIS2C server
- Web service
- Informix 4GL
- Informix Database Server

A web service can perform the following operations:

- *Create* is the process of creating the web service and publishing it to make it available to consumers.

  The term used in Informix 4GL for this task is *Publish*.
- *Consume* is the process of using the web service, providing input parameters, and retrieving the result as output parameters for the function.

  The term used in Informix 4GL for this task is *Subscribe*.

Axis2 C functions are used for both operations as the wrapper code between the web service and Informix 4GL.

12.2 Setup and configuration

In this section, we discuss the setup and configuration needed to develop Web Services with Informix 4GL.

12.2.1 Prerequisites and supported platforms

The following prerequisites are required to use Web Services with Informix 4GL:

- Apache Axis2/C version 1.5.1 (bundled with Informix 4GL)
- Apache Axis2/Java version 1.3.1 (bundled with Informix 4GL)
- IBM Informix database server version 10 or later
- Java Runtime Environment (JRE) 1.5 or later
- Perl 5.8.8

IBM Informix 4GL 7.50 is supported in the following platforms:

- HP-IA 11.23 and 11.31
- AIX 5.3 and 6.1
- Solaris 5.9 or 5.10
- Red Hat Enterprise Linux 4 and 5
- SUSE Linux Enterprise Server 10

Note: The Web Services feature was added to Informix 4GL version 7.50.xC1 but was available only for Linux platforms. Since version 7.50.xC3, all the platforms listed previously are supported.

12.2.2 Environment

The utilities for using Web Services with Informix 4GL are installed in the same directory as Informix 4GL.

We do not discuss how to install and set up Informix 4GL in this book. For more information, refer to:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.4gl_install.doc/fgl_ing_010.htm`

To use any of the I4GL Web Services tools, you must set the variables listed in Table 12-1 in the development environment. These variables define the location of those 4GL, Java, and Axis2 resources that you need for application development and deployment.

Table 12-1 Environment variables

Variable	Description
AXIS2C_HOME	Specifies the Axis2 installation directory
CLASSPATH	Path to the required Java classes
DBPATH	Path for support files. Must be `$INFORMIXDIR/etc`
INFORMIXDIR	Directory where the 4GL files are installed
INFORMIXSERVER	Default database server
INFORMIXSQLHOSTS	Optional: Specifies the location of the `sqlhosts` file, which contains database connectivity information
JAVA_HOME	Must be set to point to JRE 1.5 or later
Load Library Path for example: LD_LIBRARY_PATH	Specifies which directories to search for client or shared IBM Informix general libraries
PATH	Specifies which directories to search for executable programs. Must include the following path: `$INFORMIXDIR/bin and $JAVA_HOME/bin`
SOA_ERR_LOG	Optional: Specifies the directory where the log file (`w4glerr.log`) is created, defaults to `/tmp`
PROGRAM_DESIGN_DBS	Optional: Database used for storing web service definitions, defaults to `syspgm4gl`

Example 12-1 shows a typical shell script to set up these variables.

Example 12-1 The setup.ksh script

```
AXIS2C_HOME=$INFORMIXDIR/AXIS2C
AXJDIR=$INFORMIXDIR/AXIS2C/AXIS2JARS
CLASSPATH=$AXJDIR/wsdl4j-1.6.2.jar:$AXJDIR/backport-util-concurrent-2.2.jar:$AX
JDIR/XmlSchema.jar:$AXJDIR/XmlSchema-1.3.1.jar:$AXJDIR/xbean-2.2.0.jar:$AXJDIR/
axiom-dom-1.3.1.jar:$AXJDIR/axiom-impl-1.3.1.jar:$AXJDIR/axiom-api-1.3.1.jar:$A
XJDIR/neethi-1.3.1.jar:$AXJDIR/axis.jar:$AXJDIR/commons-logging.jar:$AXJDIR/wsd
l2ws.jar:$AXJDIR/commons-discovery.jar:$AXJDIR/jaxrpc.jar:$AXJDIR/saaj.jar:$AXJ
DIR/wsdl4j.jar:$AXJDIR/axis2-java2wsdl-1.3.1.jar:$AXJDIR/axis2-codegen-1.3.1.ja
r:$AXJDIR/axis2-kernel-1.3.1.jar
DBPATH=$INFORMIXDIR/etc
LD_LIBRARY_PATH=$INFORMIXDIR/AXIS2C/lib:$LD_LIBRARY_PATH
export AXIS2C_HOME AXJDIR CLASSPATH DBPATH LD_LIBRARY_PATH
```

Some of the I4GL Web Services utilities, such as `w4gl`, keep design and configuration information in a database on the Informix database server. The default name for this database is `syspgm4gl`. The database is created the first time the `w4gl` tool is invoked. You can specify your own database name using the environment variable `PROGRAM_DESIGN_DBS`.

12.3 Informix 4GL Web Services tools

This section describes the tools that are available within Informix 4GL to publish, deploy, package, and subscribe Web Services.

12.3.1 The w4glc Web Services compiler

The `w4glc` Web Services compiler is a script based on Perl. The `w4glc` compiler performs all the required task to use Web Services within 4GL, from the creation process to deploying and packaging.

The `w4glc` script is used by other I4GL utilities in a non-interactive way. Any error or failure that is generated during the execution of the script is written to the SOA_ERR_LOG log file.

To execute the `w4glc` utility, use the following syntax:

```
w4glc {-option} <configuration-file>
```

The `w4glc` utility does not keep any information in the database. It uses only configuration files. The configuration file is a text file that specifies the details about the I4GL function that is created as a web service. It contains information such as the location of the 4GL source files, input and output parameters, and database connectivity information.

Table 12-2 lists the parameters that are available with the `w4glc` utility.

Table 12-2 The w4glc utility parameter list

Option	Description
`check`	Reads the configuration file and performs basic checks, such as ensuring that the identified source files exist
`compile`	Compiles generated intermediate code
`deploy`	Deploys the web service on the AXIS2C server
`force`	Overwrites the existing service with identical name
`generate`	Generates the intermediate code for publish/subscribe
`help`	Provides basic help information
`keep`	Retains intermediate source files for troubleshooting
`package`	Bundles a web service for production deployment
`silent`	Generates code without on-screen display
`version`	Prints the version number

Example 12-2 shows the output for the `generate` option.

Example 12-2 The w4glc utility generate output

```
informix@irk:/work$ w4glc -generate ./ws_zipcode_irk.4cf
Begin environment check ...
Environment check is completed.
Generating code. Please wait ...
Generating Wrapper code ....
The wrapper file is /tmp/w4gl_informix/zipcode_details_wrap.c
Generating WSDL ....
Generating headers ....
Generating skeletal code ....
Code generation completed.
Removing /tmp/w4gl_informix/zipcode_details_wrap.c ...
Removing /tmp/w4gl_informix/zipcode_details.wsdl ...
Removing /tmp/w4gl_informix/axis2_skel_ws_zipcode.h ...
Removing /tmp/w4gl_informix/axis2_svc_skel_ws_zipcode.c ...
Removing /tmp/w4gl_informix/services.xml ...
Removing /tmp/w4gl_informix/axis2_skel_ws_zipcode.c ...
informix@irk:/work$
```

For additional information about the I4GL Web Services compiler, refer to:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.4gl_admin.doc/fgl_wsg_500.htm

12.3.2 The w4gl utility

The character-based interface `w4gl` utility is the main tool when using Web Services with Informix 4GL. It allows you to perform the same task as the command line tool, `w4glc`, but it also manages the data within the program design database, `syspgm4gl`.

The design database contains the following information about the Web Services environment:

- Information for the host system, such as the host name or temporary directories
- Location for the Axis2 web server
- Details for the Informix servers to which the Web Services connects
- Definition for Web Services, such as function names or parameters types

The web service definition that is specified through the utility is saved in database tables and is available for future reuse and modification, thus reducing development effort.

The `w4gl` utility uses the same I4GL character-based interface as other 4GL tools to accomplish the creation and consumption of Web Services.

Example 12-3 shows the main menu of the `w4gl` utility.

Example 12-3 The w4gl utility main menu

```
+-----------------------------------------------------------------------------+
|W4GL:   Publish  Subscribe  Host name  App server  Exit                      |
|Create and Deploy web services from I4GL functions.                          |
|--------------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
+----------------------------------------------------------------------------+
```

The `w4gl` utility offers the following options:

- Use the *Publish* option to create a new web service.
- Use the *Subscribe* option to consume a web service.
- Use the *Host name* option to managed the host information that is stored in the design database.
- Use the *Application server* option to managed the Axis2 information that is stored in the design database.

For a detailed description of all the `w4gl` utility menu options, refer to the *4GL Web Services Administration Guide*, which is available at:

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.4gl_admin.doc/fgl_wsg_006.htm

12.3.3 Web Services Description Language Parser (wsdl_parser)

Web Services Description Language (WSDL) is an XML-based language for describing network services. You can use WSDL to describe all the functions and parameters for a web service.

When you subscribe a web service, the `wsdl_parser` tool parses the WSDL file to retrieve all the information that is required to define a web service.

The syntax to invoke the `wsdl_parser` tool is:

```
wsdl_parser sid wsdl_path ws_func i4gl_func target_dir target_file
```

Table 12-3 describes each parameter. All of the parameters are required for the tool to work correctly.

Table 12-3 The wsdl_parser tool parameters list

Parameter	Description
`sid`	An integer that uniquely identifies the subscriber
`wsdl_path`	Location of the file that describes the complete description of the web service; can be an online or a local copy of the WSDL file

Parameter	Description
`ws_func`	Function to consume within the designated web service
`i4gl_func`	Name for the wrapper function to be used by the I4GL program
`target_dir`	The path where files are stored while the web service is being consumed
`target_file`	The file name that contains the generated subscriber client code

Note: The `sid` parameter is used only when you invoke the `wsdl_parser` from the `w4gl` tool. This parameter is ignored if you use `wsdl_parser` from the command line.

Example 12-4 shows the `wsdl_parser` tool used to generate the configuration file for a `ws_zipcode` web service. This example passes the WSDL definition directly from a web server as the following `wsdl_path` parameter:

```
http://irk:9876/axis/services/ws_zipcode?wsdl
```

Example 12-4 The wsdl_parser tool output

```
informix@irk:/work$ wsdl_parser 0 http://irk:9876/axis/services/ws_zipcode?wsdl
zipcode_details zipcode4gl `pwd`/publish zipcode4gl.c
informix@irk:/work$ ls publish
local.wsdl  zipcode4gl.c_zipcode4gl.4cf
informix@irk:/work$
```

12.3.4 I4GL Web Services process

Figure 12-1 illustrates how the I4GL Web Services tools work together.

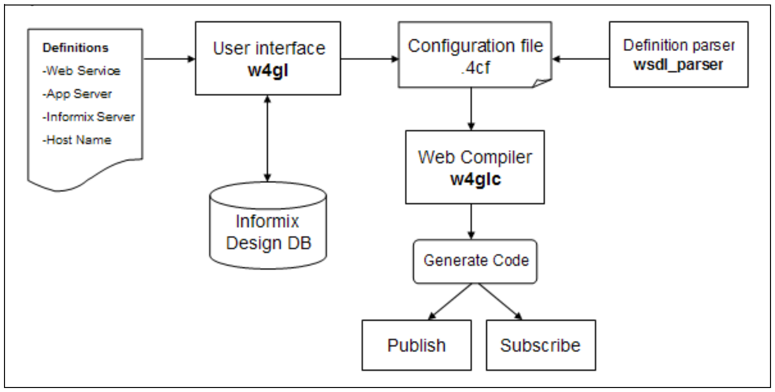


Figure 12-1 I4GL Web Services tools

12.4 Developing a web service with I4GL

In this section, we demonstrate the steps that are required to create a web service using IBM Informix 4GL. Publish and subscribe are two operations to perform with a web service:

- Publish

  The tasks required to create a new web service with the `w4gl` utility are:

  a. Add the Host name and an Axis2 server information.

  b. Add the web service definition details.

  c. Generate the web service configuration file for publishing.

  d. Deploy a web service by registering the service in the Axis2 web server.

  e. Package a web service as a single file (a `.tar` file) that is ready for the production server.

- Subscribe

  The tasks needed to consume a web service are with the `w4gl` tool are:

  a. Add details for the web service to consume.

  b. Compile the Web Services wrapper code that generates the configuration file for subscription.

In the remaining sections, we describe how to publish a simple I4GL function as a web service and show that the function can be used from other languages such as Java.

12.4.1 Example I4GL function

We use a basic I4GL function, `state_name()`, for the web service. This function connects to the database server and retrieves the name for a specific state code.

Example 12-5 shows the 4GL `code state_name()` saved in the `state_name.4gl` file. This function queries the `state` table from the `stores_demo` database. The `state_name()` function takes one input parameter state code of type CHAR(2) and returns the state name that has CHAR(15) data type.

Example 12-5 The state_name.4gl file

```
FUNCTION state_name(code)
   DEFINE state_rec RECORD
          code  CHAR(2),
          sname CHAR(15)
   END RECORD,
   code CHAR(2),
   sel_stmt CHAR(100);

   LET sel_stmt= "SELECT code, sname FROM state WHERE code = ?";

   PREPARE st_id FROM sel_stmt;
   DECLARE cur_id CURSOR FOR st_id;

   OPEN cur_id USING code;
   FETCH cur_id INTO state_rec.*;
   CLOSE cur_id;
   FREE cur_id;
   FREE st_id;
   RETURN state_rec.sname
END FUNCTION
```

To ensure that the 4GL code is correct, we compile the function with the I4GL compiler, `c4gl`, as shown in Example 12-6.

Example 12-6 Compile 4GL function

```
informix@irk:/work$ c4gl -c state_name.4gl
informix@irk:/work$
```

12.4.2 Host and application details

Before adding any of the details for the web service, we must provide details about the system where the web service will run and the Axis2 server that is used. To complete these tasks, we use the “Host name” and “Application server” menus from the `w4gl` utility.

Example 12-7 shows the HOST INFORMATION form with the host name `irk` and the `/tmp/w4gl_informix` temporary directory.

Example 12-7 Host name menu

```
+--------------------------------------------------------------------------+
|HOST NAME:   Query  Next  Previous  Add  Modify  Remove  Exit             |
|See the existing host name details.                                       |
|[1 of 1]-----------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                          |
|                            HOST INFORMATION                              |
|                                                                          |
| Machine ID          [         1]                                         |
| Host Name           [irk                                     ]           |
| Temporary Directory [/tmp/w4gl_informix                              ]   |
|                     [                                                ]   |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
|                                                                          |
+--------------------------------------------------------------------------+
```

Host information includes the following fields:

- `Machine ID` is an automatic identifier number.
- `Host Name` is the system name where the Axis2 server is installed.
- `Temporary Directory` is the directory that is used by the `w4glc` utility for creating temporary files.

After you define the host information, add the information for the Axis2 application server using the APP SERVER menu.

Example 12-8 shows the APP SERVER form.

Example 12-8 Application menu

```
+-----------------------------------------------------------------------------+
|APP SERVER:   Query  Next  Previous  Add  Modify  Remove  Exit               |
|Go to the next app server                                                    |
|[1 of 1]------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                             |
|                          APP SERVER INFORMATION                             |
|  Server ID    [    1]                                                       |
|  Server Name  [axis                                                   ]     |
|  Host Name    [irk                                                    ]     |
|  Port Number  [ 9876]                                                       |
|                           ENVIRONMENT VARIABLES                             |
|  INFORMIXDIR      [/usr3/4gl750                                       ]     |
|  INFORMIXSQLHOSTS [/usr3/sqlhosts                                     ]     |
|  CLIENT_LOCALE    [en_us.utf8        ]                                      |
|  DBDATE           [Y4MD-      ]                                             |
|  Notes            [                                                   ]     |
|                   [                                                   ]     |
|                   [                                                   ]     |
|                   [                                                   ]     |
|                                                                             |
|                                                                             |
|                                                                             |
+-----------------------------------------------------------------------------+
```

An application server is identified by the following fields:

- `Server ID` is an automatic identifier number.
- `Server Name` is the name of the Axis2 server.
- `Host Name` is the system name where the Axis2 server runs.
- `Port Number` is the port used for incoming connections to the web service.

The APP SERVER option is also used to stored specific environment information for the application server. This information is required because the Axis2 server binary loads the I4GL libraries that might need additional resource files located in the `$INFORMIXDIR` directory.

12.4.3 Definition of the web service

In this section, we demonstrate how to define a web service from the `w4gl` utility. To define a web service:

1. Add the IBM Informix database that the web service will use.
2. Add the definition for the web service, such as the name of the service or the name of the 4GL function.

3. Add specific details for the web service such as input and output variables.
4. Add the location of the 4GL source code file.

Add Informix database details

You add Informix database details using the Database menu from the `w4gl` utility.

Example 12-9 shows the DATABASE INFORMATION form with details of the database that we used in our example.

Example 12-9 Database option

```
+-------------------------------------------------------------------------------+
|DATABASE:   Query  Next  Previous  Add  Modify  Remove  Exit                   |
|Go to the previous database record.                                            |
|[1 of 1]------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                               |
|                            DATABASE INFORMATION                               |
|                                                                               |
| Database ID     [          1]                                                 |
| Database Name   [stores7                                              ]      |
| Database Server [irk1150                                              ]      |
| IDS Version     [11.50               ]                                        |
| DB_LOCALE       [en_US.819           ]                                        |
| Notes           [                                                     ]      |
|                 [                                                     ]      |
|                 [                                                     ]      |
|                 [                                                     ]      |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
+-------------------------------------------------------------------------------+
```

This menu option uses the following fields:

- `Database ID` is an automatic number that identifies a database definition.
- `Database name` defines the database name to which the web server connects.
- `IDS Version` is the Informix database server version number.
- `DB_LOCALE` is the locale of the database. The default `DB_LOCALE` is `en_US.819`.

Add details about the web service

You can add details about the web service using the Add option in the web service menu. Example 12-10 shows this option.

Example 12-10 Add the web service menu

```
+------------------------------------------------------------------------------+
|WEB SERVICE:   Query  Next  Previous  Add  Modify  Remove  Install  ...       |
|Specify a new service record.                                                 |
|--------------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                              |
|                                                                              |
|                                                                              |
```

First, you add the web service name and the function name using the Detail menu option. Example 12-11 shows this input form with the details of our web service.

Example 12-11 Web service details

```
+------------------------------------------------------------------------------+
|ADD:   Detail  Variable  File  Exit                                           |
|Specify the web service parameters.                                           |
|--------------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                              |
| Webservice ID    [    4]                                                     |
| Webservice Name  [ws_statename                                            ] |
| Function Name    [state_name                                              ] |
| Notes            [Returns the state name for a given code                 ] |
|                  [                                                        ] |
|                  [                                                        ] |
|                  [                                                        ] |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
+------------------------------------------------------------------------------+
```

The following input details on this form describes a web service:

- `Webservice ID` is an automatically generated identifier for the web service.
- `Webservice Name` is the name of the web service.
- `Function Name` is the he name of the 4GL function.

Input and output variables

Next, you add the input and output variables that the web service uses with the Variable menu option. Example 12-12 shows this menu option. Our function takes an `CHAR(2)` input parameter and returns a `CHAR(15)` value.

Example 12-12 Web service VARIABLE option

```
+------------------------------------------------------------------------------+
|VARIABLE:   Input  Output  Exit                                               |
|Exit the Variable menu.                                                       |
|--------------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                              |
| [ 1] Input parameter - Variable name   Data type                             |
|                                                                              |
| [ 1][code                            ][CHAR(2)                            ] |
| [  ][                                ][                                   ] |
| [  ][                                ][                                   ] |
| [  ][                                ][                                   ] |
| [  ][                                ][                                   ] |
|                                                                              |
| [ 1] Output parameter - Variable name  Data type                             |
|                                                                              |
| [ 1][sname                           ][CHAR(15)                           ] |
| [  ][                                ][                                   ] |
| [  ][                                ][                                   ] |
| [  ][                                ][                                   ] |
| [  ][                                ][                                   ] |
|                                                                              |
|                                                                              |
+------------------------------------------------------------------------------+
```

Source file information

Finally, you define a web service. We stored the location of the I4GL source file that contains our function using the File menu option. Example 12-13 shows the fields used by the File option.

Example 12-13 Web service File option

```
+--------------------------------------------------------------------------------+
|ADD:   Detail  Variable  File  Exit                                             |
|Exit the web services Add menu.                                                 |
|--------------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                                |
| Service Name     [ws_state_name                                           ] |
| Function Name    [state_name                                              ] |
|                                                                                |
| File Number    [ 1]                                                           |
| Directory      [/work                                                     ] |
| File Name      [state_name.4gl                                            ] |
|                                                                                |
| File Number    [  ]                                                           |
| Directory      [                                                          ] |
| File Name      [                                                          ] |
|                                                                                |
| File Number    [  ]                                                           |
| Directory      [                                                          ] |
| File Name      [                                                          ] |
|                                                                                |
|                                                                                |
|                                                                                |
+--------------------------------------------------------------------------------+
```

An I4GL function can require more than one I4Gl file. You must supply all the required file names and their directories. We supplied the 4GL file `state_name.4gl` that contains the `state_name()` function.

12.4.4 Generate the configuration file

With the details of the web service stored in the design database, you can then generate the configuration file (`.4cf`) for a Publish operation. The Generate option is inside the Install menu option.

Example 12-14 shows the Generate form. You must complete every field in the form. If you stored the definition for these items in the design database, you can retrieve them using the Ctrl+B key shortcut.

Example 12-14 Generate form

```
+------------------------------------------------------------------------------+
|INSTALL:   Generate  Deploy  Package  Exit                                    |
|Generate the configuration file for a web service.                            |
|[1 of 1]------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                              |
|                      +---------------------------------------------------+ |
|                      |DATABASES:   Query  Previous query  Exit           | |
| Service ID [   4]   M|Make a new selection                              | |
|                      |---------------------------------------------------| |
| Service Name    [ws_s|                DATABASE INFORMATION               | |
| Host Name       [irk |                                                   | |
| Temp Directory  [/tmp|ID       Database Name                             | |
|                 [    |[   1]  [stores7                                 ]| |
| App-Server Name [axis|[    ]  [                                        ]| |
| Port Number     [  98|[    ]  [                                        ]| |
| Database Name   [    |[    ]  [                                        ]| |
| Database Server [    |[    ]  [                                        ]| |
|                      |[    ]  [                                        ]| |
|                      |                                                   | |
|                      |Arrow key - Press Esc to Accept or press Ctrl+C to Ca| |
|                      +---------------------------------------------------+ |
|                                                                              |
+------------------------------------------------------------------------------+
```

After you enter all the information, use the Generate option to create the configuration file. Example 12-15 shows the output of the Generate option.

Example 12-15 Generate option

```
+------------------------------------------------------------------------------+
|INSTALL:   Generate  Deploy  Package  Exit                                    |
|Generate the configuration file for a web service.                            |
|[1 of 1]------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                              |
|                              GENERATE CONFIGURATION                          |
|                                                                              |
| Service ID [   4]   Machine ID [   1]   Server ID [   1]  Database ID [   1]  |
|                                                                              |
| Service Name    [ws_state_name                                           ]   |
| Host Name       [irk                                                     ]   |
| Temp Directory  [/tmp/w4gl_informix                                      ]   |
|                 [                                                        ]   |
| App-Server Name [axis                                                    ]   |
| Port Number     [  9876]                                                     |
| Database Name   [stores7                                                 ]   |
| Database Server [irk1150                                                 ]   |
|                                                                              |
| Generated configuration file ws_state_name_irk.4cf                           |
+------------------------------------------------------------------------------+
```

The Generate operation creates the configuration file that is required for publishing a web service. The name of the file is constructed using the web service name, the host name, and a `.4cf` extension. In our example, the name of the file is `ws_state_name_irk.4cf`.

Example 12-16 shows the configuration file for the `ws_state_name` web service.

Example 12-16 The ws_state_name_irk.4cf file

```
[SERVICE]
   TYPE = publisher
   INFORMIXDIR = /usr3/4gl750uc3
   DATABASE = stores7
   CLIENT_LOCALE = en_us.utf8
   DB_LOCALE = en_US.819
   INFORMIXSERVER = irk1150
   HOSTNAME = irk
   PORTNO =        9876
   I4GLVERSION = 7.50.xC3
   WSHOME = /usr3/4gl750uc3/AXIS2C
   WSVERSION = axis
   TMPDIR = /tmp/w4gl_informix
   SERVICENAME = ws_state_name
   [FUNCTION]
       NAME = state_name
       [INPUT]
           [VARIABLE] NAME = code TYPE = CHAR(2) [END-VARIABLE]
       [END-INPUT]
       [OUTPUT]
           [VARIABLE] NAME = sname TYPE = CHAR(15) [END-VARIABLE]
       [END-OUTPUT]
   [END-FUNCTION]
   [DIRECTORY]
       NAME = /work
       FILE = state_name.4gl,
   [END-DIRECTORY]
[END-SERVICE]
```

You can use a configuration file (4cf) with the `w4glc` utility to perform tasks, such as generate and compile, directly from the command line. Example 12-17 shows how to perform all the steps that the `w4gl deploy` options performs using the `w4glc` utility.

Example 12-17 Using w4glc

```
informix@irk:/work$ w4glc -generate -compile -deploy ws_state_name_irk.4cf
Begin environment check ...
Environment check is completed.
Generating code. Please wait ...
Generating Wrapper code ....
The wrapper file is /tmp/w4gl_informix/state_name_wrap.c
Generating WSDL ....
Generating headers ....
Generating skeletal code ....
Code generation completed.
Generating shared object for service ws_state_name ....
Compiling code. Please wait...
Executing: c4gl  --shared -o /tmp/w4gl_informix/libws_state_name.so
-I/usr3/4gl750/AXIS2C/include/axis2-1.5.0 -L/usr3/4gl750/AXIS2C/lib
-laxis2_engine -laxutil -laxis2_axiom /tmp/w4gl_informix/state_name_wrap.c
/tmp/w4gl_informix/axis2_skel_ws_state_name.c
/tmp/w4gl_informix/axis2_svc_skel_ws_state_name.c /work/state_name.4gl
/usr3/4gl750/lib/tools/w4glutil.a /usr3/4gl750/lib/tools/lib4gl.a
Compilation done.
Deploying service ws_state_name ...
Deploying the service. Please wait ...
Copying /tmp/w4gl_informix/libws_state_name.so ...
Copying /tmp/w4gl_informix/services.xml ...
Copying /tmp/w4gl_informix/state_name.wsdl ...
Service name: ws_state_name
The wrapper file is /tmp/w4gl_informix/state_name_wrap.c
Generating WSDL ....
Generating headers ....
Generating skeletal code ....
Code generation completed.
Generating shared object for service ws_state_name ....
Compiling code. Please wait...
Executing: c4gl  --shared -o /tmp/w4gl_informix/libws_state_name.so
-I/usr3/4gl750/AXIS2C/include/axis2-1.5.0 -L/usr3/4gl750/AXIS2C/lib
-laxis2_engine -laxutil -laxis2_axiom /tmp/w4gl_informix/state_name_wrap.c
/tmp/w4gl_informix/axis2_skel_ws_state_name.c
/tmp/w4gl_informix/axis2_svc_skel_ws_state_name.c /work/state_name.4gl
/usr3/4gl750/lib/tools/w4glutil.a /usr3/4gl750/lib/tools/lib4gl.a
Compilation done.
Deploying service ws_state_name ...
Deploying the service. Please wait ...
Copying /tmp/w4gl_informix/libws_state_name.so ...
Copying /tmp/w4gl_informix/services.xml ...
Copying /tmp/w4gl_informix/state_name.wsdl ...
Service name: ws_state_name
Deployed at : /usr3/4gl750/AXIS2C/services/ws_state_name

Removing /tmp/w4gl_informix/state_name_wrap.c ...
Removing /tmp/w4gl_informix/state_name.wsdl ...
Removing /tmp/w4gl_informix/axis2_skel_ws_state_name.h ...
Removing /tmp/w4gl_informix/axis2_svc_skel_ws_state_name.c ...
Removing /tmp/w4gl_informix/services.xml ...
Removing /tmp/w4gl_informix/axis2_skel_ws_state_name.c ...
Removing /tmp/w4gl_informix/libws_state_name.so ...
```

The output of the command line tool shows all the steps completed while creating and compiling the wrapper function. When errors occur during the deployment process, running each individual step with the w4glc utility might help to diagnose the reason for the problem.

12.4.5 Deployment of the web service

This process generates the auxiliary code that is required to link the Axis2 application server with the I4GL function. During deployment, the C code for the web service is created and compiled automatically using the 4GL libraries. The result of this process is a shared library ready that can be used in the application server.

This process also creates a WSDL file for the web service and copies this file, together with the web service shared library, into the application server directory.

The Deploy option is part of the Install menu. Example 12-18 shows the deployment of the ws_state_name web service.

Example 12-18 Deploy form

```
+-------------------------------------------------------------------------------+
|INSTALL:   Generate  Deploy  Package  Exit                                     |
|Deploy the web service.                                                        |
|[1 of 1]------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                               |
|                          CONFIGURATION TO DEPLOY                              |
|                                                                               |
| File Name [ws_state_name_irk.4cf                             ]             |
|           [                                                    ]             |
|           [                                                    ]             |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
| Deployed ws_state_name.                                                       |
+-------------------------------------------------------------------------------+
```

After a successful deployment, the web service is ready to be consume by any application. The files are copied into the Axis2 services directory automatically.

Example 12-19 shows the web service files on the application server.

Example 12-19 Application Server services directory

```
informix@irk:/work$ ls $INFORMIXDIR/AX*/services/ws_state_name
libws_state_name.so  services.xml  state_name.wsdl
informix@irk:/work$
```

12.4.6 Packaging of the web service

Packaging a web service is the process of creating a compressed file with all the components that are required by the web service. You can use the compressed file, in `.tar` format, on other Axis2 servers. Example 12-20 shows the Package menu option.

Example 12-20 Package option

```
+-----------------------------------------------------------------------------+
|INSTALL:   Generate  Deploy  Package  Exit                                   |
|Package the web service.                                                     |
|[1 of 1]------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                             |
|                              SERVICE PACKAGING                              |
|                                                                             |
| File Name [ws_state_name_irk.4cf                                        ] |
|           [                                                              ] |
|           [                                                              ] |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
| Packaging successful. Check $TMPDIR directory (in config file) for .tar file |
+-----------------------------------------------------------------------------+
```

The compressed file is left in the `$TMPDIR` directory. Example 12-21 shows the contents of the packaged file for the `ws_state_name` web service.

Example 12-21 Packaged .tar file

```
informix@irk:/work$ tar tvf  /tmp/w4gl_informix/ws_state_name.tar
drwxr-xr-x informix/informix 2010-07-05 21:01 ws_state_name/
-rw-r--r-- informix/informix 2010-07-05 21:01 ws_state_name/services.xml
-rw-r--r-- informix/informix 2010-07-05 21:01 ws_state_name/state_name.wsdl
-rwxr-xr-x informix/informix 2010-07-05 21:01 ws_state_name/libws_state_name.so
informix@irk:/work$
```

12.4.7 Starting the Axis2 application server

An Axis2 application server is included with the Informix 4GL package. It is located in the `AXIS2C` directory under the `$INFORMIXDIR` directory.

The Axis2 application server must be started to consume a web service. It is not required during the development or deploy process.

Example 12-22 demonstrates how to start the Axis2 application server.

Example 12-22 Starting Axis2

```
informix@irk:/usr3/4gl750/AXIS2C/bin$ ./axis2_http_server -p 9876
Started Simple Axis2 HTTP Server ...
```

For more information about the Axis2 application server, refer to:

http://ws.apache.org/axis2/c/docs/axis2c_manual.html

12.4.8 Consuming the I4GL web service

The main feature of a web service is that it can be used by any application in any platform. A web service is not tied to the technology or programming language that is used to develop the service. Any application that supports SOAP can make use of the functions that are implemented inside a web service. SOAP is a simple XML-based protocol that is used to exchange information over an HTTP link.

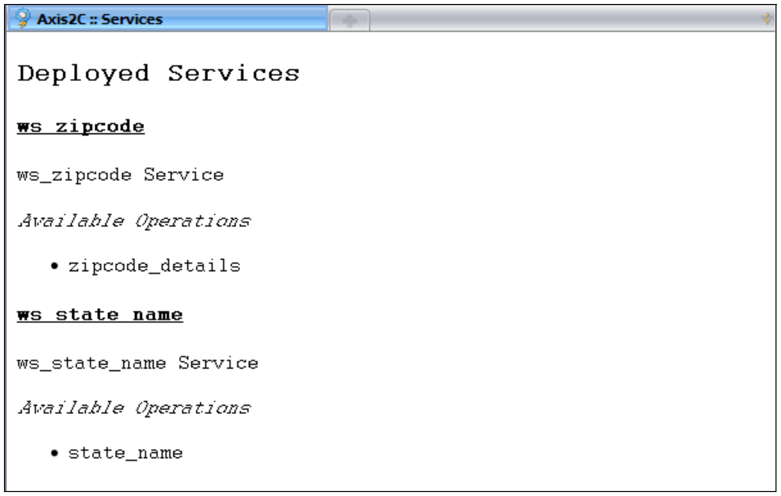
In this section, we demonstrate how to use the `ws_state_name` web service using a basic Java application.

Getting a list of available Web Services

You can get a list of available Web Services in the Axis2 application server by opening a web browser and connecting to the Axis2 server:

```
http://hostname:port/axis/services
```

Figure 12-2 shows the Web Services that are available in our example.



Deployed Services

ws_zipcode

ws_zipcode Service

Available Operations

- zipcode_details

ws_state_name

ws_state_name Service

Available Operations

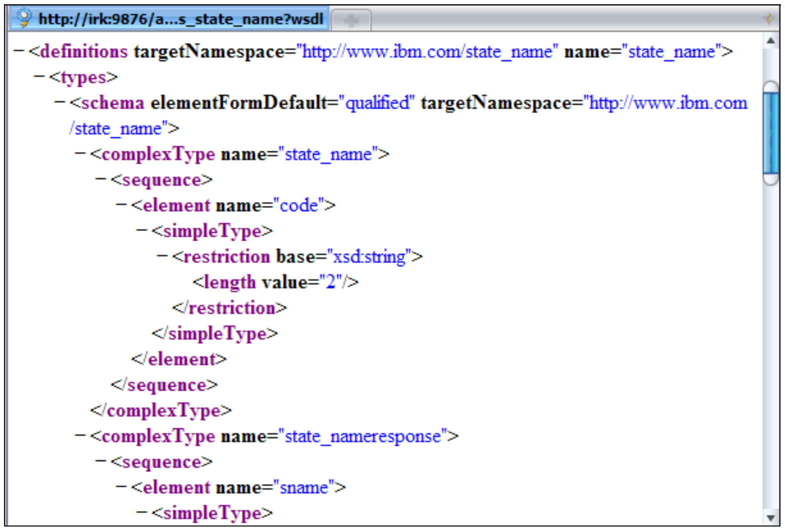
- state_name

Figure 12-2 Axis2 Web Services

You can use the `?wsdl` keyword to retrieve the WSDL file as follows:

```
http://hostname:port/axis/services/ws_state_name?wsdl
```

Figure 12-3 shows the WSDL information for the `ws_state_name` service.



```
http://irk:9876/a...s_state_name?wsdl
- <definitions targetNamespace="http://www.ibm.com/state_name" name="state_name">
  - <types>
    - <schema elementFormDefault="qualified" targetNamespace="http://www.ibm.com
      /state_name">
      - <complexType name="state_name">
        - <sequence>
          - <element name="code">
            - <simpleType>
              - <restriction base="xsd:string">
                  <length value="2"/>
                </restriction>
              </simpleType>
            </element>
          </sequence>
        </complexType>
      - <complexType name="state_nameresponse">
        - <sequence>
          - <element name="sname">
            - <simpleType>
```

Figure 12-3 WSDL information for the ws_state_name service

Java application

The main advantages of Web Services is that the application that uses the services does not need to know anything about how the service is implemented. The information needed for consuming a web service is defined in the WSDL file, such as the name of the operations that it supports and the parameters that it requires.

Example 12-23 demonstrates how to use the `ws_state_name` I4GL web service from a simple Java application.

Example 12-23 The state_name.java file

```
informix@irk:/work$ cat state_name.java
 import org.apache.axis.client.Call;
 import org.apache.axis.client.Service;
 import javax.xml.namespace.QName;

 public class state_name {
   public static void main(String [] args) {

   try {
     String endpoint = "http://irk:9876/axis/services/ws_state_name";
     String qname = "http://www.ibm.com/ws_state_name";

     Service  service = new Service();
     Call     call    = (Call) service.createCall();
```

```
      call.setTargetEndpointAddress( new java.net.URL(endpoint) );
      call.setOperationName(new QName(qname, "state_name"));
      String ret = (String) call.invoke( new Object[] { args[0] } );

      System.out.println("Sent 'CA', got '" + ret + "'");
      } catch (Exception e) {
         System.err.println(e.toString());
       }
     }
 }


informix@irk:/work$ javac state_name.java
informix@irk:/work$ java state_name CA
Sent 'CA', got 'California     '
informix@irk:/work$ java state_name AR
Sent 'CA', got 'Arkansas       '
informix@irk:/work$
```

12.5 Consuming a web service with I4GL

In this section, we demonstrate how to consume a web service using IBM Informix 4GL.

12.5.1 Web service to consume

You can use the `w4gl` utility to insert details about the specific web service that you want to consume. The Subscribe menu option allows you to manage definitions for Web Services in the design database.

Example 12-24 shows the Subscribe form with the details referring to the `ws_state_name` web service.

Example 12-24 Subscribe form

```
+----------------------------------------------------------------------------+
|SUBSCRIBE:   Query  Next  Previous  Add  Modify  Remove  Compile  Exit      |
|Specify a new web service definition.                                       |
|--------------------------------[ irkpgm4gl ]--------------------[Help: CTRL-W]|
|                                                                            |
| Subscription ID    [      3]                                               |
| WSDL Path          [http://irk:9876/axis/services/ws_state_name?wsdl      ] |
|                    [                                                      ] |
| Webservice Function[state_name                                            ] |
| I4GL Function      [statename                                             ] |
| Target Directory   [/work/publish                                         ] |
| Target File Name   [statename                                             ] |
| Notes              [Wrapper for ws_state_name Web Service                 ] |
|                    [                                                      ] |
|                    [                                                      ] |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
|                                                                             |
+-----------------------------------------------------------------------------+
```

This form includes the following details:

- `Subscription ID` is an automatic generated identifier of the web service.
- `WSDL Path` is the complete path of the WSDL file for the service. This path can be a local file or a URL.
- `Web service Function` is the function name that is provided by the web service.
- `I4GL Function` is the name of the wrapper C function. This name is the function that our I4GL used to invoke the web service operation.
- `Target Directory` is the directory where the wrapper and configuration files are created.
- `Target File Name` is the name of the C source file with the I4GL function.

The configuration file for a publish operation is generated automatically by the `w4gl` utility, but you can also create it using the `wsdl_parser` as shown in Example 12-25.

Example 12-25 Generating the configuration file

```
informix@irk:/work$ wsdl_parser 0
http://irk:9876/axis/services/ws_state_name?wsdl state_name wsstatecode
`pwd`/publish statecode.c
informix@irk:/work$ ls publish
local.wsdl  statecode.c_wsstatecode.4cf
informix@irk:/work$ cat publish/statecode.c_wsstatecode.4cf
[SERVICE]
  TYPE = subscriber
  I4GLVERSION = 7.50.xC4
  WSHOME = 0
  TARGET_DIR = /work/publish
  I4GL_FUNCTION = wsstatecode
  TARGET_FILE = statecode.c
  [WSDL_INFO]
```

```
    WSDL_PATH = /work/publish/local.wsdl
    WSDL_NAME_SPACE = http://www.ibm.com/state_name
    [FUNCTION]
      SERVICENAME = ws_state_name
      NAME = state_name
      [INPUT]
          [VARIABLE] NAME = code TYPE = char(2) [END-VARIABLE]
      [END-INPUT]
      [OUTPUT]
          [VARIABLE] NAME = sname TYPE = char(15) [END-VARIABLE]
      [END-OUTPUT]
    [END-FUNCTION]
  [END-WSDL_INFO]
[END-SERVICE]
```

12.5.2 Compiling the wrapper code

Next, you need to generate and compile the wrapper code using the Compile menu option of the `w4gl` utility. Example 12-26 shows the `w4gl` utility after the wrapper code is compiled for the `ws_state_name` web service.

Example 12-26 Compile option

```
+-------------------------------------------------------------------------------+
|SUBSCRIBE:   Query  Next  Previous  Add  Modify  Remove  Compile  Exit         |
|Compile the subscriber client code.                                            |
|--------------------------------[ irkpgm4gl ]-------------------[Help: CTRL-W]|
|                                                                               |
| Subscription ID    [      3]                                                  |
| WSDL Path          [http://irk:9876/axis/services/ws_state_name?wsdl      ] |
|                    [                                                      ] |
| Webservice Function[state_name                                            ] |
| I4GL Function      [statename                                             ] |
| Target Directory   [/work/publish                                         ] |
| Target File Name   [statename                                             ] |
| Notes              [Wrapper for ws_state_name Web Service                 ] |
| +-----------------------------------------------------------------------------+|
| |                                                                           ||
| |Subscriber code has been compiled successfully.                            ||
| +-----------------------------------------------------------------------------+|
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
+-------------------------------------------------------------------------------+
```

After this process is complete, the C files for the wrapper function, including the object file that contains the function code, are left in the Target Directory that is specified in the web service definition.

Example 12-27 shows the files for the `ws_state_name` web service. You can use these files from an Informix 4GL program to consume the web service.

Example 12-27 Web service files

```
informix@irk:/work/publish$ ls
axis2_stub_ws_state_name_statename.h  statename.c_statename.4cf
statename.c                           statename.o
informix@irk:/work/publish$
```

12.5.3 Using the web service from an I4GL application

To consume a web service from I4GL, you have to link the object file for the wrapper function in the 4GL code or use the C source file that is generated during the subscribe compile process.

Example 12-28 shows a simple 4GL program that uses the wrapper function for the `ws_state_name` web service. The wrapper function can be called like any other C function in I4GL.

Example 12-28 The I4GL subscriber code: wsstate.4gl

```
informix@irk:/work/publish$ cat wsstate.4gl
MAIN
   DEFINE sname CHAR(15)
   WHENEVER ERROR STOP

   CALL statename("CA") RETURNING sname
   DISPLAY "State name: ",sname

END MAIN
informix@irk:/work/publish$ c4gl wsstate.4gl statename.c -o wsstate
$INFORMIXDIR/lib/tools/w4glutil.a -I$AXIS2C_HOME/include/axis2-1.5.0
-L$AXIS2C_HOME/lib -laxis2_engine
informix@irk:/work/publish$ ./wsstate
State name: California
informix@irk:/work/publish$
```

For more information about Informix 4GL, refer to the 4GL Reference Manual at:

http://publib.boulder.ibm.com/infocenter/ifxhelp/v0/index.jsp?topic=/com.ibm.tools.doc/4gl.html

12.6 Troubleshooting

In this section, we discuss typical problems that can occur when developing a web service using the Informix 4GL Web Services tools and how to obtain diagnostic information through the use of tracing or log files.

12.6.1 Typical problems

This section lists typical problems that can occur.

Connection

Connection errors normally occur when there is something wrong in the configuration details for the development environment or the web service.

Any web service developed in I4GL requires the use of the Informix 4GL communication libraries for the database connection. I4GL tools such as `w4gl` keep details about the environment in the Design database. Therefore, it is critical that the communication with the IBM Informix database was configured correctly before beginning any project.

Environment variables such as INFORMIXDIR and INFORMIXSQLHOSTS must contain valid details for the Informix database server to use during development.

Also, consider the following problems that are related to connection:

- Confirm Informix database server details. Informix Web Services tools and Web Services running on the Axis2 server use the database information that is defined in the Design database.
- Verify connection information. Connection information regarding the Axis2 server, such as the host name or port number, is stored as part of an Application Server definition. Check that these values are valid.
- Consuming Web Services with a 4GL application requires a TCP connection from the I4GL program to the application server that is running the web service. Make sure that the location of the web service is correct and that the application server can be reached from the I4GL process.

Compilation errors

A common reason for having compilation errors is due to an incorrect setup or incorrect use of the Informix 4GL Web Services utilities.

Environment variables such as `PATH` and `LD_LIBRARY_PATH` (or the suitable variable for the platform) might cause compilation and runtime errors when developing a web service with Informix 4GL. When a compile problem occurs while using one of the I4GL web service tools, the information about the error is written into the W4GL log file. The default name and location of this file is `/tmp/w4glerr.log`.

Example 12-29 shows an error message that is generated by the `w4gl` utility during the deploy process.

Example 12-29 A w4gl error.

```
The file "state_name.err" has been written.
-CDCAK0012: The web service not deployed. Check error log '/tmp/w4glerr.log'
```

The Deploy menu option performs several operations automatically, including generating the wrapper code, compiling, and moving the Web Services files into the Axis2 server. To know the specific task that is failing, use the web compiler utility (`w4glc`) to perform each individual task manually.

By default, the `w4glc` script deletes the temporary files that are used to perform a task. Use the `-keep` flag to avoid file deletion.

Example 12-30 shows how to run the generate process using the `w4glc` utility.

Example 12-30 Using w4glc generate to avoid file deletion

```
informix@irk:/work$ w4glc -generate -keep  ws_state_name_irk.4cf
Begin environment check ...
Environment check is completed.
Generating code. Please wait ...
Generating Wrapper code ....
The wrapper file is /tmp/w4gl_informix/state_name_wrap.c
Generating WSDL ....
Generating headers ....
Generating skeletal code ....
Code generation completed.
```

With the wrapper files in the temporary directory, you can execute the compile process from the command line using the `w4glc` utility and examine the output for any errors. Example 12-31 shows the output of the compile process when an compilation error occurs.

Example 12-31 Compilation output

```
informix@irk:/work$ w4glc -compile -keep  ws_state_name_irk.4cf
Begin environment check ...
Environment check is completed.
Generating shared object for service ws_state_name ...
Compiling code. Please wait...
```

```
Executing: c4gl -keep --shared -o /tmp/w4gl_informix/libws_state_name.so
-I/usr3/4gl750/AXIS2C/include/axis2-1.5.0 -L/usr3/4gl750/AXIS2C/lib
-laxis2_engine -laxutil -laxis2_axiom...
The compilation was not successful.  Errors found: 1.
The file "state_name.err" has been written.
Error executing: c4gl -keep --shared -o /tmp/w4gl_informix/libws_state_name.so
-I/usr3/4gl750/AXIS2C/include/axis2-1.5.0 ...
/usr3/4gl750/lib/tools/w4glutil.a /usr3/4gl750/lib/tools/lib4gl.a at
/usr3/4gl750/lib/globals.pm line 583.
Error executing: c4gl -keep --shared -o /tmp/w4gl_informix/libws_state_name.so
-I/usr3/4gl750/AXIS2C/include/axis2-1.5.0 -L/usr3/4gl750/AXIS2C/lib
-laxis2_engine -laxutil ...
informix@irk:/work$
```

Similar to any I4GL program, if the error was inside the Informix 4GL code, a `.err` file is created that contains the error message. Example 12-32 shows the content of the `state_name.err` file.

Example 12-32   The state_name.err file

```
informix@irk:/work$ cat state_name.err
FUNCTION state_name(code)
   DEFINE state_rec RECORD
          code  CHAR(2),
          sname CHAR(15)
   END RECORD,
   code CHAR(2),
   sel_stmt CHAR(100);

   LET sel_stmt= "SELECT code, sname FROM state WHERE code = ?";

   PEPARE st_id FROM sel_stmt;
|________^
|
|      A grammatical error has been found on line 11, character 10.
| The construct is not understandable in its context.
| See error number -4373.
   DECLARE cur_id CURSOR FOR st_id;

   OPEN cur_id USING code;
   FETCH cur_id INTO state_rec.*;
...
```

Keeping the wrapper C files in the temporary directory might be useful when the compilation error appears inside the Axis2 functions. Example 12-33 shows the temporary files that are created with the generate option for the `ws_state_name` web service.

Example 12-33 Temporary files

```
informix@irk:/work$ ls /tmp/w4gl_informix/
axis2_skel_ws_state_name.c      services.xml      tmpXMLReqFile
axis2_skel_ws_state_name.h      state_name_wrap.c  ws_state_name.tar
axis2_svc_skel_ws_state_name.c  state_name.wsdl
informix@irk:/work$
```

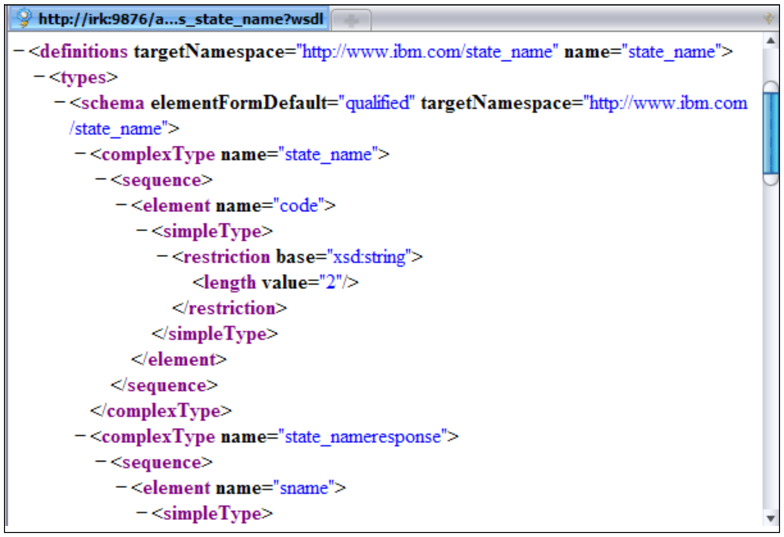
While compiling any web service, during creation or consume, make sure all the required libraries and include files are passed to the compiler correctly. These requirements can vary from platform to platform. Always check the release notes.

The release and documentation files for Informix 4GL are located in the `release/en_us/0333` directory inside the INFORMIXDIR variable. These files contain additional information that is relevant to the version of I4GL that is installed, such as known issues or compiler requirements, that might help diagnose the problem.

Consuming the web service

Errors when consuming the web service might be caused by an invalid web service definition. Items such as the function name or parameters type are defined in the configuration file and are used to create the WSDL file that the application server uses to define the web service. You can retrieve the WSDL information for a web service using the location of the service and the `?wsdl` suffix from any web browser.

Figure 12-4 shows the WSDL information for the `ws_state_name` web service.



```
- <definitions targetNamespace="http://www.ibm.com/state_name" name="state_name">
  - <types>
    - <schema elementFormDefault="qualified" targetNamespace="http://www.ibm.com
      /state_name">
      - <complexType name="state_name">
        - <sequence>
          - <element name="code">
            - <simpleType>
              - <restriction base="xsd:string">
                  <length value="2"/>
                </restriction>
              </simpleType>
            </element>
          </sequence>
        </complexType>
      - <complexType name="state_nameresponse">
        - <sequence>
          - <element name="sname">
            - <simpleType>
```

Figure 12-4 WSDL for ws_state_name service

Make sure the operations, parameters, and data types are correct for the I4GL code that the function uses.

Testing a web service

Informix I4GL does not provide any specific tool for testing Web Services. Although you can create a simple 4GL code to consume the web service, depending on the complexity of the web service, it can be more useful to perform a complete testing.

You can use open source tools, such as soapUI, to test your web service before publish it in the production environment. For more information about soapUI, refer to:

`http://www.soapui.org/`

12.6.2 Tracing

A developer can use one of the following types of tracing when diagnosing an Informix 4GL web service problem:

- Application server trace can be used to diagnose problems between a client application and the Axis2 application server.
- Database trace can be used when diagnostic problems are specific to operations with the database server, such as SQL errors or incorrect data returned.

Application server trace

The Axis2c application server provides a method to trace all the exchanged messages between the application server and a client that is consuming a web service.

You can set this trace using the **`-l log_level`** option when you start the Axis2 server. To launch the Axis2 server with trace enable, run the following command:

```
axis2_http_server -p 9876 -l 6 -f /tmp/w4gl_informix/axis_trace.log
```

Use the **`-l`** to specify the logging level for the application server. The maximum value is `6`, which enables full tracing.

You can specify the location of the trace file using the **`-f log_file`** option. Example 12-34 shows the contents of a typical trace file.

Example 12-34 A typical application_trace sample

```
[debug] phase_resolver.c(139) Service name is : ws_zipcode
[debug] phase_resolver.c(1123) Operation name is : zipcode_details
[debug] phase_holder.c(139) Add handler AddressingOutHandler to phase
[debug] phase_holder.c(139) Add handler AddressingOutHandler to phase
[debug] phase_resolver.c(222) svc name is:ws_state_name
[debug] phase_resolver.c(139) Service name is : ws_state_name
[debug] phase_resolver.c(1123) Operation name is : state_name
[debug] phase_holder.c(139) Add handler AddressingOutHandler to phase
...
```

For more information about the logging options with the Axis2c application server, refer to:

http://ws.apache.org/axis2/c/docs/axis2c_manual.html#simple_axis_server

Database trace

IBM Informix 4GL uses the SQLI protocol to exchange data with the Informix server. Thus, any web service that is developed using Informix 4GL also uses the SQLI protocol for any communication with the database server.

You can use the SQLIDEBUG environment variable to collect all the messages between the application server and the Informix database server.

To enable this trace, create the SQLIDEBUG environment variable before starting the Axis2 Application server.

Example 12-35 demonstrates how to set the SQLIDEBUG variable and how to run `sqliprint` to un-encode the SQLI file.

Example 12-35 The SQLIDEBUG variable client side

```
informix@irk:/usr3/4gl750/AXIS2C/bin$ export SQLIDEBUG=2:/tmp/sqlitrace
informix@irk:/usr3/4gl750/AXIS2C/bin$ ./axis2_http_server -p 9876
Started Simple Axis2 HTTP Server ...

...
...
informix@irk:/work$ ls /tmp/sqlitrace*
/tmp/sqlitrace_17008_0_8c819d0

informix@irk:/works$ sqliprint -o tracefile.txt /tmp/sqlitrace_17008_0_8c819d0

informix@irk:/works$
```

Note: The `sqliprint` tool is included with Informix Client Software Development Kit (Client SDK).

You can also use the SQLIDEBUG trace at the server side. For more information, refer to 3.3.6, “Troubleshooting” on page 117.

13

Application development considerations

In this chapter, we examine some of the considerations a developer might need to address in a multi-user environment. A single user workstation that connects to an exclusive-use database does not the issue of two or more independent uses of the same data at the same time. However, in a multi-user environment, concurrency is a challenge.

IBM Informix database servers are designed to provide features to help in handling concurrency and sorting facilities. The application developer should design applications to take advantage of these built-in features, rather than attempt to implement their own facilities in the application. In this chapter we examine the factors that cause concurrency problems, and focus on ways to keep the scope and duration of locks to a minimum. To do this, we consider isolation levels, sharing data, and data contention issues that can occur when two or more attempts are made to access or change the same row of data.

In the last two sections, we focus on configuration parameters that effect the application development, and how to monitor issues when the application developers work with the database administrator to tune the engine and the application to work effectively.

13.1 Concurrency and locking

Concurrency involves two or more independent uses of the same data at the same time. In a database system with many users, each user needs to be able to access and modify data. Unless the developer and database system impose controls, there can be negative consequences. Programs might access old data that is in the process of being changed by another user, and changes might seem to disappear even though it seems like the change was performed successfully.

To take advantage of database server controls, tables in a multiple user environment should be logging tables. At a minimum, if you must use a nonlogging table within a transaction, either set Repeatable Read isolation level or lock the table in exclusive mode.

To avoid concurrency problems, the database server imposes a system of *locks*. A lock is a claim, or reservation, that a program can place on a piece of data. The database server assures that no other program can modify it, as long as the lock is in place. When another application requests the data, the database server either makes the program wait or turns it back with an error.

The application developer can control the effect of lock access using a combination of SQL statements and with the buffering mode selected for the database. The most used SQL statements are SET LOCK MODE, SET ISOLATION, and SET TRANSACTION. We discuss these in more detail later. For now, we need a better understanding of the types of locks that we can encounter.

IBM Informix offers several server edition. IBM Informix Extended Parallel Server, Informix Online, and Standard Engine will have different syntax for concurrency related commands. You can find more information about the command syntax for the discussion in this chapter in *IBM Informix Guide to SQL: Syntax, v11.50*, SC27-3611.

13.1.1 Types of locks

A lock is implemented as a variable associated with a data item. It can be explicitly placed by an application or, more frequently, is implicitly handled by the database management system. The lock is used to mark a data item as reserved; the type of lock designation determines what actions are permitted by users in regard to the data item.

An IBM Informix instance can have several types of locks:

- A *shared lock* reserves its object for reading only. It prevents the object from changing while the lock remains. More than one program can place a shared lock on the same object. More than one object can read the record while it is locked in shared mode. In the lock list output visible from **`onstat -k`**, a thread with a shared lock is be designated with an *S*. If an object is currently locked in exclusive mode and the user thread wants to acquire a shared lock, the designation is "IS" (intent-shared).

  When a session first connects to a database, IBM Informix Servers place a shared lock on the database, to prevent another session from acquiring an exclusive lock on the same database. SELECT queries place a shared lock at the table level, because it is faster for the engine to find a table level lock than it is to search through potentially thousands of row locks.

- *Intent-exclusive locks* are set automatically by Informix. If a row in a table is updated, an exclusive lock is placed on the row and an intent-exclusive lock is placed on the table. This assures that no other session can place a shared or exclusive lock on the table as long as an individual row is locked exclusively. In the lock list output visible from **`onstat -k`**, a thread with an intent exclusive lock is designated with an *IX*. A related designation that is sometimes visible is *SIX*. This designation indicates the object is currently shared, with Intent-exclusive when the chance arrives.

- An *exclusive lock* reserves its object for the use of a single application. This lock type is used when the application needs to change the object. You cannot place an exclusive lock where any other kind of lock exists. After you place an exclusive lock, you cannot place another lock on the same object. In the lock list output visible from **`onstat -k`**, a thread with an exclusive lock is designated with an *X*.

- A *promotable* (or *update*) lock establishes an intent to update. You can only place it where no other updatable or exclusive lock exists. You can place an updatable lock on records that already have shared locks. When the application is about to change the locked object, you can promote the update lock to an exclusive lock, but only if no other locks, including shared locks, are on the record at the time the lock would change from update to exclusive. If a shared lock was on the record when the update lock was set, you must drop the shared lock before the update lock can be promoted to an exclusive lock. In the lock list output visible from **`onstat -k`**, a thread with an update lock is designated with an *U*.

13.1.2 Lock duration

The length of time a lock remains in effect is known as *lock duration.* The duration of a lock is determined by the application, the closing of a database, and the type of transaction method used by the application and database

If the database does not use transactions (no transaction log exists and you do not use a COMMIT WORK statement), an explicit table lock remains until it is removed by the execution of the UNLOCK TABLE statement.

If database transactions are in use, the end-of-lock-duration event occurs when the transaction ends or a COMMIT WORK is issued in the application. The ending transaction causes a release of all table, row, page, and index locks that were on hand during the transaction.

13.1.3 Lock granularity

With IBM Informix Servers, the developer can apply locks to databases, tables, disk pages, data rows, or index-key values. At a database level, an exclusive lock is simple to enforce for a developer, but it has a big impact on users. No one gets access until the database lock processing is completed. Concurrency drops to zero, and performance is maximized for the exclusive use of a single user running an application.

At the other end of the granularity scope for locks, a transaction can exclusively lock a row, and have no impact on other users in their work efforts (if they are not trying to access the same row, and the lock mode is row). Concurrency is maximized, but performance will tend to slide as the number of users for a database or table increases.

Table locks

It is the responsibility of both database administrator and developer to select and enable the best level of lock granularity for users and for their database system. At the database level, administrative activities such as imports and exports are usually the task of the database administrator. For such activity, the database administrator would use a command such as

```
DATABASE database_name EXCLUSIVE;
```

Another task, principally for the database administrator, is to enable a change for a table or an index structure. The task of enabling work on an entire table can be done with a command similar to one of the following command:

- `LOCK TABLE table_name IN EXCLUSIVE MODE;`
- `LOCK TABLE table_name in SHARE MODE;`

When the task is finished (end of statement or transaction reached), the table is implicitly unlocked. It could be unlocked explicitly with the following command:

```
UNLOCK TABLE table_name;
```

Note: A table lock on a table can decrease update concurrency radically. Only one update transaction can access that table at any given time, and that update transaction locks out all other transactions. However, multiple read-only transactions can simultaneously access the table. This behavior is useful in a data warehouse environment where the data is loaded and then queried by multiple users.

Page locks

Lock mode PAGE is the default for Informix tables, and it is considered the optimal level in lock efficiency when rows are being accessed and modified in physical order. If your tables are large in row count and small in row size, a page level lock can be severely limiting, because it will lock a large number of rows on a page, and discourage user access.

In this case, it would be more efficient to change the default lock mode. For all new tables, this can be done by way of the Informix `onconfig` file, using the parameter DEF_TABLE_LOCKMODE. For example:

```
DEF_TABLE_LOCKMODE ROW;
```

Note: Use a small (or default) page size if your application contains small sized rows. Increasing the page size for an application that randomly accesses small rows can decrease performance. In addition, a page lock on a larger page will lock more rows, which is likely to reduce concurrency in some situations.

Tables that use page locks cannot support the USELASTCOMMITTED concurrency feature.

Row and key locks

Row and key locks generally provide the best overall performance when you are updating a relatively small number of rows, because they increase concurrency. However, the database server incurs some overhead in obtaining a lock. For an operation that changes a large number of rows, obtaining one lock per row might not be cost effective. For operations that consistently change a large number of rows, page locks might be a better option.

If a table is not created with row locking and you want row or key locks, you must alter the table. Here is an example to show how to create a table with row locking turned on:

```
CREATE TABLE table_namer(field1 serial,field2 char(20)...)
LOCK MODE ROW;
```

The ALTER TABLE statement can also change the lock mode. An example for this command syntax is:

```
ALTER TABLE table_name LOCK MODE (ROW);
```

When the lock mode is ROW and you insert or update a row, the database server creates a row lock. In some cases, you place a row lock by simply reading the row with a SELECT statement.

When the lock mode is ROW and you insert, update, or delete a key (performed automatically when you insert, update, or delete a row), the database server also creates a lock on the key in the index.

Key-value locks

When a user deletes a row within a transaction, the row cannot be locked because it becomes a non-existent row. However, the database server must somehow record that a row existed until the end of the transaction. The database server uses key-value locking to lock the deleted row. Key locks are used identically to row locks. When the table uses row locking, key locks are implemented as locks on imaginary rows.

When the table uses page locking, a key lock is placed on the entire index page that contains the key or that would contain the key if it existed. A page lock on an index page can decrease concurrency more substantially than a page lock on a data page. Index pages are dense and hold a large number of keys. By locking an index page, you make a potentially large number of keys unavailable to other users until you release the lock.

Note: To determine the current lock mode for a table:

- On UNIX, try "`oncheck -pt dbname:tablename | grep Locking`"
- On Windows systems, examine the output of "`oncheck -pt dbname:tablename`"

Database locks

The act of opening a database places a shared lock on the database name. The statements which open a database are CONNECT, DATABASE, or CREATE DATABASE. As long as a database is open, the shared lock on the database prevents any other program from dropping the database or putting an exclusive lock on it.

Locking a database for exclusive use are not needed very often, because this would prevent other users and programs from accessing it for the duration of the lock. The usual reason for a database lock would be the need for a major structure change across several tables, when implementing a series of related indexes on several tables, or when you have an application that needs uninterrupted access to the database for a period of time. To lock a database in exclusive mode, the syntax is:

```
DATABASE database_name EXCLUSIVE;
```

Tasks that run from the sysadmin database (dbscheduler tasks, such as Auto Update Statistics), occasionally might prevent exclusive database access. In this case, you must temporarily disable the dbscheduler. In the sysadmin database, stop the scheduler API with:

```
execute function task(scheduler shutdown)
```

Restart the scheduler API with:

```
execute function task(scheduler start)
```

Smart large object locks

Smart large objects are quite different in the way they work from the rest of the structures and processes in IBM Informix databases. The locking and locking granularity for smart large objects is also different. The database server uses one of the following granularity levels for locking smart large objects:

- The sbspace chunk header partition
- The smart large object
- A byte range of the smart large object

The default locking granularity for a smart BLOB is at the level of the smart large object. When you update a smart large object, the database server locks the smart large object that is being updated. Concurrently, there is an update lock placed on the sbspace chunk header partition while the object is being updated.

Byte locks

Byte locks, also known as byte-range locks, are used to lock a specific byte range of a smart large object. Byte-range locking is advantageous because it allows multiple users to update the same smart large object simultaneously, as long as they are updating different parts of it. Also, users can read a part of a smart large object while another user is updating or reading a different part of the same smart large object.

How the database server manages byte-range locks

The database server manages byte-range locks in the lock table in a similar fashion to other locks placed on rows, pages, and tables. However, the lock table must store the byte range as well. If a user places additional locks on a byte range, any new byte locks in the range that are contiguous are consolidated into one lock range.

Likewise, if a user unlocks a portion of the bytes included within a byte-range lock, the database server will split into multiple byte-range locks.

To enable use of byte-range locks

By default, the database server places a lock on the entire smart large object. At the time of sbspace creation, there is an option to use byte-range locking.

When the DBA sets the default locking mode for the sbspace to byte-range locking, the database server locks only the necessary bytes when it updates any smart large objects stored in the sbspace. To set byte-range locking for the sbspace that stores the smart large object, the database administrator must use the `onspaces` utility.

The following example sets byte-range locking for a new sbspace:

```
onspaces -c -S sblob -g 2 -p /ifmx/sblob1 -o 0 -s 1000 -Df LOCK_MODE=RANGE
```

When byte-range locking is set for the individual smart large object, the database server implicitly locks only the necessary bytes when it selects or updates the smart large object. The application developer can set byte-range locking for the smart large object when it is opened, using one of the following methods:

- Set the `MI_LO_LOCKRANGE` flag in the `mi_lo_open() DataBlade` API function.
- Set the `LO_LOCKRANGE` flag in the `ifx_lo_open() ESQL/C` function.

To lock a byte range explicitly, use one of the following functions:

- mi_lo_lock()
- ifx_lo_lock()

These functions lock the range of bytes that is specified for the smart large object. If the developer specifies an exclusive lock with either function, UPDATE statements do not place locks on the smart large object if they update the locked bytes.

The database server releases exclusive byte-range locks placed with `mi_lo_lock()` or `ifx_lo_lock()` at the end of the transaction. The database server releases shared byte-range locks placed with `mi_lo_lock()` or `ifx_lo_lock()` based on the same rules as locks placed with SELECT statements, depending upon the isolation level. The application can also release shared byte-range locks with `mi_lo_unlock()` or `ifx_lo_unlock()`.

For more information about these DataBlade API functions, see *IBM Informix: DataBlade API Programmer's Guide*:

http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.dapip.doc/dapip.htm

You can fine additional details about the ESQL/C functions in the *IBM Informix: ESQL/C Programmer's Manual* at:

http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.esqlc.doc/esqlc.htm

13.2 Locking issues and performance

Row and key locks generally provide the best performance whenever a database system only needs to update a small number of rows at a time. However, the database incurs additional overhead in obtaining a lock as the granularity gets smaller. The developer should help decide the granularity based on the character of the trnasactions.The following are guidelines for your decision making:

- For low row count transactions, use row level locking.
- For large processes that update an entire table, set locking to page or lock the table in exclusive mode before processing.
- For massive updates to many tables or the whole database, lock the database in exclusive mode before processing.

The type of isolation can affect overall performance because it affects concurrency. Before you execute a SELECT statement, you can set the isolation level with one of these options:

- The SET ISOLATION statement (an extension to ANSI SQL-92 standard)
  - Can be executed more than once in a transaction.
  - Can change the enduring isolation level for a session
  - Has an additional isolation level (Cursor Stability)
- SET TRANSACTION (ANSI/ISO-compliant)
  - Can only be executed once in a transaction.
- Dirty Read isolation (or ANSI Read Uncommitted) level does not place any locks on any rows fetched during a SELECT statement. Dirty Read isolation is appropriate for static tables that are used for queries. Use Dirty Read isolation with care if update activity occurs at the same time. With Dirty Read, the

reader can read a row that has not been committed to the database and might be eliminated or changed during a rollback.

13.2.1 Deadlocks

A deadlock is a situation in which a pair of programs blocks the progress of each other. Each program has a lock on some object that the other program wants to access. In a database with local database queries only, a deadlock arises only when all programs concerned set their lock modes to wait for locks.

An Informix database server detects deadlocks immediately when they only involve data at a single network server. Before a lock is granted, the database server examines the lock list for each user. If a user holds a lock on the resource that the requestor wants to lock, the database server traverses the lock wait list for the user to see if the user is waiting for any locks that the requestor holds. If so, the requestor receives a deadlock error. It prevents the deadlock from occurring by returning an error code (error -143 ISAM error: deadlock detected) to the second program to request a lock. The error code is the one the program receives if it sets its lock mode to not wait for locks. If your program receives an error code related to locks even after it sets lock mode to wait, you know the cause is an impending deadlock.

The best way to avoid deadlocks is to set lock mode to wait for a specific number of seconds (N), where N is the number of seconds it is reasonable for your applications to wait on a lock. Example syntax:

```
SET LOCK MODE TO WAIT 30;
```

Deadlock errors can be unavoidable when applications update the same rows frequently. However, certain applications might always be in contention with each other. Examine applications that are producing a large number of deadlocks and try to run them at different times. To monitor the number of deadlocks, monitor the `deadlks` field in the output of **`onstat -p`**.

Distributed transactions

Deadlock handling is more challenging when you are using a distributed query, which involves more than one database server. In this case, the server cannot monitor the locks of the other database, so deadlocks cannot be detected before they occur.

Occasionally, there might be a need for each database to wait on a lock from another local transaction. When the wait occurs, the entire distributed transaction might need to wait until the action is cleared up. If both servers get into a wait-for-the-other-server to complete mode, you have a multi-server deadlock that neither server can get out of easily.

To avoid this situation, set the DEADLOCK_TIMEOUT parameter in the `onconfig` file. This parameter tells the server how much time to wait before returning an error code (error -143 ISAM error: deadlock detected). For more information about using this configuration parameter, see *IBM Informix Dynamic Server Administrator's Guide, v11.50*, SC27-3606.

Distributed transactions, also known as *global transactions*, can have additional problems beyond concurrency and locking. In a distributed transaction, we can have queries that include multiple, separate databases spanning multiple host systems across a network. With the added task of reaching out to different servers, we are confronted with delays caused by network traffic, connectivity issues, additional user authentication, and user activity on the other database server that can slow or temporarily prevent quick access to the remote database server.

In the context of a query that is doing an update, insert, select, or delete, this new group of interfering factors can occasionally present problems. Developers should include coding conventions to accommodate distributed (XA) transactions and include code to avoid certain problems. We have included an ESQL/C routine (`xa_tool.ec`) in Appendix B, “Accommodating distributed transactions” on page 469 that is designed to help provide a start for handling XA problems.

13.3 Isolation levels

The number and duration of locks placed on data during a SELECT statement depends on the level of isolation that the user sets. The type of isolation can affect overall performance because it affects concurrency.

You can set the isolation level with the SET ISOLATION or the ANSI SET TRANSACTION statement before you execute the SELECT statement. The main differences between the two statements are that SET ISOLATION has an additional isolation level, Cursor Stability, and SET TRANSACTION cannot be executed more than once in a transaction as SET ISOLATION can.

Dirty Read isolation

Dirty Read isolation (or ANSI Read Uncommitted) does not place any locks on any rows fetched during a SELECT statement. Dirty Read isolation is appropriate for static tables that are used for queries. It offers the best performance of all isolation levels, because the database server does not check or place any locks for queries.

If update activity can occur at the same time as a dirty read, there is a chance that a rollback might need to occur during the dirty read. The reader could read a row that has not been committed (this is known as a phantom read) to the database and the data would be lost during a subsequent rollback. Because of potential problems with uncommitted data that is rolled back, use Dirty Read isolation with care.

Databases that do not have logging turned on (and thus do not allow transactions) use Dirty Read as a default isolation level. In fact, Dirty Read is the only isolation level allowed for databases that do not have logging turned on.

Committed Read isolation

Committed Read isolation (or ANSI Read Committed) removes the problem of phantom reads. A reader with this isolation level checks for locks before it returns a row. By checking for locks, the reader cannot return any uncommitted rows.

The database server does not actually place any locks for rows read during Committed Read. It simply checks for any existing rows in the internal lock table.

Committed Read is the default isolation level for databases with logging, and it is an appropriate isolation level for most activities.

Cursor Stability isolation

A reader with Cursor Stability isolation acquires a shared lock on the row that is currently fetched. This action assures that no other user can update the row until the user fetches a new row.

If you do not use a cursor to fetch data, Cursor Stability isolation behaves in the same way as Committed Read. No locks are actually placed.

Repeatable Read isolation

Repeatable Read isolation (also known as ANSI Serializable and ANSI Repeatable Read) is the strictest isolation level. With Repeatable Read, the database server locks all rows examined (not just fetched) for the duration of the transaction.

Repeatable Read is useful during any processing in which multiple rows are examined, but nothing will change during the transaction. The original application holds a read lock on each account that it examines until the end of the transaction, so the attempt by the second application to change the first account fails (or waits, depending upon SET LOCK MODE).

Note: With Repeatable Read, because even examined rows are locked, if the database server reads the table sequentially, a large number of rows unrelated to the query result can be locked.

Use Repeatable Read isolation for tables when the database server can use an index to access a table. If an index exists and the optimizer chooses a sequential scan instead, you can use directives to force use of the index. However, forcing a change in the query path might negatively affect query performance.

13.4 Configuration options

We have brought configuration parameters into our discussion several times. In this section, we examine IBM Informix Server configuration options that directly affect application development. We point out what the developers should know, and how these parameters affect the application and transactions. For more information about any of the parameters in the `onconfig` file or environment variables relating to configuration, see *IBM Informix Dynamic Server Administrator's Reference, v11.50,* SC27-3607.

13.4.1 Server identification

When the Informix server needs to resolve a database instance name, the `oninit` process uses the contents of a file known as the `sqlhosts` file. Each line in the `sqlhosts` file provides a database server naming reference, called the DBSERVERNAME, along with a protocol, a host system reference, and a listening protocol service reference. On Windows systems, this information is stored in the registry and is updated and accessed using the `setnet32.exe` utility. Multiple DBSERVERNAME references allow for an evenly distributed connection distribution over a variety of protocols

In addition to the information in the `sqlhosts` file, the DBSERVERNAME crosses reference to the `onconfig` file, which holds the configuration information describing how the database server is to operate. When a client application connects to a database server, it specifies a DBSERVERNAME, which helps the server to know which protocol will be used for the connectivity.

The DBSERVERNAME used in the application connection must be consistent with a name specified in the `sqlhosts` file and a DBSERVERNAME that is defined in the `onconfig` file.

In the `onconfig` file, there is a default (main instance) DBSERVERNAME and one or more alternate DBSERVERNAME parameters that are available. The DBA specifies the default name to be used with the DBSERVERNAME parameter in the `onconfig` file. For alternative application processing, the DBA can allow for or specify alternative protocols, referenced by an a different DBSERVERNAME. The alternate DBSERVERNAME or names are specified by the DBSERVERALIASES parameter in the `onconfig` file. An application developer should check with the DBA to determine whether to use the DBSERVERNAME or one of the DBSERVERALIASES for connecting with his application. Example SQL statements include CONNECT, DATABASE, CREATE TABLE, and ALTER TABLE, which can specify the database server instance.

13.4.2 Storage space identifiers

When tables are created, the database creator has syntax options available to define where tables and indexes are to be placed physically in a disk structure. Typically, the database administrator defines the disk structure layout. The disk layout is assigned by way of dbspaces, and each dbspace is further broken down into allocations of chunks.

The dbspace is the parent structure unit, used to hold storage information for tables and indexes, with low level definitions to describe where the actual data pointers for row data resides, as well as information about indexes, fragments, and table scope. The dbspace area that tracks pointer locations for data is known as the *table space*.

When defining the table and index schema, the database administrator can specify a dbspace location to hold the table or the index. Other space specifications for default storage are available if none are specified at creation time. These default locations define where to store binary large objects (BLOBs), character large objects (CLOBs), and temporary storage locations.

The default location for the location of temp tables and sorting space, if not predefined, is the dbspace where a table is accessed, or the root dbspace (rootdbs). If a default `onconfig` file definition for the location of temp dbspace is defined, we have more control over the location used

The DBSPACETEMP parameter

The DBSPACETEMP parameter specifies a list of dbspaces that the database server uses to globally manage the storage of temporary tables. When a temporary space is available, it improves performance by enabling the database server to spread I/O for temporary tables across multiple disks efficiently. The database server also uses temporary dbspaces during backups to store the before-images of data that are overwritten while the backup is occurring. More than one dbspace can be specified for this parameter. Simply list them as comma-separated values.

The DBSPACETEMP parameter can contain dbspaces with a non-default page size, but all of the dbspaces in the DBSPACETEMP list must have the same page size.

When using a logged database, file activity in temporary dbspaces is not logged. If the developer writes a query that requires tempspace, it is useful to include the phrase WITH NO LOG, so that you can force the query to use the designated temporary dbspace.

The SBSPACENAME parameter

The SBSPACENAME parameter specifies the name of the default sbspace. If your database tables include smart-large-object columns that do not explicitly specify a storage space, that data is stored in the sbspace that SBSPACENAME specifies. The default sbspace is also used by the built-in encryption and decryption functions to store BLOB or CLOB values.

An sbspace is a specialized type of dbspace, used for storing smartblobs and binary large object dbspaces. (Binary data or some type of multidimensional data is common to any datablade that uses R-tree indexing.)

When a create table statement includes a column that defines a CLOB or BLOB object, the column information will be stored in an sbspace. If the PUT clause is not specified in the create table statement when the defined CLOB or BLOB is created, the default location where the column data will be stored is the sbspace designated by the SBSPACENAME parameter.

If you are using IBM Informix with J/Foundation, you must provide a smart large object where the database server can store the Java archive (`.jar`) files. These `.jar` files contain the Java user-defined routines (UDRs). If you use Java UDRs, you will want to create additional, separate sbspaces for storing smart large objects.

The SBSPACETEMP parameter

The SBSPACETEMP parameter specifies the name of the default temporary sbspace for storing temporary smart large objects without metadata or user-data logging. If you store temporary smart large objects in a standard sbspace, the metadata is logged. For more information about using temporary smart large objects, see *IBM Informix DataBlade API Programmer's Guide, V11.50,* SC23-9429.

The SYSSBSPACENAME parameter

The SYSSBSPACENAME parameter specifies the name of the sbspace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types. Normally, the database server places statistics in the `sysdistrib` system catalog table.

Note: For information about writing user-defined statistics, refer to *IBM Informix User-Defined Routines and Data Types Developer's Guide, V11.5,* SC23-9438.

For information about providing statistics data for a smart BLOB column, refer to *IBM Informix DataBlade API Programmer's Guide, V11.50*, SC23-9429.

13.4.3 Limiters and limits

In addition to the storage location identifiers mentioned in 13.4.2, “Storage space identifiers” on page 450, a number of parameters in the `onconfig` file impose resource limits. Limits help the engine and system to utilize appropriate resources rather than consume all the resources on the system and end up in a crash. When the resources are no longer needed, they are released to be used by other processes. These configuration limits can be adjusted in cooperation with the database administrator, to accommodate for application concerns that might need them.

The MULTIPROCESSOR parameter

The MULTIPROCESSOR parameter is used to determine the type of processing method the engine will use. If it is set to 0, locking is done in a way that is suitable for a single- processor computer, and processor affinity is ignored.

If the nearby `onconfig` file SINGLECPU_VP parameter is non-zero (On), MULTIPROCESSOR and user-defined VPCLASSes (of any kind) will not work.

The VPCLASS parameter

The VPCLASS parameter allows you to designate a class of virtual processors (VPs), create a user-defined VP, and specify the following information for it:

- The number of virtual processors that the database server should start initially.
- The maximum number of virtual processors allowed for this class.
- The assignment of virtual processors to CPUs if processor affinity is available.
- The disabling of priority aging by the operating system if the operating system implements priority aging.

You can have multiple VPCLASS parameter definitions. Use one VPLCLASS reference for each class of virtual processor, one per line. Basic guidelines for defining a VPLCASS are in *IBM Informix Dynamic Server Administrator's Reference, V11.50*, SC27-3607.

For information about creating a user-defined virtual process, see *IBM Informix User-Defined Routines and Data Types Developer's Guide, V11.5,* SC23-9438 or *J/Foundation Developer's Guide, V11.5*, SC23-9434.

The NETTYPE parameter

The NETTYPE parameter provides tuning options for the protocols that DBSERVERNAME entries define in the `sqlhosts` file or registry. Each DBSERVERNAME entry in the `sqlhosts` file or registry is defined in relation to either the DBSERVERNAME parameter or the DBSERVERALIASES parameter in the ONCONFIG file.

The NETTYPE configuration parameter describes a network connection as follows:

- The protocol (or type of connection).
- The number of poll threads assigned to manage the connection.
- The expected number of concurrent connections.
- The class of virtual processor that will run the poll threads.
- You can specify a NETTYPE parameter for each protocol that you want the database server to use.
- There is a special NETTYPE setting available if you want to enable the database server to use multiplexed connections on UNIX. To do this, you designate a NETTYPE with the value `sqlmux`, as in the following example.

  ```
  NETTYPE sqlmux
  ```

The LOCKS parameter

The LOCKS parameter specifies the initial size of the lock table. Every SQL session that connects to a database, and accesses tables and rows, generates a lock. If the lock is non-exclusive, it is shared; if something needs to change, it is exclusive. If there is an intent to change an object, it is an intent-exclusive lock. The lock table holds an entry for each type of lock entry, and each of these lock entries will have a small allocation in resident memory.

When the number of locks exceeds the lock table value, on 32-bit servers, the database server will increase the size of the lock table by doubling the lock table value, up to 99 times, or to the maximum value for the server allowance (whichever comes first). On 32-bit servers, the maximum limit is 8,000,000 locks.

On 64-bit servers, the lock table limit is based on the maximum starting locks value (500,000)+99 allowed increments of 1,000,000 locks for a total of 599,000,000 locks.

The amount of memory storage per lock ranges from 100 to 200 bytes, depending on the byte-word size and the platform.

Note: All lock table increments are kept in virtual memory. If the server engine has limited shared memory, locks can become a memory resource drain.

The STACKSIZE parameter

The STACKSIZE parameter specifies the stack size for database server user threads. Setting a value for STACKSIZE that is too large wastes virtual memory space and can cause swap-space problems.

For 32-bit platforms, the default STACKSIZE value of 32 KB is sufficient for non-recursive database activity. For 64-bit platforms, the recommended STACKSIZE value is 64 KB.

In recursive SQL routines, the server checks for the possibility of stack-size overflow and automatically expands the stack.

UDRs should increase the stack size for a routine as needed, using the stack modifier in the CREATE FUNCTION statement.

The USELASTCOMMITTED parameter

This parameter specifies the isolation level whenever the LAST COMMITTED feature of the COMMITTED READ isolation level is implicitly in effect. The LAST COMMITTED feature can reduce the risk of locking conflicts between concurrent transactions on tables that have exclusive row locks.

Important: The USELASTCOMMITTED parameter works only with tables that have been created or altered to have ROW as their locking granularity.

For the USELASTCOMMITTED parameter to work as expected, and given that a table has been created or altered to have ROW as their locking granularity, the following considerations also apply:

- If SET TRANSACTION is enabled with READ COMMITTED or READ UNCOMMITTED, USELASTCOMMITTED will work.
- Tables created without any explicit lock mode setting will use the default setting in DEF_TABLE_LOCKMODE.

The DEF_TABLE_LOCKMODE parameter

The DEF_TABLE_LOCKMODE parameter sets the lock mode for every newly created table for all sessions connected to a logging or nonlogging database, and has no effect on existing tables in the database.

The specified value can be ROW, which sets the lock mode to ROW for every new table connected to a database. If the specified value is PAGE (default), any exclusive locked table remains inaccessible.

There is also an environment variable on the client side, DEF_TABLE_LOCKMODE, which can be set with a lock mode value (PAGE or ROW). The environment variable has rules of precedence involved if the `onconfig` value is also set. For more information, see *IBM Informix Dynamic Server Administrator's Reference, V11.50,* SC27-3607.

The DEADLOCK_TIMEOUT parameter

The DEADLOCK_TIMEOUT parameter specifies the maximum number of seconds that a database server thread can wait to acquire a lock. Use this parameter only for distributed queries that involve a remote database server.

Note: This parameter applies only to distributed queries. There is a separate, automated mechanism for deadlocks that are internal to a local database.

The OPTCOMPIND parameter

The OPTCOMPIND parameter determines the method used by the engine to process a query. All queries are optimized to run based on the guideline recommended by this parameter, unless the developer forces a different guideline using a directive.

When the OPTCOMPIND parameter is set to one of the following values, a given query is optimized using the guideline assigned:

0	When appropriate indexes exist for tables in the query, the optimizer chooses index scans, without consideration of the cost, over table scans.
1	As long as the isolation is not Repeatable Read, the optimizer will use cost based decisions. If Repeatable Read is in use, it behaves as though OPTCOMPIND=0. Setting 1 is recommended for optimal performance.
2	The optimizer uses cost to determine an execution path regardless of isolation level. Index scans are not given preference; the optimizer decision is purely based on cost. Setting 2 is the default if the parameter is not set.

Note: Use the same OPTCOMPIND value in the development and in the production environment. Performance and query behavior can have variations if the OPTCOMIND value is different in the development and production environments.

The DIRECTIVES parameter

The DIRECTIVES parameter enables or disables the use of SQL directives. SQL directives allow you to specify behavior for the query optimizer in developing query plans for SELECT, UPDATE, and DELETE statements.

Set DIRECTIVES to 1, which is the default value, to enable the database server to process directives.

Set DIRECTIVES to 0 to disable the database server from processing directives.

Note: Client programs have the option to set the IFX_DIRECTIVES environment variable to ON or OFF to enable or disable processing of directives by the database server. The setting of the IFX_DIRECTIVES environment variable overrides the setting of the DIRECTIVES configuration parameter. If you do not set the IFX_DIRECTIVES environment variable, all sessions for a client inherit the database server configuration for processing SQL directives.

The MAX_PDQPRIORITY parameter

The MAX_PDQPRIORITY parameter limits the PDQ resources that the database server can allocate to a given DSS query. MAX_PDQPRIORITY is a factor that is used to scale the value of PDQ priority set by users. For example, if the database administrator sets MAX_PDQPRIORITY to 80, and a user sets the PDQPRIORITY environment variable to 50 and issues a query, the database server silently processes the query with a PDQ priority of 40. The database administrator can use the `onmode` utility to change the value of MAX_PDQPRIORITY while the database server is online.

In Informix database server, PDQ resources include memory, CPU, disk I/O, and scan threads. the MAX_PDQPRIORITY parameter lets the database administrator run decision support concurrently with OLTP, without a deterioration of OLTP performance. However, if the MAX_PDQPRIORITY parameter is too low, the performance of decision- support queries can degrade.

Table 13-1 lists the MAX_PDQPRIORITY value that you can set.

Table 13-1 The MAX_PDQPRIORITY value

Value	Database Server Action
0	Turns off PDQ. DSS queries use no parallelism.
1	Fetches data from fragmented tables in parallel (parallel scans) but uses no other form of parallelism.
100	Uses *all* available resources for processing queries in parallel.

Value	Database Server Action
An integer between 1-100	Sets the percentage of the user-requested PDQ resources to be allocated to the query.

13.4.4 Java configuration parameters

The configuration parameters listed in Table 13-2 allow you to use J/Foundation, which incorporates an embedded Java Virtual Machine (JVM) on the database server. For more information about these parameters, see *J/Foundation Developer's Guide, V11.5*, SC23-9434.

Table 13-2 Java configuration parameters

Parameter name	Description
AFCRASH 0x10	When the 0x10 bit is on for AFCRASH, all the messages that the JVM generates are logged into the `JVM_vpid` file, where `vpid` is the process ID of the Java virtual processor. This file is stored in the directory where the JVPLOG file is stored.
JVPDEBUG	When set to 1, tracing messages are written to the JVPLOG file.
JVPHOME	Directory where the classes of the IBM Informix JDBC Driver are installed.
JVPLOGFILE	Absolute path name for your Java VP log files.
JVPPROPFILE	Absolute path name for the Java VP properties file.
JVPJAVAVM	Libraries to use for the JVM.
JVPJAVAHOME	Directory where the Java Runtime Environment (JRE) for the database server is installed.
JVMTHREAD	Thread package (green or native) to use for the JVM.
JVPVJAVALIB	Path from JVPJAVAHOME to the location of the Java VM libraries.
JVPCLASSPATH	Initial Java class path setting.
VPCLASS jvp=n	Number of Java virtual processors that the database server should start.

For additional `onconfig` file parameters and how they relate to development, see Appendix A, “Parameters in the onconfig file” on page 467.

13.5 Working with your database administrator

There is an assumption throughout this book that our main audience is the developer, with a distinctive separate role from the database administrator (DBA). The database server administrator has the task of allocating and handling resources effectively in the server side of a client -server environment. When a performance issue occurs or when a transaction process is not working as it should, you need to work with the database administrator to find a solution. Present the problem to the database administrator, who might have a good idea about a particular configuration area that might relate to the problem. There are a few areas we can discuss here, which might help target the common problems that developers experience.

Configuration parameters that affect logging

Checkpoints, logging, and page cleaning are necessary to maintain database consistency. A direct trade-off exists between the frequency of checkpoints or the size of the logical logs and the time that it takes to recover the database in the event of a failure. A database administrator has to consider ways to reduce the overhead for these activities based on the acceptable delay during recovery. Sometimes these decisions are influenced by the behavior of the application. Here we discuss the configuration parameters that effect logging.

LOGBUFF and PHYSBUFF

The LOGBUFF and PHYSBUFF configuration parameters effect logging I/O activity because they specify the respective sizes of the logical-log and physical-log buffers that are in shared memory. The size of these buffers determines how quickly they fill and therefore how often they need to be flushed to disk. If checkpoints tend to have long duration, you can seek further information related to buffer tuning and checkpoint tuning information in *the IBM Informix Dynamic Server Performance Guide, V11.50,* SC27-3618-00.

LOGFILES

The LOGFILES parameter specifies the number of logical-log files. This parameter only becomes important with respect to the long transaction high water mark (LTXHWM and LTXEHWM), if we are having trouble with transactions that need to be rolled back. See the sections on DYNAMIC_LOGS, LTXHWM and LTXEHWM, later in this chapter for more details.

LOGSIZE

Choose a log size based on how much logging activity occurs and the amount of risk in case of catastrophic failure. If you cannot afford to lose more than an hour's worth of data, create many small log files that each hold an hour's worth of transactions. If your system is stable with high logging activity, choose larger logs to improve performance.

Note: A backup process can hinder transaction processing if data is located on the same disk as the logical-log files. It is better to keep logical logs on separate disks from data, if possible. If extra disks are not an option for separate logical log space, however, you can wait for periods of low user activity before you back up the logical-log files.

You need to adjust the size of the logical log when your transactions include simple large objects or smart large objects, as the following sections describe.

Estimating logical-log size when logging simple large objects

To obtain better overall performance for applications that perform frequent updates of TEXT or BYTE data in blobspaces, reduce the size of the logical log. Blobpages cannot be reused until the logical log to which they are allocated is backed up. When TEXT or BYTE data activity is high, the performance impact of more frequent checkpoints is balanced by the higher availability of free blobpages.

When you use volatile blobpages in blobspaces, smaller logs can improve access to simple large objects that must be reused. Simple large objects cannot be reused until the log in which they are allocated is flushed to disk. In this case, you can justify the cost in performance because those smaller log files are backed up more frequently.

Estimating logical-log size when logging smart large objects

If you plan to log smart large object user data, you must ensure that the log size is considerably larger than the amount of data being written. Smart large object metadata is always logged even if the smart large objects are not logged.

Use the following guidelines when you log smart large objects:

- If you are *appending* data to a smart large object, the increased logging activity is roughly *equal* to the amount of data written to the smart large object.
- If you are *updating* a smart large object (overwriting data), the increased logging activity is roughly *twice* the amount of data written to the smart large object. The database server logs both the before-image and after-image of a smart large object for update transactions. When updating the smart large objects, the database server logs only the updated parts of the before and after image.

- Metadata updates affect logging less. Even though metadata is always logged, the number of bytes that are logged is usually much smaller than the smart large objects.

The DYNAMIC_LOGS parameter

The default value for the DYNAMIC_LOGS configuration parameter is 2, which means that the database server allocates a new logical log file automatically after the current log file when it detects that the next log file contains an open transaction. The database server automatically checks whether the log after the current log still contains an open transaction at the following times:

- Immediately after it switches to a new log file while writing log records (not while reading and applying log records)
- At the beginning of the transaction cleanup phase which occurs as the last phase of logical recovery

Logical recovery happens at the end of fast recovery and at the end of a cold restore or roll forward. For more information about the phases of fast recovery, see *IBM Informix Dynamic Server Administrator's Guide*, Version 11.50, SC27-3606.

If you set DYNAMIC_LOGS to 0, the database server still checks whether the next active log contains an open transaction when it switches log files. If it finds an open transaction in the next log to be active, it issues the following warning:

```
WARNING: The oldest logical log file (%d) contains records from an open
transaction (0x%p), but the Dynamic Log Files feature is turned off.
```

The LTXHWM and LTXEHWM parameters

The LTXHWM parameter indicates how full the logical log is when the database server starts to check for a possible long transaction and to roll it back. LTXEHWM indicates the point at which the database server suspends new transaction activity to locate and roll back a long transaction.

With the dynamic log file feature, long transaction high watermarks are no longer as critical because the database server does not run out of log space unless you use up the physical disk space available on the system. Under normal conditions, you should keep the default values for LTXHWM and LTXEHWM. If a rollback is ever needed, it can indicate a serious problem within an application.

After Version 9.4, LTXHWM and LTXEHWM are not in the `onconfig.std` file and default to the following values, depending on the value of the DYNAMIC_LOGS configuration parameter:

- With DYNAMIC_LOGS set to 1 or 2, the long transaction high watermark default values are 80 for LTXHWM and 90 for LTXEHWM. Because the database server does not run out of logs, other users can still access the log during the rollback of a long transaction.
- With DYNAMIC_LOGS to 0, the default values are 50 for LTXHWM and 60 for LTXEHWM.

You might want to change these default values for one of the following reasons:

- To allow other transactions to continue update activity (which requires access to the log) during the rollback of a long transaction. In this case, you increase the value of LTXEHWM to raise the point at which the long transaction rollback has exclusive access to the log.
- To perform scheduled transactions of unknown length, such as large loads that are logged. In this case, you increase the value of LTXHWM so that the transaction has a chance to complete before reaching the high watermark.

13.5.1 Parameters for negotiation

While the database administrator should make final decisions for server parameters, there are several parameters for which you need agreement to achieve the most effective results.

The VPCLASS parameter

If you decide to use UDRs, a UDR can use the processing power in the CPU class that users currently have at their disposal, or you can define the UDR with its own processor class when you create the function. You define a new class of virtual processors to isolate UDR execution from other transactions that execute on the CPU virtual processors. This method is typically used when you write UDRs to support user-defined data types. The class name that you specify in the VPCLASS parameter must match the name specified in the CLASS modifier of the CREATE FUNCTION statement.

Note: Because a user-defined CPU class is treated as a virtual processor, the `onconfig` file SINGLE_CPU_VP parameter must be off (0) and MULTIPROCESSOR must be on (1).

Guidelines for setting VPCLASS:

- For uniprocessor computers, it is recommended that you use one CPU virtual processor.
- VPCLASS cpu,num=1.
- For multiprocessor systems with four or more CPUs that are primarily used as database servers, it is recommended that you set the VPCLASS `num` option to one less than the total number of processors. For example, if you have four CPUs, use the following specification:

  ```
  VPCLASS cpu,num=3
  ```

Process priority aging

On some operating systems, *priority aging* occurs when the operating system lowers the priority of long-running processes as they accumulate processing time. Always disable the priority aging parameter of VPCLASS because it can cause the performance of the database server processes to decline over time. For further information, check the system notes for your database server.

Processor affinity

Processor affinity distributes the computation impact of CPU virtual processors and other processes. On computers that are dedicated to the database server, assigning CPU virtual processors to all but one of the CPUs achieves maximum CPU utilization. On computers that support both database server and client applications, you can bind applications to certain CPUs through the operating system. By doing so, you effectively reserve the remaining CPUs for use by database server CPU virtual processors, which you bind to the remaining CPUs with the VPCLASS configuration parameter.

Set the *aff* option of the VPCLASS parameter to the numbers of the CPUs on which to bind CPU virtual processors. For example, on an 8 CPU system, the following VPCLASS setting assigns CPU virtual processors to processors 4 to 7, and CPU’s 0, 1, 2, and 3 would be most available to the application:

```
VPCLASS cpu,num=4,aff=4-7
```

For additional methods that can be used with the VPCLASS, see *IIBM Informix Dynamic Server Administrator's Guide, V11.50*, SC27-3606.

13.5.2 Monitoring isolation levels

As a general discovery tool, the **`onstat -g sql`** command can be used for determining the isolation levels and lock modes for SQL statements actively running against the server, their associated databases and any SQL errors that might have occurred. The **`onstat -g sql`** command can also be run with an optional session ID. The output contains the most recent and the current SQL statements being run by the session.

13.5.3 Monitoring locks

The **`onstat -k`** command displays the current database locks held within the system. The type of lock is determined by the logging mode, isolation levels, and application design. Example 13-1shows the output of a sample **`onstat -k`** command.

Example 13-1 The onstat -k command output

```
IBM Informix Dynamic Server Version 11.50.UC6  -- On-Line -- Up 03:55:17 --
15360 Kbytes
Locks
address  wtlist   owner lklist type tblsnum  rowid    key#/bsiz
a095f78  0        a4d9e68  0 HDR+S    100002   203         0
 1 active, 2000 total, 2048 hash buckets, 0 lock table overflows
```

The information in this output explains lock aspects. The `type` column explains the lock types associated with each open lock. The abbreviations in the Type column defines lock types as follows:

HDR	Header
B	Byte lock
S	Shared
I	Intent
X	Exclusive
U	Update
IX	Intent exclusive
IS	Intent-shared
SIX	Shared, Intent exclusive

The lock level is determined by the value of the `tblsnum` (tblspace ID) and the row ID as show in Table 13-3 and Table 13-4.

Table 13-3 Determining lock level from Row ID

If RowID	Lock level is
Equals 0	Table
Ends in 00	Page
<6 digits & non zero	Row

Table 13-4 Determining lock level from tblsnum

If tblsnum ID	Lock level is
Equals 100002	Database
<10000	ER Pseudo lock

13.5.4 Monitoring user threads

The **`onstat -u`** command output reveals all of the current user threads running on a server, though the output can be cryptic. In a highly active system, it is normal to see user threads waiting on various conditions. These wait conditions change rapidly as resources are engaged and released.

To identify a problem when a particular session is having a problem, monitor the primary user threads by watching the flags over a few minutes. If the user thread remains and the thread and wait conditions are unchanging, it provides a sign that the time has come to focus in on a problem user thread.

Watch the flag values, then narrow down the problem by zeroing in the details of the problem with additional **`onstat`** commands.

To monitor user threads:

1. Look at flag position 5: Primary threads (P) are the only ones of interest (Ignore other user threads).
2. Look at flag position 1 (Wait conditions):
   - B=buffer
   - C=checkpoint
   - G=write of the Logical Log buffer
   - L=lock
   - S=mutex
   - T=transaction

- X=transaction cleanup
- Y=condition

  Watch for persistent L, S, T, or Y.

3. Look at flag position 3 (thread activity):
   - A=DBSpace backup
   - B=Begin work
   - P=Preparing/prepared
   - X=XA prepare
   - C=Commiting/committed
   - R=Aborting/aborted
   - H=Heuristic aborting/aborted

   Watch for P, X, C, R, H.

For more information about **`onstat`** commands and the meanings of the columns, refer to:

`http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.adref.doc/ids_adr_0608.htm`

A

Parameters in the onconfig file

In 13.4, “Configuration options” on page 449, we discuss several parameters that have a direct impact on developer methods and understandings. The list provided in that section is limited to the direct-impact related parameters. It is likely that application developers will not have much use for the remaining `onconfig` parameters most of the time.

Normally, this remaining configuration information is a concern only for the DBAs. Occasionally, there will come a time when performance of an application might come into question. Sometimes slow performance is caused by the application, and is under control of the developer. At other times it might be an `onconfig` file setting in the IBM Informix server that requires adjustment.

Table A-1 on page 468 lists the parameters that might help developers and DBAs to narrow the target area for further investigation. The table is not an exhaustive list. Parameters that do not apply or that we have already discussed in this book are not listed. For further information about any of these parameters, see *IBM Informix Dynamic Server Administrator's Reference, v11.50,* SC27-3607.

From the developer’s point of view, it makes sense to categorize the configuration parameters from a functional perspective, rather than a strict listing by name. Toward this objective, the parameter names are listed in the table by function groupings to help narrow down the parameter list that can be considered for an area of specific interest. The functional groupings do not reflect specific naming conventions used in the `onconfig` file.

Table A-1 Parameters in the onconfig file

Related function area	Parameters
Storage Space	ROOTNAME, ROOTOFFSET, ROOTSIZE, MIRROR, MIRROROFFSET, LOGFILES, LOGSIZE, DBSPACETEMP, SBSPACETEMP, SBSPACENAME, SYSSBSPACENAME, CLEANERS, STAGEBLOB, TBLTBLFIRST, TBLTBLNEXT, DATASKIP
Path locations	ROOTPATH, JVPJAVALIB, JVPJAVAVM, MSGPATH, PLOG_OVERFLOW_PATH
Buffer Movement	BUFFERPOOL,PHYSBUFF, PHYSFILE
Memory Cache	DD_HASHSIZE, DD_HASHMAX, DS_HASHSIZE, DS_POOLSIZE, PC_HASHSIZE, PC_POOLSIZE, STMT_CACHE, STMT_CACHE_HITS, STMT_CACHE_SIZE, STMT_CACHE_NOLIMIT, STMT_CACHE_NUMPOOL, PLCY_POOLSIZE, PLCY_HASHSIZE, USRC_POOLSIZE, USRC_HASHSIZE
Memory control	RESIDENT, SHMVIRTSIZE, SHMADD, EXTSHMADD, SHMTOTAL, SHMVIRT_ALLOCSEG, SHMNOACCESS,VP_MEMORY_CACHE_KB, ONLIDX_MAXMEM
Virtual processor and connection	NETTYPE, LISTEN_TIMEOUT, MAX_INCOMPLETE_CONNECTIONS, FASTPOLL, MULTIPROCESSOR, VPCLASS, SINGLE_CPU_VP, AUTO_AIOVPS, DIRECT_IO
SQL Control	LOCKS, OPTCOMPIND, DIRECTIVES, EXT_DIRECTIVES, OPT_GOAL, IFX_FOLDVIEW, RA_PAGES, RA_THRESHOLD
Transaction control	DEF_TABLE_LOCKMODE, BLOCKTIMEOUT, TXTIMEOUT, HETERO_COMMIT, DEADLOCK_TIMEOUT, DYNAMIC_LOGS, LOGBUFF, LTXHWM, LTXEHWM, TEMPTAB_NOLOG, AUTO_REPREPARE
Decision support	DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, DS_NONPDQ_QUERY_MEM

B

Accommodating distributed transactions

This appendix discusses the use of distributed transactions with an IBM Informix database server.

B.1 Distributed transactions

A *transaction* is a series of actions performed as a single logical unit of work in which either all of the actions are performed or none of them are.

A *distributed transaction* is a transaction that runs in multiple processes, usually on several systems, and normally involves actions against two or more databases. Each participant of a distributed transaction must agree to commit the changes before the distributed transaction can be committed.

There are three core components on a distributed transaction:

- Application program (AP)

  Application program implements the desired business function. It specifies a sequence of operations that involve resources such as databases. An application program participates in one or more units of work, and might decide to commit or roll back each unit of work independently.

- Resource manager (RM)

  Resource manager manages access to shared resources such as databases. The resource manager provides the services to manage the data involved in the distributed transaction.

- Transaction manager (TM)

  Transaction manager manages global transactions and coordinates the decision to commit them or roll them back ensuring their atomicity. The transaction manager also coordinates recovery activities of the resource managers when necessary, such as after a component failure.

The XA standards, set forth by the *Open Group's X/Open Distributed Transaction Processing* (DTP) model, define the interfaces between the transaction manager, the application program, and the resource manager in a DTP environment. These interfaces are implemented in the *Informix TP/XA Interface Library*.

B.2 TP/XA Transaction Manager XA Interface Library

TP/XA is a library of functions that allows the IBM Informix database server act as a resource manager in the X/Open DTP environment.The TP/XA library facilitates communication between a third-party transaction manager and the database server.

TP/XA is supplied with IBM Informix ESQL/C, which is included with Informix Client Software Development Kit (Client SDK).

In addition to the TP/XA library, a header file `xa.h`, is supplied that contains the definition for the functions and common structures, such as XID or `xa_switch_t`, which are used by the transaction manager and resource manager.

Table B-1 describes the functions used to work with XA transactions.

Table B-1 TP/XA macro definitions

Function	Description
xa_open()	Initializes the resource manager (database server) for an XA transaction
xa_close()	Close a currently open resource manager
xa_start()	Starts an XA transaction
xa_end()	Dissociates from an XA transaction
xa_rollback()	Tells the resource manager to roll back an XA transaction
xa_prepare()	Asks the resource manager to prepare to commit an XA transaction
xa_commit()	Tells the resource manager to commit an XA transaction
xa_recover()	Obtains a list of XIDs that are currently in a prepared or heuristically completed state
xa_forget()	Tells the resource manager to forget a heuristically completed transaction
xa_complete()	Test Completion of asynchronous XA Request. This function is provided only for compliance with the X/Open XA Specification

B.3 XA_TOOL ESQL/C sample

Example B-1is a basic ESQL/C application which demonstrates how to use some of the TP/XA functions such as `xa_prepare()` and `xa_rollback()` to perform operations with a distributed transaction.

Example B-1 The xa_tool.ec application

```
/**************************************************************
 *
 *    WARNING:    USE OF THIS PROGRAM MAY HAVE UNDESIRABLE AFFECTS
 *    TO THE DATABASE SERVER, INCLUDING DATA CORRUPTION!!
 *
 *    TITLE:    xa_tool.ec
 *
 *    Compile with:
 *        on UNIX systems:
 *             esql -o xa_tool xa_tool.ec  -lifxa
 *        on Windows systems
 *             esql xa_tool.ec
 *
 * This program will connect to a global transaction branch
 * and let the user manipulate the transaction.
 *
 * Usage:
 *                xa_tool <fID> <gtl> <bql> <hex data>
 *
 * The fID, gtl, bql, and hex data should be provided from the
 * transaction desired to be manipulated.    The information is
 * found by execution of the onstat -G command.    Example output
 * of the onstat -G command:
 *
 *        Global Transaction Identifiers
 *        address    flags            fID    gtl    bql    data
 *        cb2a964    0x8442a        0        2        4        4D4E4F000000
 *        1 active, 128 total
 *
 * If transaction cb2a964 is the transaction that is desired
 * to be manipulated, then this program should be executed
 * with the following command:
 *
 *        xa_tool 0 2 4 4D4E4F000000
 *
 *
 *************************************************************
 */


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "xa.h"

extern struct xa_switch_t infx_xa_switch;

#define xa_open(info, rmid, flags)   \
           ((*infx_xa_switch.xa_open_entry)(info,rmid,flags))
#define xa_start(gtrid,rmid, flags)  \
           ((*infx_xa_switch.xa_start_entry)(gtrid,rmid,flags))
#define xa_rollback(gtrid,rmid, flags) \
           ((*infx_xa_switch.xa_rollback_entry)(gtrid,rmid,flags))
#define xa_commit(gtrid,rmid, flags)    \
           ((*infx_xa_switch.xa_commit_entry)(gtrid,rmid,flags))
#define xa_rollback(gtrid,rmid, flags)    \
           ((*infx_xa_switch.xa_rollback_entry)(gtrid,rmid,flags))
#define xa_end(gtrid,rmid, flags)    \
           ((*infx_xa_switch.xa_end_entry)(gtrid,rmid,flags))
#define xa_prepare(gtrid,rmid, flags)\
           ((*infx_xa_switch.xa_prepare_entry)(gtrid,rmid,flags))
```

```
#define xa_close(info,rmid, flags)\
           ((*infx_xa_switch.xa_close_entry)(info,rmid,flags))
#define xa_forget(gtrid,rmid, flags)\
           ((*infx_xa_switch.xa_forget_entry)(gtrid,rmid,flags))
#define xa_recover(gtrid, count, rmid, flags)\
           ((*infx_xa_switch.xa_recover_entry)(gtrid,count,rmid,flags))

/* Doesn't matter what database is opened... */
#define OPEN_DATABASE "sysmaster"


/* Initialize the xid */
void
setup_myxid (XID *xid, int x_fID, int x_gtl, int x_bql, char* x_data)
{
    int ii, c;
    xid->formatID = x_fID;
    xid->gtrid_length = x_gtl;
    xid->bqual_length = x_bql;

    for(ii=0; ii<x_gtl+x_bql; ii++)
      if (sscanf( &x_data[ii*2], "%02x", &c) != 1)
      {
        printf("Invalid GTRID data!\n");
        exit(1);
      }
      else
      xid->data[ii] = (char) c;
}


void xa_tool(XID *xid)
{
    int choice = 0;
    int cc;

    while( choice != 'Q')
    {
        printf("\tP -- XA_PREPARE\n");
        printf("\tC -- XA_COMMIT\n");
        printf("\tR -- XA_ROLLBACK\n");
        printf("\tF -- XA_FORGET\n");
        printf("\tQ -- terminate program\n");
        printf("Enter Choice:  ");
        choice = getchar();

        choice = toupper(choice);
        printf("%c\n\n", choice);

        switch(choice)
        {
            case 'P':
                printf("Executing XA_PREAPRE\n");
                if ((cc = xa_prepare(xid, 0, TMNOFLAGS)) != XA_OK)
                {
                    printf("XA_PREPARE failed with %d\n", cc);
                }
                else
                    printf("... XA_PREPARE finished\n\n");
                break;
            case 'C':
                printf("Executing XA_COMMIT\n");
                if ((cc = xa_commit(xid, 0, TMNOFLAGS)) != XA_OK)
                {
                    printf("XA_COMMIT failed with %d\n", cc);
                }
                else
                    printf("... XA_COMMIT finished\n\n");
                break;
            case 'R':
                printf("Executing XA_ROLLBACK\n");
                if ((cc = xa_rollback(xid, 0, TMNOFLAGS)) != XA_OK)
                {
                    printf("XA_ROLLBACK failed with %d\n", cc);
                }
                else
                    printf("... XA_ROLLBACK finished\n\n");
                break;
            case 'F':
                printf("Executing XA_FORGET\n");
                if ((cc = xa_forget(xid, 0, TMNOFLAGS)) != XA_OK)
                {
                    printf("XA_FORGET failed with %d\n", cc);
                }
                else
                    printf("... XA_FORGET finished\n\n");
                break;
            case 'Q':
                break;
            default:
                printf("%c is not a valid option!\n\n", choice);
                choice = 0;
        }
    if (choice != 'Q') {
            choice = getchar();
            choice = 0;
    }
    }

}

int main(int arc, char *argv[])
{
    $int cc;
    XID xid;
    int xid_fID, xid_gtl, xid_bql;
    char xid_data[300];
    if (arc != 5)
    {
        printf("Error:    Incorrect number of parameters.\n");
        printf("Usage:    %s <fID> <gtl> <bql> <data>\n", argv[0]);
        exit(2);
    }
    printf("\n\n");

    xid_fID = atoi(argv[1]);
    xid_gtl = atoi(argv[2]);
    xid_bql = atoi(argv[3]);

    strcpy(xid_data, argv[4]);

    /* setup the XID which is used to identify the transaction */
    setup_myxid(&xid, xid_fID, xid_gtl, xid_bql, xid_data);

    /* establish connection to the database */
    printf("Calling xa_open ... \n");
    if ((cc = xa_open(OPEN_DATABASE, 0, TMNOFLAGS)) != XA_OK)
    {
        printf("xa_open failed with %d, sqlcode = %d\n", cc, SQLCODE);
        exit(1);
    }
    else
        printf("... xa_open finished\n\n");

    /** This is for xa_tool...    **/
    xa_tool(&xid);


    /* close the connection */
    printf("Calling xa_close...\n");
    if ((cc = xa_close("", 0, TMNOFLAGS)) != XA_OK)
    {
        printf("xa_close failed with %d\n", cc);
        exit(1);
    }
    else
        printf("... xa_close finished\n\n");
}
```

You also can use the `xa_tool` example to complete unresolved transactions left in the database server. Local transactions are rolled back automatically by the Informix database server if the connection between the client application and the database server is lost. This roll back occurs because the database server is in direct control of the transaction.

When using distributed transactions, the database server cannot resolved a transaction automatically. The Transaction Manager is the only process in charge of committing or aborting the global transaction. Thus, if there is a communication error or a failure within the resource manager, there is a chance of leaving unresolved transactions in the database server. These transactions can be resolved using the `xa_commit()`, `xa_rollback()`, and `xa_forget()` functions.

Example B-2 shows how to compile and use the `xa_tool.ec` program to rollback an XA transaction left in the Informix database server.

Example B-2 The xa_tool.ec program output

```
D:\Infx\ids1150>onstat -G

IBM Informix Dynamic Server Version 11.50.FC6     -- On-Line -- Up 05:04:17

Global Transaction Identifiers
address          flags  isol    timeout  fID        gtl  bql  data
83246d88         -L--G  COMMIT  0        0          2    4    4D4E4F000000
 1 active, 128 total

D:\Infx\ids1150>
C:\work>esql -nologo xa_tool.ec
IBM Informix CSDK Version 3.50, IBM Informix-ESQL Version 3.50.TC7
xa_tool.c ...

C:\work>xa_tool 0 2 4 4D4E4F000000

Calling xa_open ...
... xa_open finished

        P -- XA_PREPARE
        C -- XA_COMMIT
        R -- XA_ROLLBACK
        F -- XA_FORGET
        Q -- terminate program
Enter Choice:  R
R
Executing XA_ROLLBACK
... XA_ROLLBACK finished

        P -- XA_PREPARE
        C -- XA_COMMIT
        R -- XA_ROLLBACK
        F -- XA_FORGET
        Q -- terminate program
```

```
Enter Choice:  q
Q
Calling xa_close...
... xa_close finished

C:\work>
```

For more information related to the TP/XA library, refer to the *TP/XA Transaction Manager XA Interface Library User Manual* at:

http://publibfp.boulder.ibm.com/epubs/pdf/5193.pdf

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see "How to get IBM Redbooks publications" on page 480. Note that some of the documents that we reference here might be available in softcopy only.

- *Embedding Informix Dynamic Server: An Introduction*, SG24-7666

Other publications

The following publications are also relevant as further information sources:

- *IBM Informix Dynamic Server Administrator's Guide, v11.50*, SC23-7748
- *IBM Informix Guide to SQL: Reference, v11.50*, SC23-7750
- *Embedded SQLJ User's Guide, Version 2.90*, G251-2278
- *IBM Informix Storage Manager Administrator's Guide, v2.2,* G229-6388
- *IBM Informix Security Guide, v11.50*, SC23-7754
- *IBM Informix GLS User's Guide,* G229-6373
- *IBM Informix Dynamic Server Administrator's Reference,* SC23-7749
- *IBM Informix Guide to SQL: Syntax, v11.50*, SC23-7751
- *IBM Informix Guide to SQL: Tutorial*, G229-6427
- *IBM Informix Dynamic Server Performance Guide, v11.50*, SC23-7757
- *IBM Informix High-Performance Loader User's Guide, v11.50,* SC23-9433
- *IBM Informix Migration Guide*, SC23-7758
- *IBM Informix JDBC Driver Programmer's Guide*, SC23-9421
- *IBM Informix ESQL/C Programmer's Manual, v3.50,* SC23-9420
- *IBM Data Server Provider for .NET Programmer's Guide*, SC23-9848

- *IBM Informix Dynamic Server Installation Guide for UNIX, Linux, and Mac OS X,* GC23-7752
- *IBM Informix DB-Access User's Guide*, SC23-9430
- *IBM Informix ODBC Driver Programmer's Manual*, SC23-9423
- *IBM Informix Dynamic Server Installation Guide for Windows*, GC23-7753
- *Guide to Informix MaxConnect, Version 1.1,* G251-0577
- *IBM Informix Backup and Restore Guide, v11.50*, SC23-7756
- *IBM Informix Dynamic Server Enterprise Replication Guide*, v11.50, SC23-7755
- "Expand transaction capabilities with savepoints in Informix Dynamic Server" by Uday B. Kale in IBM developerWorks, 26 March 2009:

  http://www.ibm.com/developerworks/data/library/techarticle/dm-0903idssavepo ints/index.html

Online resources

The following website is also relevant as further information sources:

- IBM Informix Dynamic Server v11.50 Information Center:

  http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp

How to get IBM Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Redbooks

IBM Informix Developer's Handbook

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Informix Developer's Handbook

Learn application development with supported APIs, drivers, and interfaces

Understand Informix supported programming environments

Follow practical examples to develop an Informix application

IBM Informix is a low-administration, easy-to-use, and embeddable database that is ideal for application development. It supports a wide range of development platforms, such as Java, .NET, PHP, and web services, enabling developers to build database applications in the language of their choice. Informix is designed to handle RDBMS data and XML without modification and can be extended easily to handle new data sets.

This IBM Redbooks publication provides fundamentals of Informix application development. It covers the Informix Client installation and configuration for application development environments. It discusses the skills and techniques for building Informix applications with Java, ESQL/C, OLE DB, .NET, PHP, Ruby on Rails, DataBlade, and Hibernate.

The book uses code examples to demonstrate how to develop an Informix application with various drivers, APIs, and interfaces. It also provides application development troubleshooting and considerations for performance.

This book is intended for developers who use IBM Informix for application development. Although some of the topics that we discuss are highly technical, the information in the book might also be helpful for managers or database administrators who are looking to better understand their Informix development environment.

SG24-7884-00 ISBN 0738434701