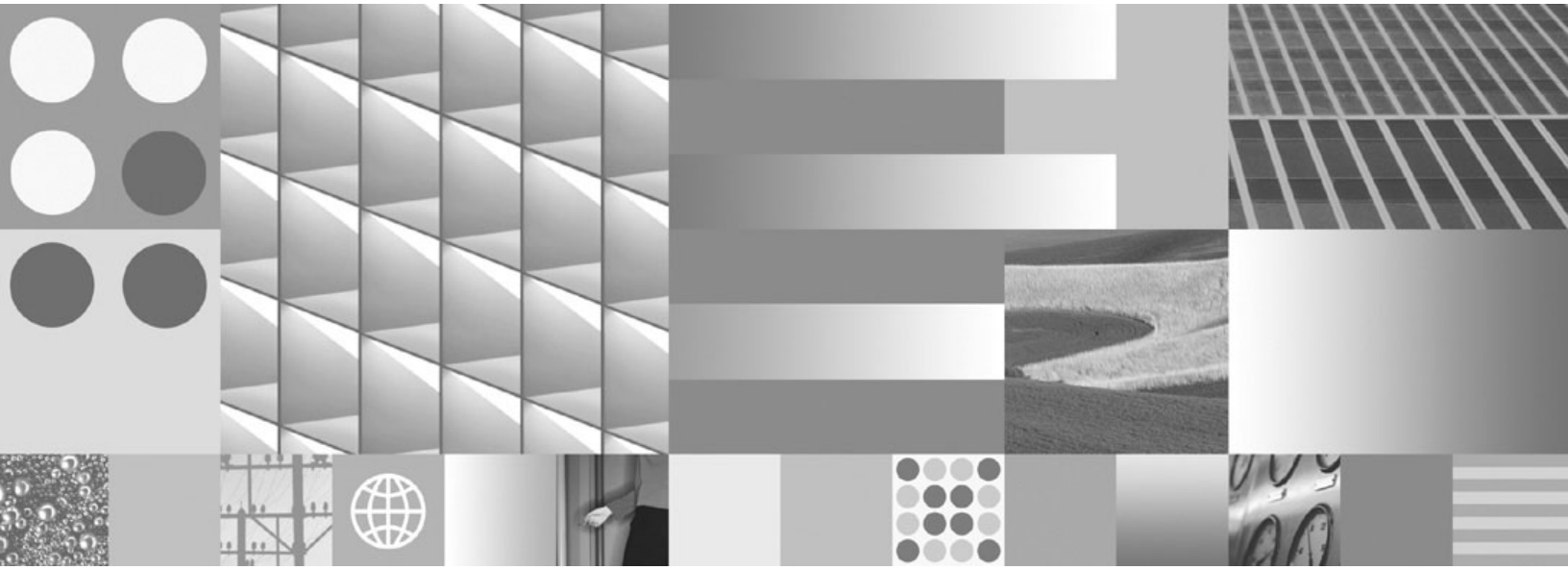
IBM Informix

Version 3.50

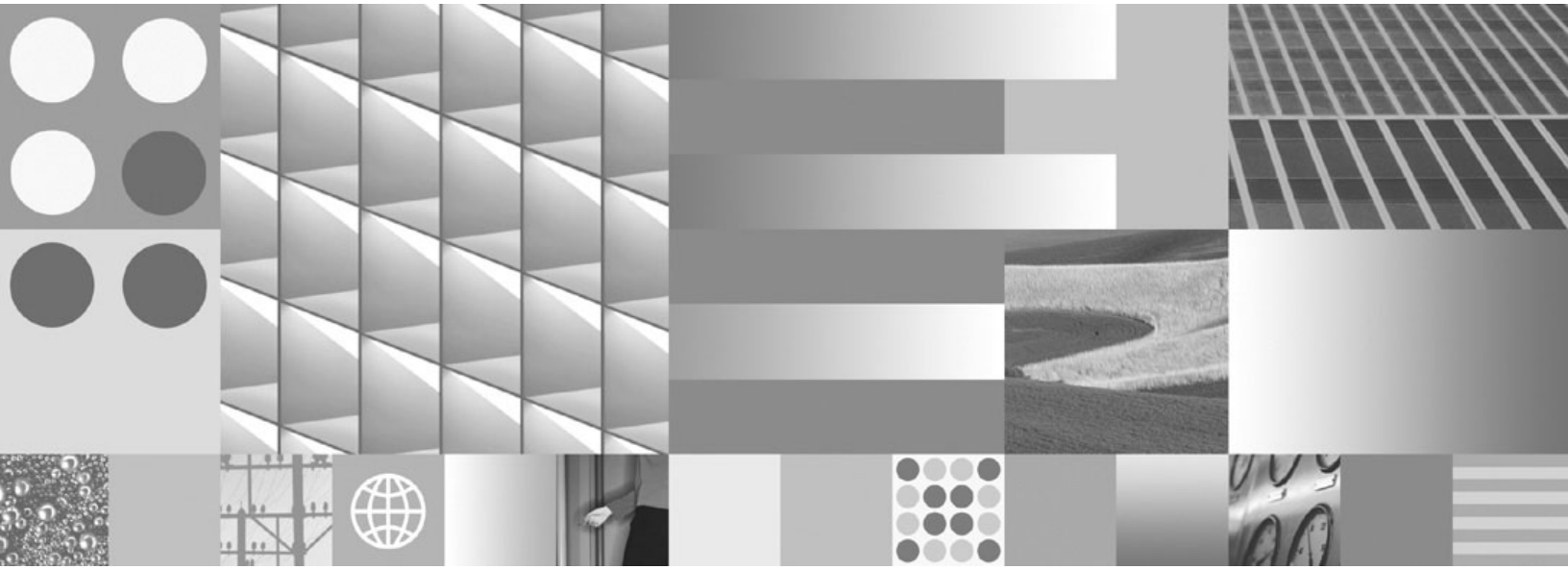


IBM Informix .NET Provider Reference Guide

SC23-9425-01

IBM Informix

Version 3.50



IBM Informix .NET Provider Reference Guide

SC23-9425-01

Note:
Before using this information and the product it supports, read the information in "Notices" on page B-1.

This edition replaces SC23-9425-00.

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this publication should not be interpreted as such.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2003, 2008.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Introduction vii
About This Publication vii
What's New in .NET Provider, Version 3.50 vii
Documentation Conventions viii
Typographical Conventions viii
Feature, Product, and Platform Markup viii
Example Code Conventions ix
Additional Documentation ix
Compliance with Industry Standards x
How to Provide Documentation Feedback x

Chapter 1. Overview of the IBM Informix .NET Provider 1-1
What is the .NET Provider? 1-1
Supported programming environments 1-1
Supported databases 1-2
Visual Studio data access support 1-2
Support for IPv6 1-2
Installing the IBM Informix .NET Provider 1-2
Update the PATH Environment Variable for Microsoft Windows 64-bit Systems 1-3
Preparing the database server 1-3
Overview of the .NET Provider class library 1-3
Thread-safety of provider types 1-4
Namespace requirements 1-4
Connecting to a database 1-5
Reconciling DataSet changes with the database 1-6
The connection pool 1-7
Setting FullTrust permission 1-7
Using ? parameter markers 1-7
Parameter arrays 1-7
Calling stored procedures 1-7
Using IfxProviderFactory objects to write database-independent code 1-8
Using distributed transactions 1-8
Generic Coding with the ADO.NET Common Base Classes 1-8
Error messages 1-9
Tracing 1-9

Chapter 2. Mapping data types 2-1
Retrieving data 2-1
Setting parameter's data types 2-2
Display format of FLOAT, DECIMAL, or MONEY data types 2-3

Chapter 3. Type reference 3-1
Supported Public .NET Interfaces 3-5
Supported Public .NET base classes 3-6
Prototype syntax 3-7
IfxBlob class 3-8
The IfxBlob internal cursor 3-8
Creating an IfxBlob 3-8
IfxBlob public properties 3-9
IfxBlob public methods 3-10
IfxClob class 3-13
The IfxClob internal cursor 3-13
Creating an IfxClob 3-13
IfxClob public properties 3-14
IfxClob public methods 3-15

© Copyright IBM Corp. 2003, 2008

IfxCommand class . . . 3-18
Creating an IfxCommand . . . 3-18
IfxCommand public properties . . . 3-18
IfxCommand public methods . . . 3-19
IfxCommand examples . . . 3-20
IfxCommandBuilder class . . . 3-21
Creating an IfxCommandBuilder . . . 3-22
IfxCommandBuilder public properties . . . 3-22
IfxCommandBuilder public methods . . . 3-22
IfxCommandBuilder examples . . . 3-23
IfxConnection class . . . 3-23
Creating an IfxConnection . . . 3-23
IfxConnection public properties . . . 3-24
IfxConnection public methods . . . 3-27
IfxConnection public events . . . 3-28
Examples . . . 3-28
IfxConnectionStringBuilder class . . . 3-29
Creating an IfxConnectionStringBuilder . . . 3-29
IfxConnectionStringBuilder public properties . . . 3-29
IfxConnectionStringBuilder public methods . . . 3-30
IfxDataAdapter class . . . 3-30
Creating an IfxDataAdapter . . . 3-30
IfxDataAdapter public properties . . . 3-31
IfxDataAdapter public methods . . . 3-32
IfxDataAdapter examples . . . 3-32
IfxDataReader class . . . 3-33
IfxDataReader public properties . . . 3-34
IfxDataReader public methods . . . 3-34
IfxDataReader example . . . 3-35
IfxDataSourceEnumerator class . . . 3-36
Creating an IfxDataSourceEnumerator . . . 3-36
IfxDataSourceEnumerator public properties . . . 3-36
IfxDataSourceEnumerator public methods . . . 3-37
IfxDateTime structure . . . 3-37
Creating an IfxDateTime . . . 3-38
IfxDateTime public properties . . . 3-39
IfxDateTime public methods . . . 3-40
IfxDecimal structure . . . 3-44
Creating an IfxDecimal . . . 3-44
IfxDecimal properties . . . 3-45
IfxDecimal methods . . . 3-45
IfxError class . . . 3-49
IfxError public properties . . . 3-50
IfxErrorCollection class . . . 3-50
IfxErrorCollection public properties . . . 3-50
IfxErrorCollection public methods . . . 3-50
IfxException class . . . 3-50
IfxException public properties . . . 3-50
IfxMonthSpan structure . . . 3-51
Creating an IfxMonthSpan . . . 3-51
IfxMonthSpan public properties . . . 3-51
IfxMonthSpan public methods . . . 3-52
IfxParameter class . . . 3-56
Creating an IfxParameter class . . . 3-56
IfxParameter public properties . . . 3-57
IfxParameter examples . . . 3-57
IfxParameterCollection class . . . 3-58
Creating an IfxParameterCollection . . . 3-58
IfxParameterCollection public properties . . . 3-59
IfxParameterCollection public methods . . . 3-59
IfxProviderFactory class . . . 3-60

IfxProviderFactory public methods . . . 3-60
IfxSmartLOBCreateTimeFlags enumeration . . . 3-61
IfxSmartLOBFileLocation enumeration . . . 3-62
IfxSmartLOBLocator class . . . 3-62
IfxSmartLOBLockMode enumeration . . . 3-62
IfxSmartLOBOpenMode enumeration . . . 3-63
IfxSmartLOBWhence enumeration . . . 3-63
IfxTimeSpan structure . . . 3-63
Creating an IfxTimeSpan . . . 3-64
IfxTimeSpan public properties . . . 3-65
IfxTimeSpan public methods . . . 3-66
IfxTimeUnit enumeration . . . 3-69
IfxTransaction class . . . 3-70
IfxTransaction public properties . . . 3-70
IfxTransaction public methods . . . 3-70
IfxTransaction example . . . 3-70
IfxType enumeration . . . 3-71

Chapter 4. Sample programs . . . 4-1
Demonstration programs . . . 4-1
Examples . . . 4-1
Retrieving a single value . . . 4-1
Retrieving multiple rows . . . 4-2
Executing SQL that does not return data and using a transaction . . . 4-3
Retrieving data into a DataSet . . . 4-3
Using an IfxCommandBuilder object to reconcile changes with the database . . . 4-4
Calling a stored procedure . . . 4-5
Using distributed transactions . . . 4-6
Writing CLOBs to files . . . 4-7

Appendix. Accessibility . . . A-1
Accessibility features for IBM Informix Dynamic Server . . . A-1
Accessibility Features . . . A-1
Keyboard Navigation . . . A-1
Related Accessibility Information . . . A-1
IBM and Accessibility . . . A-1

Notices . . . B-1
Trademarks . . . B-3

Index . . . X-1

Introduction

About This Publication vii
What's New in .NET Provider, Version 3.50 vii
Documentation Conventions viii
Typographical Conventions viii
Feature, Product, and Platform Markup viii
Example Code Conventions ix
Additional Documentation ix
Compliance with Industry Standards x
How to Provide Documentation Feedback x

In This Introduction

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

About This Publication

This publication contains the information you need in order to use the IBM® Informix® .NET Provider to access and manipulate data in IBM Informix databases. This publication assumes you are familiar with the Microsoft® .NET specification, object-oriented programming principles, and using IBM Informix servers and databases.

Microsoft provides information about programming with .NET on its web site. For more information about working with IBM Informix Dynamic Server, see the *Getting Started* publication in your server documentation set.

Windows® and .NET are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

What's New in .NET Provider, Version 3.50

For a comprehensive list of new features for this release, see the *IBM Informix Dynamic Server Getting Started Guide*. The following changes and enhancements are relevant to this publication.

Table 1. What's New in IBM Informix .NET Provider Reference Guide for Version 3.50.xC3

Overview	Reference
The *Single Threaded* and *Skip Parsing* connection parameters can help improve performance for some operations.	See "ConnectionString property" on page 3-24 for more information.

Table 2. What's New in IBM Informix .NET Provider Reference Guide for Version 3.50.xC1

Overview	Reference
Informix BIGINT and BIGSERIAL data types are supported. These data types are similar to the Informix INT8 and SERIAL8 data types, but have performance advantages.	See Table 2-1 on page 2-1 and "IfxType enumeration" on page 3-71.

© Copyright IBM Corp. 2003, 2008

Documentation Conventions

This section describes the following conventions, which are used in the product documentation for IBM Informix Dynamic Server:

- Typographical conventions
- Feature, product, and platform conventions
- Example code conventions

Typographical Conventions

This publication uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	Keywords of SQL, SPL, and some other programming languages appear in uppercase letters in a serif font.
italics	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface	Names of program entities (such as classes, events, and tables), environment variables, file names, path names, and interface elements (such as icons, menu items, and buttons) appear in boldface.
`monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
`KEYSTROKE`	Keys that you are to press appear in uppercase letters in a sans serif font.
>	This symbol indicates a menu item. For example, "Choose **Tools > Options**" means choose the **Options** item from the **Tools** menu.

Technical changes to the text are indicated by special characters depending on the format of the documentation:

HTML documentation

New or changed information is surrounded by blue >> and << characters.

PDF documentation

A plus sign (+) is shown to the left of the current changes. A vertical bar (|) is shown to the left of changes made in earlier shipments.

Feature, Product, and Platform Markup

Feature, product, and platform markup identifies paragraphs that contain feature-specific, product-specific, or platform-specific information. Some examples

of this markup follow:

Dynamic Server

Identifies information that is specific to IBM Informix Dynamic Server

End of Dynamic Server

Windows Only

Identifies information that is specific to the Windows operating system

End of Windows Only

This markup can apply to one or more paragraphs within a section. When an entire section applies to a particular product or platform, this is noted as part of the heading text, for example:

Table Sorting (Windows)

Example Code Conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

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

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

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

Additional Documentation

You can view, search, and print all of the product documentation from the IBM Informix Dynamic Server information center on the Web at http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp.

For additional documentation about IBM Informix Dynamic Server and related products, including release notes, machine notes, and documentation notes, go to the online product library page at http://www.ibm.com/software/data/informix/

pubs/library/. Alternatively, you can access or install the product documentation from the Quick Start CD that is shipped with the product.

Compliance with Industry Standards

The American National Standards Institute (ANSI) and the International Organization of Standardization (ISO) have jointly established a set of industry standards for the Structured Query Language (SQL). IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

How to Provide Documentation Feedback

You are encouraged to send your comments about IBM Informix user documentation by using one of the following methods:

- Send e-mail to docinf@us.ibm.com.
- Go to the Information Center at http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp and open the topic that you want to comment on. Click the feedback link at the bottom of the page, fill out the form, and submit your feedback.

Feedback from both methods is monitored by those who maintain the user documentation of Dynamic Server. The feedback methods are reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact IBM Technical Support. For instructions, see the IBM Informix Technical Support Web site at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Chapter 1. Overview of the IBM Informix .NET Provider

What is the .NET Provider? . . . 1-1
Supported programming environments . . . 1-1
Supported databases . . . 1-2
Visual Studio data access support . . . 1-2
Support for IPv6 . . . 1-2
Installing the IBM Informix .NET Provider . . . 1-2
Update the PATH Environment Variable for Microsoft Windows 64-bit Systems . . . 1-3
Preparing the database server . . . 1-3
Overview of the .NET Provider class library . . . 1-3
Thread-safety of provider types . . . 1-4
Namespace requirements . . . 1-4
Connecting to a database . . . 1-5
Reconciling DataSet changes with the database . . . 1-6
The connection pool . . . 1-7
Setting FullTrust permission . . . 1-7
Using ? parameter markers . . . 1-7
Parameter arrays . . . 1-7
Calling stored procedures . . . 1-7
Using IfxProviderFactory objects to write database-independent code . . . 1-8
Using distributed transactions . . . 1-8
Generic Coding with the ADO.NET Common Base Classes . . . 1-8
Error messages . . . 1-9
Tracing . . . 1-9

What is the .NET Provider?

The IBM Informix .NET Provider is a .NET assembly that lets .NET applications access and manipulate data in IBM Informix databases. It does this by implementing several interfaces in the Microsoft .NET Framework that are used to access data from a database.

Using the IBM Informix .NET Provider is more efficient than accessing an IBM Informix database through either of these two methods:

- Using the Microsoft .NET Framework Data Provider for ODBC along with the IBM Informix ODBC Driver
- Using the Microsoft .NET Framework Data Provider for OLE DB along with the IBM Informix OLE DB Provider

Supported programming environments

The IBM Informix .NET Provider can be used by any application that can be executed by the Microsoft .NET Framework. The following list includes some examples of programming languages that create applications that meet this criteria:

- Visual BASIC .NET
- Visual C# .NET
- Visual J# .NET
- ASP.NET

The IBM Informix .NET Provider runs on all Microsoft Windows platforms that provide full .NET support. If you want to use the IBM Informix .NET Provider that implements ADO.NET 2.0 interfaces, you must have Microsoft .NET Framework

© Copyright IBM Corp. 2003, 2008

Version 2.0 or later installed on your system. The IBM Informix .NET Provider that implements ADO.NET 2.0 interfaces is available with CSDK 3.0 and later releases.

You must have the Microsoft .NET Framework SDK Version 1.1 or later installed on your system. You must have Version 2.90 or later of the IBM Informix Client SDK installed.

Supported databases

You can use IBM Informix .NET Provider to connect to these database servers:

- IBM Informix Dynamic Server (IDS) Version 7.31, Version 9.x, and later.
- IBM Informix Extended Parallel Server (XPS) Version 8.4 and later.

Visual Studio data access support

If you use Microsoft Visual Studio, an IBM Informix .NET Provider add-in is installed automatically when you install the Client SDK. The Visual Studio 2005 add ins are available only with CSDK. The provider from IConnect does not include these add ins. Look under the Data tab of the toolbox for tools that let you configure these IBM Informix.NET Provider types visually:

- IfxConnection
- IfxCommand
- IfxDataAdapter

Support for IPv6

The IBM Informix .NET Provider can use Internet Protocol Version 6 (IPv6) addresses, as well as Internet Protocol Version 4 (IPv4) addresses.

If your system uses IPv6 it is recommended that you use host names in your connection strings instead of using IPv6 format IP addresses. Other than that, no special actions need be taken.

Installing the IBM Informix .NET Provider

The IBM Informix .NET Provider installation registers two strong-named assemblies in the global assembly cache (GAC). The name of the DLL file is IBM.Data.Informix.dll. The assembly version for the IBM Informix .NET Provider 2.0 Framework is different from the assembly version for the 1.1 Framework. The assembly version for IBM Informix .NET Provider 1.1 Framework is frozen at Version 2.81.0.0.

The correct method to verify the version of the .NET assembly file is to see the version information of the file itself, by right-clicking the file, choosing Properties, and then viewing the version tab, which indicates the version of the CSDK. You will see version 2.81.0.0 under IBM.Data.Informix under C:\windows\assembly.

If your application is not running in a debugger it will automatically find the assembly that is registered in the GAC.

The location of the IBM Informix .NET Framework assemblies are shown in Table 1-1 on page 1-3:

Table 1-1. Locations of the .NET Framework Assemblies

Framework that the assembly implements	Location
IBM Informix .NET Provider 1.1 Framework	%INFORMIXDIR%\bin\netf11
IBM Informix .NET Provider 2.0 Framework	%INFORMIXDIR%\bin\netf20

Because of security functions in .NET, you must reference one of these versions instead of the GAC version when running your application in a debugger. If you are using Microsoft Visual Studio .NET, complete the following steps to add the reference:

1. Right click on **References** in the Solution Explorer window.
2. Click **Browse** and navigate to the assembly. The file name of the assembly is IBM.Data.Informix.dll. Select the file from %INFORMIXDIR%\bin\netf20 or %INFORMIXDIR%\bin\netf11 folder, depending upon which provider you want to use.
3. Select the assembly and click **Open**.
4. Click **OK**.

To verify the version of the .NET Assembly File, right-click the file and select **Properties**.

Update the PATH Environment Variable for Microsoft Windows 64-bit Systems

If you run .NET programs on Microsoft Windows 64-bit systems, such as Windows Vista and Windows Server 2003, set your PATH environment variable to include the path to IfxDotNetIntrinsicModule.dll:

- %INFORMIXDIR%/bin/netf11/ if you use the Microsoft .NET Framework Version 1.1
- %INFORMIXDIR%/bin/netf20/ if you use the Microsoft .NET Framework Version 2.0

The DLL is not required on 32-bit Windows operating systems. If you move your application from a 32-bit to a 64-bit Windows operating system, you must update the PATH environment variable or you will receive an error.

Preparing the database server

Before you use the IBM Informix .NET Provider to access databases on a particular database server you must execute the script, cdotnet.sql, against the sysmaster database on that server as the user informix.

Overview of the .NET Provider class library

The IBM Informix .NET Provider supports all of the .NET public classes and base classes that are needed to access an IBM Informix database. “Supported Public .NET Interfaces” on page 3-5 shows the supported public objects, properties, and methods. “Supported Public .NET base classes” on page 3-6 shows the supported public base classes. See your Microsoft .NET Framework SDK documentation for further information.

In the .NET Framework, access to a data source and its data is handled by the ADO.NET classes (ADO.NET stands for ActiveX Data Objects on the .NET

platform). The .NET Framework is a set of services and programs that provide the runtime environment for .NET applications. ADO.NET contains two primary components: the data set classes and the .NET provider classes.

The DataSet object represents a data source in memory (in a disconnected state). .NET applications use DataSet to manipulate data. The DataTable and DataColumn interfaces represent the source table and its columns. The DataRelation interface represents relationships, such as parent-child, between tables.

When you retrieve data from the database, the full result set is retrieved from the server and converted to XML before it is put in a DataSet. The full result set is stored on the client. Therefore, it is recommended that you keep your data sets as small as possible. Remember to use MAX ROWS in your SELECT statements if you do not need to return all data.

The main IBM Informix .NET Provider classes that provide data access are:

- IfxConnection—for connection to a database and management of transactions.
- IfxCommand—for issuing SQL commands.
- IfxDataReader—for reading a forward-only stream of data records.
- IfxTransaction—for controlling transactions.
- IfxDataAdapter—for pushing data into a dataset and for reconciling changes in a dataset with the database.

The following .NET Provider classes let you develop provider-independent code when you use the .NET Provider for the 2.0 framework:

- DbProviderFactory
- DbConnectionStringBuilder
- DbCommand

The IfxDataReader object provides quick retrieval of data from the database. However, the data is read-only and you can only move forward, one record at a time, through the result set. Unlike DataSet objects, IfxDataReader objects do not create temporary tables to hold the data, and so they use less memory.

If data is changed on the client, you might want to apply those changes to the database. You can use the primary key of your database table to ensure that you are updating the correct row in the table. For single-table updates, you can use the IfxCommandBuilder class to facilitate automatic reconciliation of changes in the data set with the database (see "Reconciling DataSet changes with the database" on page 1-6). See "IfxCommandBuilder class" on page 3-21 for more information about the IfxCommandBuilder class.

Thread-safety of provider types

Only static members of an IBM Informix .NET Provider type are thread-safe. No instance of any of the types is guaranteed to be safe when called from multiple threads at the same time.

Namespace requirements

The namespace for the IBM Informix .NET Provider is: IBM.Data.Informix. This means that the full name of the objects in the IBM Informix .NET provider all begin with IBM.Data.Informix. For instance the full name of IfxConnection is IBM.Data.Informix.IfxConnection.

To avoid having to enter the entire namespace for each of the objects you can *import* the namespace. The exact way that you do this depends on your programming language. The C# language uses the keyword `using`. If you are programming in C# you can reference the namespace by including this line at the start of your module:

```
using IBM.Data.Informix;
```

Connecting to a database

You connect to a database using the Open method of an IfxConnection object. You define information about how to connect to the database (such as the machine and server where the database resides) by passing a connection string to the IfxConnection object. The connection string has the form:

attribute=value[;*attribute=value*]...

The brackets ([]) and the ellipsis (...) are not part of the string. They show that attribute/value pairs beyond the first are optional and any number of attribute/value pairs can be included. Separate attribute/value pairs from each other with a semicolon.

The full list of possible attributes is described in section "IfxConnection class" on page 3-23 where the IfxConnection class and its methods and properties are described.

If you are using Microsoft Visual Studio you can create a connection visually:

1. Drag an IfxConnection from the Data tab of the toolbox onto one of your forms.
2. Click in the ConnectionString property of the new IfxConnection object.
3. Click the ellipses (...) button that appears in the ConnectionString text box. A dialog box opens.
4. Fill in the items of the dialog to provide the connection information. Click on the Help button of the dialog box for details on how to use the dialog.

The following fragment shows a simple connection to a database called testdb on an IBM Informix server called testserver that resides on a machine named berry:

```
IfxConnection conn=new IfxConnection("Host=berry; Service=9401;
  Server=testserver;User ID=informix; password=ifxtest;
  Database=testdb");
conn.Open();
```

In addition to the ConnectionString property, an IfxConnection object can also determine connection properties from the Setnet utility, and from the environment or registry (for example, the values of DELIMIDENT and OPTOFC can be determined in this way). If properties are set in the connection string, the IfxConnection object uses those values. If they are not set in the connection string, it uses values set by the environment. For any properties that remain unset, IfxConnection object takes values from the Setnet registry.

Note: In compliance with industry standards, the IBM Informix .NET Provider acts as though DELIMIDENT is set to Y unless you explicitly set it to N.

When your application has finished using the database, close the connection as in the following fragment:

```
conn.Close();
```

Note: Connection string attribute names are case insensitive, but often their values are not.

Reconciling DataSet changes with the database

If you retrieve data from the database using an IfxDataAdapter object and make changes to the data in the data set, the IfxCommandBuilder class allows you to generate automatic INSERT, DELETE, and UPDATE commands to reconcile those changes with the database. "Retrieving data into a DataSet" on page 4-3 includes an example that demonstrates how to use IfxDataAdapter objects. "Using an IfxCommandBuilder object to reconcile changes with the database" on page 4-4 includes an example that demonstrates how to use IfxCommandBuilder objects.

Automatic generation of SQL statements for data reconciliation is initiated when you set the SelectCommand property of an IfxDataAdapter object with the SELECT statement you want to execute. Then, when you create an IfxCommandBuilder object, it automatically generates SQL statements for single-table updates to reconcile changes in the data set with the database. An IfxCommandBuilder object is always associated with an IfxDataAdapter object (in a one-to-one relationship).

The SELECT statement that you execute using the SelectCommand property must return at least one primary key or unique column. If none are present, an InvalidOperation exception is returned, and the reconciliation commands are not generated.

The IfxCommandBuilder object also uses the IfxCommand Connection, CommandTimeout, and Transaction properties for the SELECT statement you are executing (set by the SelectCommand property). If any of these properties are modified, or if the SELECT statement itself is replaced, you should call the IfxCommandBuilder.RefreshSchema method. Otherwise, the InsertCommand, UpdateCommand, and DeleteCommand properties retain their original values.

The IfxCommandBuilder.Dispose method disassociates the IfxCommandBuilder object from the IfxDataAdapter object, and the generated commands are no longer used.

An IfxCommandBuilder object may not generate efficient SQL statements. You can view the commands it generates by using the GetDeleteCommand, GetInsertCommand, and GetUpdateCommand methods.

The following limitations apply to the use of IfxCommandBuilder objects:

- The SELECT statement must retrieve at least one primary key or unique column as part of the query.
- The SELECT statement must refer to a single table; it cannot contain stored procedures or views that contain JOIN operators.
- The SELECT statement must refer to columns that permit read-write operations.
- The IfxCommandBuilder object makes no attempt, nor does it provide any mechanism, to fetch output arguments from the SELECT statement.
- If the CommandText, Connection, CommandTimeout or Transaction properties for the query change, you must execute the IfxCommandBuilder.RefreshSchema method.
- The UPDATE and DELETE commands generated by an IfxCommandBuilder object will not change any row that was modified in the database after the data was read by the SELECT.

- The IfxCommandBuilder object is designed to work with single, unrelated tables. You cannot use IfxCommandBuilder objects to update tables with primary key/foreign key relationships.
- If columns in your SELECT command contain special characters, such spaces, periods, quotation marks or non-alphanumeric characters, you cannot use IfxCommandBuilder objects unless you use the QuotePrefix and QuoteSuffix properties to specify the delimiter for table and column names in the queries it generates.

The IfxDataAdapter, IfxCommandBuilder, and other classes are described in detail and illustrated with examples in Chapter 3, "Type reference," on page 3-1.

The connection pool

Connection pooling allows client applications to reuse connections instead of creating a new one each time the IBM Informix .NET Provider needs to connect to a database.

To make a connection available in the pool, you must close it after your application has finished using the connection. For reuse, a connection must currently be unused and must still be connected to the server.

You use the Pooling, Max Pool Size, Connection Life Time, and Min Pool Size connection string attributes to control the connection pool.

Setting FullTrust permission

In order to use the IBM Informix .NET Provider, calling applications must have FullTrust permission set.

Using ? parameter markers

You can use the question mark symbol (?) to mark a parameter's place in an SQL statement or stored procedure. Because the IBM Informix .NET Provider does not have access to names for these parameters, you must pass them in the correct order. The order you add IfxParameter objects to an IfxParameterCollection object must directly correspond to the position of the placeholder ? symbol for that parameter. You use the ParameterCollection.Add method to add a parameter to a collection.

Parameter arrays

The IBM Informix .NET Provider supports input parameter arrays for UPDATE and INSERT statements. This allows an application to use a single command to specify a row of parameter values and these values can be sent in a single roundtrip to the server.

Calling stored procedures

To use stored procedures in your applications, set the following properties of the IfxCommand object as shown:

- CommandText - set to the name of the stored procedure
- CommandType - set to StoredProcedure

You can use the IfxCommandBuilder.DeriveParameters method to retrieve information about parameters for stored procedures.

If a stored procedure returns a value, your application must add a parameter for this to the parameter collection used by the IfxCommand object.

The section "Calling a stored procedure" on page 4-5 includes an example that shows how to run a stored procedure and read any results that it returns.

Using IfxProviderFactory objects to write database-independent code

Starting with .NET Provider Version 2, you may use the IfxProviderFactory class to write database-independent code.

For more information on the IfxProviderFactory class, see 3-60.

Using distributed transactions

Your application can enlist a connection for distributed transactions by setting the Enlist connection string attribute to true, yes, or 1. It is advisable to set the Pooling connection string attribute to true, yes, or 1 while working with distributed transactions.

"Using distributed transactions" on page 4-6 includes an example of how to use distributed transactions with your application.

Note: Distributed transactions are supported through the Microsoft Distributed Transaction Coordinator (MS DTC). The MS DTC components are required to call some unmanaged code, which can affect the level of security available and potentially degrade performance.

Generic Coding with the ADO.NET Common Base Classes

The .NET Framework version 2.0 features a namespace called `System.Data.Common`, which features a set of base classes that can be shared by any .NET data provider. This facilitates a generic ADO.NET database application development approach, featuring a constant programming interface. The main classes in the Informix .NET Data Provider for .NET Framework 2.0 are inherited from the `System.Data.Common` base classes. As a result, generic ADO.NET applications will work with Dynamic Server databases through the Informix .NET Data provider.

The following C# code demonstrates a generic approach to establishing a database connection.

```
DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.Informix");
 DbConnection conn = factory.CreateConnection();
 DbConnectionStringBuilder sb = factory.CreateConnectionStringBuilder();

 if( sb.ContainsKey( "Database" ) )
 {
    sb.Remove( "database" );
    sb.Add( "database", "SAMPLE" );
 }

 conn.ConnectionString = sb.ConnectionString;

 conn.Open();
```

The DbProviderFactory object is the point where any generic ADO.NET application begins. This object creates generic instances of .NET data provider objects, such as connections, data adapters, commands, and data readers, which work with a specific database product. In the case of the example above, the "IBM.Data.Informix" string passed into the GetFactory method uniquely identifies the Informix .NET Data Provider, and results in the initialization of a DbProviderFactory instance that creates database provider object instances specific to the Informix .NET Data Provider. The DbConnection object can connect to Dynamic Server databases, just as a IfxConnection object, which is actually inherited from DbConnection. Using the DbConnectionStringBuilder class, you can determine the connection string keywords for a data provider, and generate a custom connection string. The code in the above example checks if a keyword named "database" exists in the Informix .NET Data Provider, and if so, generates a connection string to connect to the SAMPLE database.

Error messages

Error messages from the IBM Informix server are represented as IBM Informix .NET Provider exceptions.

Tracing

An application can enable tracing by setting the IFXDOTNETTRACE environment variable.

Setting	Meaning
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

- 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

Trace information is written to the file you set using the IFXDOTNETTRACEFILE environment variable.

Chapter 2. Mapping data types

Retrieving data 2-1
Setting parameter's data types 2-2
Display format of FLOAT, DECIMAL, or MONEY data types 2-3

In this chapter

This chapter describes how data types are mapped between IBM Informix databases and the .NET Framework. It includes:

- How data types are mapped when you retrieve data from the database using IfxDataReader and IfxDataAdapter objects
- How a parameter's data type is mapped (when you use IfxParameter objects)

Retrieving data

Table 2-1 shows each Informix data type, the recommended type to store that data type in, and the .NET Framework data type that it best fits in. The recommended type should be used when accessing data through an IfxDataReader. The best-fit .NET type is the type that an IfxDataAdapter object will use when it fills a DataSet object.

You can use types other than those shown, for example you can use the IfxDataReader.GetString method to get any data type that can be stored in an IBM Informix database. The types recommended are the most efficient and least likely to change the value.

Table 2-1. Best-fit types for retrieving IBM Informix data types

Informix data type	Recommended type	Best-fit native .NET data type
BIGINT	Int64	Int64
BIGSERIAL	Int64	Int64
BLOB	IfxBlob	Byte[]
BOOLEAN	Boolean	Boolean
BYTE	Byte[]	Byte[]
CHAR	String	String
CHAR(1)	String	String
CLOB	IfxClob	Byte[]
DATE	IfxDateTime	DateTime
DATETIME	IfxDateTime	DateTime
DECIMAL(p<=28) fixed scale	IfxDecimal	Decimal
DECIMAL (p<=28) floating point	IfxDecimal	Double
DECIMAL (p>28)	IfxDecimal	String
DOUBLE	Double	Double
FLOAT	Double	Double
IDSSECURITYLABEL	Int64[]	Int64[]

© Copyright IBM Corp. 2003, 2008

Table 2-1. Best-fit types for retrieving IBM Informix data types (continued)

Informix data type	Recommended type	Best-fit native .NET data type
INTEGER	Int32	Int32
INT8	Int64	Int64
INTERVAL, year-month	IfxMonthSpan	String
INTERVAL, day-fraction	IfxTimeSpan	TimeSpan
LVARCHAR	String	String
MONEY	IfxDecimal	As for Decimal with same precision
NCHAR	String	String
REAL	Float	Float
SERIAL	nt32	Int32
SERIAL8	Int64	Int64
SMALLFLOAT	Float	Float
TEXT	String	String
VARCHAR	String	String

Note: For the format of Informix data types, DECIMAL, MONEY, DATETIME, and INTERVAL returned usingIfxDataReader.GetString method see the section about the Literal Row segment in *IBM Informix Guide to SQL: Syntax*.

Note: The ROW and TEXT types and the collection types, LIST, MULTISET, SET, can be mapped to a string literal .NET Framework type and accessed with the IfxDataReader.GetString method. The format for the string is documented in the *IBM Informix Guide to SQL: Syntax*, in the section about the Literal Row segment.

In order to make the expression of any nested string literals simpler, a leading quotation mark is not returned in the string. A single-quotation mark, rather than a double-quotation mark, is used to begin and end any string literals embedded in the ROW type. This is to avoid confusion if a double-quotation mark might be used as a delimited identifier.

Setting parameter's data types

It is recommended that your application sets a parameter's type as an Informix type whenever possible (using the IfxType argument of IfxParameter constructor). "IfxType enumeration" on page 3-71 shows the IfxType enumeration.

You can specify a parameter type as a .NET DbType instead, and the IBM Informix .NET Provider will infer the Informix type as best it can. The .NET DbType specifies the data type of a Parameter object of a .NET Framework data provider. Some DbType types, such as GUID, do not map to any Informix type, and an error will be returned. Some DbType types, such as AnsiString, can map to several Informix types, such as VARCHAR, TEXT, or BLOB; you must be aware that the IBM Informix .NET Provider may not choose the data type you intend.

If you do not specify either an Informix data type or a .NET DbType, the IBM Informix .NET Provider attempts to infer an Informix data type from the value

itself. For example, if the value is 4, the provider maps this to an INTEGER data type. Relying on these inferred mappings can lead to unexpected results.

Display format of FLOAT, DECIMAL, or MONEY data types

The display format of the Informix FLOAT, DECIMAL, or MONEY data types is specified by the values of the DBMONEY or CLIENT_LOCALE environment variables.

The DBMONEY environment variable takes precedence over the CLIENT_LOCALE environment variable. If you do not set DBMONEY, the locale setting (CLIENT_LOCALE) is used to format the value. By default, DBMONEY is set to a dollar sign and a period ($.), and CLIENT_LOCALE is set to US English (en_us.CP1252). For example, if you set DBMONEY=Pt, the separator becomes a comma (,). A decimal value of 169.0 will then be formatted with a comma: 169,0.

The Informix .NET Provider determines display format using the following precedence:

1. Connection string
2. Environment
3. Registry (SetNet settings)

The values in the connection string override all other settings. For more information, see the *IBM Informix Guide to SQL: Reference*.

Chapter 3. Type reference

Supported Public .NET Interfaces . . . 3-5
Supported Public .NET base classes . . . 3-6
Prototype syntax . . . 3-7
IfxBlob class . . . 3-8
The IfxBlob internal cursor . . . 3-8
Creating an IfxBlob . . . 3-8
IfxBlob constructors . . . 3-8
IfxBlob public properties . . . 3-9
IfxBlob public methods . . . 3-10
IfxBlob.Close . . . 3-10
IfxBlob.FromFile . . . 3-10
IfxBlob.GetLocator . . . 3-11
IfxBlob.Lock . . . 3-11
IfxBlob.Open . . . 3-11
IfxBlob.Read . . . 3-11
IfxBlob.Release . . . 3-11
IfxBlob.Seek . . . 3-12
IfxBlob.ToFile . . . 3-12
IfxBlob.Truncate . . . 3-12
IfxBlob.Unlock . . . 3-12
IfxBlob.Write . . . 3-12
IfxClob class . . . 3-13
The IfxClob internal cursor . . . 3-13
Creating an IfxClob . . . 3-13
IfxClob constructors . . . 3-13
IfxClob public properties . . . 3-14
IfxClob public methods . . . 3-15
IfxClob.Close . . . 3-15
IfxClob.FromFile . . . 3-15
IfxClob.GetLocator . . . 3-16
IfxClob.Lock . . . 3-16
IfxClob.Open . . . 3-16
IfxClob.Read . . . 3-16
IfxClob.Release . . . 3-16
IfxClob.Seek . . . 3-17
IfxClob.ToFile . . . 3-17
IfxClob.Truncate . . . 3-17
IfxClob.Unlock . . . 3-17
IfxClob.Write . . . 3-17
IfxCommand class . . . 3-18
Creating an IfxCommand . . . 3-18
IfxCommand constructors . . . 3-18
IfxCommand public properties . . . 3-18
IfxCommand public methods . . . 3-19
IfxCommand.Cancel . . . 3-19
IfxCommand.CreateParameter . . . 3-19
IfxCommand.ExecuteNonQuery . . . 3-19
IfxCommand.ExecuteReader . . . 3-20
IfxCommand.ExecuteScalar . . . 3-20
IfxCommand.Prepare . . . 3-20
IfxCommand examples . . . 3-20
IfxCommandBuilder class . . . 3-21
Creating an IfxCommandBuilder . . . 3-22
IfxCommandBuilder constructors . . . 3-22
IfxCommandBuilder public properties . . . 3-22

© Copyright IBM Corp. 2003, 2008

IfxCommandBuilder public methods . . . 3-22
IfxCommandBuilder.DeriveParameters . . . 3-22
IfxCommandBuilder.GetDeleteCommand . . . 3-22
IfxCommandBuilder.GetInsertCommand . . . 3-22
IfxCommandBuilder.GetUpdateCommand . . . 3-22
IfxCommandBuilder.RefreshSchema . . . 3-23
IfxCommandBuilder examples . . . 3-23
IfxConnection class . . . 3-23
Creating an IfxConnection . . . 3-23
IfxConnection constructors . . . 3-24
IfxConnection public properties . . . 3-24
ConnectionString property . . . 3-24
UserDefinedTypeFormat property . . . 3-26
IfxConnection public methods . . . 3-27
IfxConnection.BeginTransaction . . . 3-27
IfxConnection.ChangeDatabase . . . 3-28
IfxConnection.Close . . . 3-28
IfxConnection.CreateCommand . . . 3-28
IfxConnection.GetIfxBlob . . . 3-28
GetIfxClob . . . 3-28
IfxConnection.EnlistTransaction . . . 3-28
IfxConnection.Open . . . 3-28
IfxConnection public events . . . 3-28
Examples . . . 3-28
IfxConnectionStringBuilder class . . . 3-29
Creating an IfxConnectionStringBuilder . . . 3-29
IfxConnectionStringBuilder public properties . . . 3-29
IfxConnectionStringBuilder public methods . . . 3-30
IfxConnectionStringBuilder.Add . . . 3-30
IfxConnectionStringBuilder.AppendKeyValuePair . . . 3-30
IfxConnectionStringBuilder.Clear . . . 3-30
IfxConnectionStringBuilder.ContainsKey . . . 3-30
IfxConnectionStringBuilder.EquivalentTo . . . 3-30
IfxConnectionStringBuilder.Remove . . . 3-30
IfxConnectionStringBuilder.ToString . . . 3-30
IfxConnectionStringBuilder.TryGetValue . . . 3-30
IfxDataAdapter class . . . 3-30
Creating an IfxDataAdapter . . . 3-30
IfxDataAdapter constructors . . . 3-30
IfxDataAdapter public properties . . . 3-31
IfxDataAdapter public methods . . . 3-32
IfxDataAdapter.Fill . . . 3-32
IfxDataAdapter.FillSchema . . . 3-32
IfxDataAdapter.GetFillParameters . . . 3-32
IfxDataAdapter.Update . . . 3-32
IfxDataAdapter examples . . . 3-32
IfxDataReader class . . . 3-33
IfxDataReader public properties . . . 3-34
IfxDataReader public methods . . . 3-34
IfxDataReader.Close . . . 3-34
IfxDataReader.GetBoolean . . . 3-34
IfxDataReader.GetByte . . . 3-34
IfxDataReader.GetBytes . . . 3-34
IfxDataReader.GetChar . . . 3-34
IfxDataReader.GetChars . . . 3-34
IfxDataReader.GetData . . . 3-34
IfxDataReader.GetDateTime . . . 3-34
IfxDataReader.GetDataTypeName . . . 3-34
IfxDataReader.GetDecimal . . . 3-34
IfxDataReader.GetDouble . . . 3-34
IfxDataReader.GetFieldType . . . 3-34

IfxDataReader.GetFloat . . . 3-35
IfxDataReader.GetGuid . . . 3-35
IfxDataReader.GetInt16 . . . 3-35
IfxDataReader.GetInt32 . . . 3-35
IfxDataReader.GetInt64 . . . 3-35
IfxDataReader.GetName . . . 3-35
IfxDataReader.GetOrdinal . . . 3-35
IfxDataReader.GetSchemaTable . . . 3-35
IfxDataReader.GetString . . . 3-35
IfxDataReader.GetTimeSpan . . . 3-35
IfxDataReader.GetValue . . . 3-35
IfxDataReader.GetValues . . . 3-35
IfxDataReader.IsDBNull . . . 3-35
IfxDataReader.NextResult . . . 3-35
IfxDataReader.Read . . . 3-35
IfxDataReader example . . . 3-35
IfxDataSourceEnumerator class . . . 3-36
Creating an IfxDataSourceEnumerator . . . 3-36
IfxDataSourceEnumerator public properties . . . 3-36
IfxDataSourceEnumerator public methods . . . 3-37
IfxDataSourceEnumerator.GetDataSources . . . 3-37
IfxDateTime structure . . . 3-37
Creating an IfxDateTime . . . 3-38
IfxDateTime constructors . . . 3-38
IfxDateTime public properties . . . 3-39
IfxDateTime public methods . . . 3-40
IfxDateTime.Add . . . 3-40
IfxDateTime.AddDays . . . 3-40
IfxDateTime.AddMilliseconds . . . 3-40
IfxDateTime.AddMinutes . . . 3-40
IfxDateTime.AddMonths . . . 3-41
IfxDateTime.AddSeconds . . . 3-41
IfxDateTime.AddYears . . . 3-41
IfxDateTime.Compare . . . 3-41
IfxDateTime.CompareTo . . . 3-42
IfxDateTime.DaysInMonth . . . 3-42
IfxDateTime.Equals . . . 3-42
IfxDateTime.GetHashCode . . . 3-42
IfxDateTime.GreaterThan . . . 3-42
IfxDateTime.GreaterThanOrEqual . . . 3-42
IfxDateTime.LessThan . . . 3-43
IfxDateTime.LessThanOrEqual . . . 3-43
IfxDateTime.NotEquals . . . 3-43
IfxDateTime.Parse . . . 3-43
IfxDateTime.ToString . . . 3-44
IfxDecimal structure . . . 3-44
Creating an IfxDecimal . . . 3-44
IfxDecimal constructors . . . 3-44
IfxDecimal properties . . . 3-45
IfxDecimal methods . . . 3-45
IfxDecimal.Abs . . . 3-45
IfxDecimal.Add . . . 3-45
IfxDecimal.Ceiling . . . 3-45
IfxDecimal.Clone . . . 3-46
IfxDecimal.Compare . . . 3-46
IfxDecimal.CompareTo . . . 3-46
IfxDecimal.Divide . . . 3-46
IfxDecimal.Equals . . . 3-46
IfxDecimal.Floor . . . 3-46
IfxDecimal.GetHashCode . . . 3-47
IfxDecimal.GreaterThan . . . 3-47

IfxDecimal.GreaterThanOrEqual . . . 3-47
IfxDecimal.LessThan . . . 3-47
IfxDecimal.LessThanOrEqual . . . 3-47
IfxDecimal.Max . . . 3-47
IfxDecimal.Min . . . 3-47
IfxDecimal.Modulo . . . 3-48
IfxDecimal.Multiply . . . 3-48
IfxDecimal.Negate . . . 3-48
IfxDecimal.NotEquals . . . 3-48
IfxDecimal.Parse . . . 3-48
IfxDecimal.Remainder . . . 3-48
IfxDecimal.Round . . . 3-48
IfxDecimal.Subtract . . . 3-49
IfxDecimal.ToString . . . 3-49
IfxDecimal.Truncate . . . 3-49
IfxError class . . . 3-49
IfxError public properties . . . 3-50
IfxErrorCollection class . . . 3-50
IfxErrorCollection public properties . . . 3-50
IfxErrorCollection public methods . . . 3-50
IfxErrorCollection.GetEnumerator . . . 3-50
IfxException class . . . 3-50
IfxException public properties . . . 3-50
IfxMonthSpan structure . . . 3-51
Creating an IfxMonthSpan . . . 3-51
IfxMonthSpan constructors . . . 3-51
IfxMonthSpan public properties . . . 3-51
IfxMonthSpan public methods . . . 3-52
IfxMonthSpan.Add . . . 3-52
IfxMonthSpan.Compare . . . 3-52
IfxMonthSpan.CompareTo . . . 3-53
IfxMonthSpan.Divide . . . 3-53
IfxMonthSpan.Duration . . . 3-53
IfxMonthSpan.Equals . . . 3-53
IfxMonthSpan.GetHashCode . . . 3-53
IfxMonthSpan.GreaterThan . . . 3-53
IfxMonthSpan.GreaterThanOrEqual . . . 3-54
IfxMonthSpan.LessThan . . . 3-54
IfxMonthSpan.LessThanOrEqual . . . 3-54
IfxMonthSpan.Multiply . . . 3-54
IfxMonthSpan.Negate . . . 3-54
IfxMonthSpan.NotEquals . . . 3-54
IfxMonthSpan.Parse . . . 3-54
IfxMonthSpan.Subtract . . . 3-55
IfxMonthSpan.ToString . . . 3-55
IfxParameter class . . . 3-56
Creating an IfxParameter class . . . 3-56
IfxParameter constructors . . . 3-56
IfxParameter public properties . . . 3-57
IfxParameter examples . . . 3-57
IfxParameterCollection class . . . 3-58
Creating an IfxParameterCollection . . . 3-58
IfxParameterCollection public properties . . . 3-59
IfxParameterCollection public methods . . . 3-59
IfxParameterCollection.Add . . . 3-59
IfxParameterCollection.Clear . . . 3-59
IfxParameterCollection.Contains . . . 3-59
IfxParameterCollection.CopyTo . . . 3-59
IfxParameterCollection.GetEnumerator . . . 3-59
IfxParameterCollection.IndexOf . . . 3-60
IfxParameterCollection.Insert . . . 3-60

IfxParameterCollection.Remove . . . 3-60
IfxParameterCollection.RemoveAt . . . 3-60
IfxProviderFactory class . . . 3-60
IfxProviderFactory public methods . . . 3-60
IfxProviderFactory.CreateConnectionStringBuilder . . . 3-60
IfxProviderFactory.CreateConnection . . . 3-60
IfxProviderFactory.CreateCommand . . . 3-60
IfxProviderFactory.CreateParameter . . . 3-61
IfxProviderFactory.CreateCommandBuilder . . . 3-61
IfxProviderFactory.CreateDataAdapter . . . 3-61
IfxProviderFactory.CreateDataSourceEnumerator . . . 3-61
IfxProviderFactory.CreatePermission (PermissionState) . . . 3-61
IfxSmartLOBCreateTimeFlags enumeration . . . 3-61
IfxSmartLOBFileLocation enumeration . . . 3-62
IfxSmartLOBLocator class . . . 3-62
IfxSmartLOBLockMode enumeration . . . 3-62
IfxSmartLOBOpenMode enumeration . . . 3-63
IfxSmartLOBWhence enumeration . . . 3-63
IfxTimeSpan structure . . . 3-63
Creating an IfxTimeSpan . . . 3-64
IfxTimeSpan constructors . . . 3-64
IfxTimeSpan public properties . . . 3-65
IfxTimeSpan public methods . . . 3-66
IfxTimeSpan.Add . . . 3-66
IfxTimeSpan.Compare . . . 3-66
IfxTimeSpan.CompareTo . . . 3-66
IfxTimeSpan.Divide . . . 3-67
IfxTimeSpan.Duration . . . 3-67
IfxTimeSpan.Equals . . . 3-67
IfxTimeSpan.GetHashCode . . . 3-67
IfxTimeSpan.GreaterThan . . . 3-67
IfxTimeSpan.GreaterThanOrEqual . . . 3-67
IfxTimeSpan.LessThan . . . 3-67
IfxTimeSpan.LessThanOrEqual . . . 3-68
IfxTimeSpan.Negate . . . 3-68
IfxTimeSpan.NotEquals . . . 3-68
IfxTimeSpan.Parse . . . 3-68
IfxTimeSpan.ToString . . . 3-69
IfxTimeUnit enumeration . . . 3-69
IfxTransaction class . . . 3-70
IfxTransaction public properties . . . 3-70
IfxTransaction public methods . . . 3-70
IfxTransaction.Commit . . . 3-70
IfxTransaction.Rollback . . . 3-70
IfxTransaction example . . . 3-70
IfxType enumeration . . . 3-71

All the classes described in this section belong in the namespace IBM.Data.Informix. For example, the full identification of the IfxConnection class is IBM.Data.Informix.IfxConnection.

Supported Public .NET Interfaces

Table 3-2 on page 3-6 shows the Microsoft .NET interfaces that are implemented by the IBM Informix .NET Provider for the 1.1 .NET framework and the 2.0 .NET framework.

See your Microsoft .NET Framework SDK documentation for further information about the Microsoft public interfaces and classes. If the IBM Informix .NET Provider does not support a particular .NET class or method, that class or method is implemented as no-operation.

Table 3-1. Interfaces implemented by IBM Informix .NET Provider classes

Class	Extends	Description
IfxCommand	IDbCommand	Represents a query or command that is run when the application is connected to the database
IfxCommandBuilder	DbCommandBuilder	Generates single-table INSERT, DELETE, and UPDATE commands that reconcile changes made in a data set with the associated IBM Informix database
IfxConnection	IDbConnection	Represents an application's unique session with a data source
IfxDataAdapter	IDbDataAdapter	Enables an application to execute SQL commands against the database, fill data sets, and reconcile changes in the data set with the database
IfxDataReader	IDDataReader	Allows forward-only, read-only, access to a stream of data from the database
IfxError		Represents an instance of a warning or an error generated by the IBM Informix database
IfxErrorCollection	ICollection	Represents a collection of IfxError objects in an IfxException object
IfxException		Represents an exception that is generated when a warning or error is returned by an IBM Informix database
IfxParameter	IDbDataParameter	Implements a parameter to a command and maps it to a column within a data set
IfxParameterCollection	IDbParameterCollection	Implements multiple parameters to a command and maps them to columns within a data set
IfxTransaction	IDbTransaction	Represents a local transaction

Supported Public .NET base classes

Table 3-1 shows the Microsoft .NET base classes that are implemented by the IBM Informix .NET Provider for the .NET 2.0 framework.

See your Microsoft .NET Framework SDK documentation for further information about the Microsoft public classes. If the IBM Informix .NET Provider does not support a particular .NET base class, that class is implemented as no-operation.

Table 3-2. Base classes implemented by IBM Informix .NET Provider classes

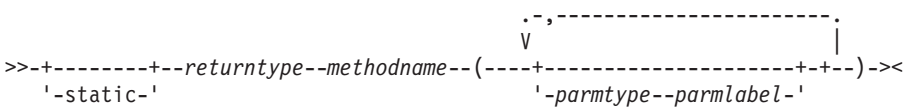
Class	Base class	Description
IfxCommand	DbCommand	Represents a query or command that is run when the application is connected to the database
IfxCommandBuilder	DBCommandBuilder	Generates single-table INSERT, DELETE, and UPDATE commands that reconcile changes made in a data set with the associated IBM Informix database
IfxConnection	DbConnection	Represents an application's unique session with a data source

Table 3-2. Base classes implemented by IBM Informix .NET Provider classes (continued)

Class	Base class	Description
IfxConnectionStringBuilder	DbConnectionStringBuilder	Provides the base class from which the strongly typed connection string builders (OdbcConnectionStringBuilder and SQLconnectionStringBuilder) derive.
IfxDataAdapter	DbDataAdapter	Enables an application to execute SQL commands against the database, fill data sets, and reconcile changes in the data set with the database
IfxDataReader	DbDataReader	Allows forward-only, read-only, access to a stream of data from the database
IfxDataSourceEnumerator	DbDataSourceEnumerator	Lets data providers to obtain a list of data sources.
IfxParameter	DbParameter	Implements a parameter to a command and maps it to a column within a data set
IfxParameterCollection	DbParameterCollection	Implements multiple parameters to a command and maps them to columns within a data set
IfxProviderFactory	DbProviderFactory	Represents a set of methods that you can use to create instances of a provider's implementation of the data source classes.
IfxTransaction	DbTransaction	Represents a local transaction

Prototype syntax

Because the objects of the IBM Informix .NET Provider can be used in many different programming languages, the prototypes of the methods are given in this publication using a pseudo code. The syntax of the pseudo code is as follows:



returntype
This is the type of the object that is returned. If the method returns nothing then this will be `void`.

methodname
The name of the method.

parmtype
What type of object is expected at this position in the argument list.

parmlabel
A name for the parameter. This is only used as a convenience when referring to the parameter in the text. This parameter name will always be italicized, even in the text.

If the static keyword is present it means that the method is callable without creating an instance of the class of which it is a part. In place of the instance of the class use the name of the class itself. In Visual Basic this is called a Shared method. In C# it is called a static method.

Example: The IfxDecimal.Floor method is static and accepts a single IfxDecimal. That means that if mydec is an instance of IfxDecimal you can call floor on it like this: IfxDecimal.Floor(mydec). But you cannot call it like this: mydec.Floor(mydec).

The syntax for prototypes of constructors is the same as the syntax given above except that `static` and *returntype* are not used.

IfxBlob class

An IfxBlob represents a BLOB, which is a large block of binary data that allows random access of its contents. You can treat a BLOB in much the same way you treat an operating system file. You can read or write to certain positions in the file without reading or writing through all of the data up to that position.

BLOBs and CLOBs, are both types of smart large objects. 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 as it is moved to or from the database server.

The IfxBlob internal cursor

Each IfxBlob has an internal pointer to a position in the BLOB. This is referred to in this publication as the *cursor* of the instance.

The position of the cursor when an IfxBlob object is opened depends on the mode in which it is opened. The section "IfxSmartLOBOpenMode enumeration" on page 3-63 lists the possible modes.

After a read or a write the cursor is moved to the character after the last one affected by the operation. The method IfxBlob.Seek allows you to set the cursor position explicitly.

Creating an IfxBlob

You can get IfxBlob objects from these methods:

- IfxConnection.GetIfxBlob
- IfxDataReader.GetIfxBlob

You can also create an IfxBlob with a constructor.

IfxBlob constructors

- `IfxBlob( IfxConnection` *`connection`* `)`

Creates a new IfxBlob that is associated with *connection*.

IfxBlob public properties

These are the public properties of the IfxBlob object.

Table 3-3. IfxBlob public properties

Property	Type	Access Notes	Description
EstimatedSize	System.Int64		Gets or sets the estimated final size of the BLOB. You can set this if you have a good estimate of what the final size will be. The database server's optimizer can then use that information. Do not set this unless you have a good idea of the final size. Setting it too large will cause wasted resources on the database server.
ExtentSize	System.Int32		Gets or sets the next extent size that the database server will use when allocating disk space for this BLOB. Only applications that encounter severe storage fragmentation should ever set the allocation extent size.
Flags	System.Int32		Gets or sets the flags for this BLOB. To interpret this value compare it to the members of the IfxSmartLOBCreateTimeFlags enumeration. See "IfxSmartLOBCreateTimeFlags enumeration" on page 3-61 for details.
IsNull	System.Boolean	read-only	Returns true if the instance is null; otherwise it returns false.
IsOpen	System.Boolean	read-only	Returns true if the instance is open; otherwise it returns false.
LastAccessTime	System.Int32	read-only	The system time on the database server (rounded to the second) at which the BLOB was last accessed. This information is only available if KeepAccessTime is set in the Flags property.

Table 3-3. IfxBlob public properties (continued)

Property	Type	Access Notes	Description
LastChangeTime	System.Int32	read-only	The system time on the database server (rounded to the second) at which the status of the BLOB was last changed. Updating, changing ownership, and changes in the number of references are all changes in status.
LastModificationTime	System.Int32	read-only	The system time on the database server (rounded to the second) at which the BLOB was last written to.
MaxBytes	System.Int64		Get or set the maximum size for this BLOB. The database server will not let the BLOB be larger than this value.
Null	IfxBlob	read-only static	An IfxBlob object that has a null value.
Position	System.Int64		Returns the current position of the cursor in the BLOB. The value is the number of bytes from the first byte of the BLOB.
ReferenceCount	System.Int32	read-only	Returns the number of rows in the database that currently contain a reference to this BLOB.
SBSpace	System.String		Gets or sets the name of the sbspace in which the BLOB is stored.
Size	System.Int64	read-only	Gets the current size of the BLOB in bytes.

IfxBlob public methods

IfxBlob.Close

- `void IfxBlob.Close()`

Closes the instance.

IfxBlob.FromFile

- `void IfxBlob.FromFile( System.String` *`filename`*`, System.Boolean` *`appendToSmartLOB`*`, IfxSmartLOBFileLocation` *`fileLocation`* `)`

Writes the operating system file *filename* into the BLOB. If *appendToSmartLOB* is true the file is written to the end of the BLOB. If it is false it overwrites the current contents of the BLOB.

The value of *fileLocation* indicates whether the file indicated in *filename* is located on the client or the server. Server side files are not currently supported.

IfxBlob.GetLocator

- `IfxSmartLOBLocator IfxBlob.GetLocator()`

Returns the IfxSmartLOBLocator associated with this instance.

IfxBlob.Lock

- `void IfxBlob.Lock( System.Int64 smartLOBOffset, IfxSmartLOBWhence whence, System.Int64 range, IfxSmartLOBLockMode lockMode )`

Use this method to place a lock on a portion of the BLOB. The type of lock (exclusive or shared) is determined by *lockMode*.

The lock is placed on a group of contiguous bytes that is *range* bytes long. The start of the locked range is determined by the values of *smartLOBOffset* and *whence*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxBlob.Open

- `void IfxBlob.Open( IfxSmartLOBOpenMode mode )`

Before an instance of IfxBlob can be read from or written to it must be opened using this method. The value of *mode* determines what sort of access will be allowed to the BLOB. See "IfxSmartLOBOpenMode enumeration" on page 3-63 for a description of the different modes.

IfxBlob.Read

- `System.Int64 IfxBlob.Read( char[] buff )`
- `System.Int64 IfxBlob.Read( char[] buff, System.Int64 buffOffset, System.Int64 numBytesToRead, System.Int64 smartLOBOffset, IfxSmartLOBWhence whence )`

Reads characters into *buff* from the BLOB represented by this instance. The number returned is how many bytes were successfully read into *buff*.

If only *buff* is given, then the BLOB is read into it starting at element 0. This version of the method will not write past the end of the array *buff*. The BLOB is truncated if it is longer than the buffer. The read begins at current cursor position in the BLOB.

If the other arguments are provided then exactly *numBytesToRead* bytes are read into *buff* starting at element *buffOffset*. An error is returned if this method is asked to write outside the bounds of the array.

Before the read occurs the cursor is moved according to the values of *whence* and *smartLOBOffset*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxBlob.Release

- `void IfxBlob.Release()`

Use this method to free database server resources used by this instance if the instance was never read from or written to a database. Do not call this method if you have written the BLOB to a database or if it was created because of a read from a database.

After calling this method do not use the instance.

IfxBlob.Seek

- `System.Int64 IfxBlob.Seek( System.Int64 offset, IfxSmartLOBWhence whence )`

Changes the position of the cursor within the BLOB. The value returned is the new position of the cursor from the start of the BLOB.

The new position of the cursor is determined by the values of *offset* and *whence*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxBlob.ToFile

- `System.String IfxBlob.ToFile( System.String filename, System.IO.FileMode mode, IfxSmartLOBFileLocation fileLocation )`

Writes the contents of the BLOB to an operating system file named *filename*. The value of *fileLocation* determines whether the file will be written on the client or on the server. Server side files are not currently supported.

The value of *mode* determines how the output file is opened. Look up System.IO.FileMode in the *.NET Framework Class Library* for details on the available modes.

IfxBlob.Truncate

- `void IfxBlob.Truncate( System.Int64 offset )`

Deletes everything in the BLOB past the position *offset*.

IfxBlob.Unlock

- `void IfxBlob.Unlock( System.Int64 smartLOBOffset, IfxSmartLOBWhence whence, System.Int64 range )`

Use this method to remove all locks placed on a certain range of bytes in the BLOB. The size of the range that is unlocked is *range* bytes.

The values of *smartLOBOffset* and *whence* determine where the range starts. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxBlob.Write

- `System.Int64 IfxBlob.Write( char[] buff )`
- `System.Int64 IfxBlob.Write( char[] buff, System.Int64 buffOffset, System.Int64 numBytesToWrite, System.Int64 smartLOBOffset, IfxSmartLOBWhence whence )`

Writes bytes from *buff* to the BLOB represented by this instance. The number returned is how many bytes were successfully written.

If only *buff* is given, then the entire array is written to the BLOB starting at the BLOB's current cursor position.

If the other arguments are provided then exactly *numBytesToWrite* bytes are written to the BLOB from *buff* starting at array element *buffOffset*. An error is returned if *buffOffset* is outside the bounds of the array..

Before the write is performed the cursor is moved according to the values of *whence* and *smartLOBOffset*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

If the write starts beyond the current end of the BLOB then it will be padded with bytes that have a value of 0 from the current end to the point where the write begins.

IfxClob class

An IfxClob represents a CLOB, which is a large block of character data that allows random access of its contents. You can treat a CLOB in much the same way you treat an operating system file. You can read or write to certain positions in the file without reading or writing through all of the data up to that position.

CLOBs and BLOBs, are both types of smart large objects. Both types share many of the same methods. A CLOB is different from a BLOB in that the data in it is treated as characters instead of bytes. This means that it is subject to codeset conversion and other functions of the Global Language System (GLS). If a multibyte character set is being used then one character may require more than one byte to represent it in the CLOB.

The IfxClob internal cursor

Each IfxClob tracks an internal pointer to a position in the CLOB. This is referred to as the *cursor* of the instance.

The position of the cursor when an IfxClob object is opened depends on the mode in which it is opened. See "IfxSmartLOBOpenMode enumeration" on page 3-63 for a list of the possible modes.

After a read or a write the cursor is moved to the next character after the last one affected by the operation. The method IfxClob.Seek allows you to set the cursor position explicitly.

Creating an IfxClob

You can get IfxClob objects from these methods:

- IfxConnection.GetIfxClob
- IfxDataReader.GetIfxClob

You can also create an IfxClob with a constructor:

IfxClob constructors

- `IfxClob( IfxConnection connection )`

Creates a new IfxClob that is associated with *connection*.

IfxClob public properties

These are the public properties of the IfxClob object.

Table 3-4. IfxClob public properties

Property	Type	Access Notes	Description
EstimatedSize	System.Int64		Gets or sets the estimated final size of the CLOB. You can set this if you have a good estimate of what the final size will be. The database server's optimizer can then use that information. Do not set this unless you have a good idea of the final size. Setting it too large will cause wasted resources on the database server.
ExtentSize	System.Int32		Gets or sets the next extent size that the database server will use when allocating disk space for this BLOB. Only applications that encounter severe storage fragmentation should ever set the allocation extent size.
Flags	System.Int32		Gets or sets the flags for this CLOB. To interpret this value compare it to the members of the IfxSmartLOBCreateTimeFlags enumeration. See "IfxSmartLOBCreateTimeFlags enumeration" on page 3-61 for details.
IsNull	System.Boolean	read-only	Returns true if the instance is null; otherwise it returns false.
IsOpen	System.Boolean	read-only	Returns true if the instance is open; otherwise it returns false.
LastAccessTime	System.Int32	read-only	The system time on the database server (rounded to the second) at which the CLOB was last accessed. This information is only available if KeepAccessTime is set in the Flags property.

Table 3-4. IfxClob public properties (continued)

Property	Type	Access Notes	Description
LastChangeTime	System.Int32	read-only	The system time on the database server (rounded to the second) at which the status of the CLOB was last changed. Updating, changing ownership, and changes in the number of references are all changes in status.
LastModificationTime	System.Int32	read-only	The system time on the database server (rounded to the second) at which the CLOB was last written to.
MaxBytes	System.Int64		Get or set the maximum size, in bytes, for this CLOB. The database server will not let the CLOB be larger than this value.
Null	IfxBlob	read-only static	An IfxBlob object that has a null value.
Position	System.Int64		Returns the current position of the cursor in the CLOB. The value is the number of bytes from the first byte of the CLOB.
ReferenceCount	System.Int32	read-only	Returns the number of rows in the database that currently contain a reference to this CLOB.
SBSpace	System.String		Gets or sets the name of the sbspace in which the BLOB is stored.
Size	System.Int64	read-only	Gets the current size of the BLOB in bytes.

IfxClob public methods

IfxClob.Close

- `void IfxClob.Close( IfxSmartLOBOpenMode` *`mode`* `)`

Closes the instance.

IfxClob.FromFile

- `void FromFile( System.String` *`filename`*`, System.Boolean` *`appendToSmartLOB`*`, IfxSmartLOBFileLocation` *`fileLocation`* `)`

Writes the operating system file *filename* into the CLOB. If *appendToSmartLOB* is true the file is written to the end of the CLOB. If it is false it overwrites the current contents of the CLOB.

The value of *fileLocation* indicates whether the file indicated in *filename* is located on the client or the server. Server side files are not currently supported.

IfxClob.GetLocator

- `IfxSmartLOBLocator GetLocator()`

Returns the IfxSmartLOBLocator associated with this instance.

IfxClob.Lock

- `void Lock( System.Int64 smartLOBOffset, IfxSmartLOBWhence whence, System.Int64 range, IfxSmartLOBLockMode lockMode )`

Use this method to place a lock on a portion of the CLOB. The type of lock (exclusive or shared) is determined by *lockMode*.

The lock is placed on a group of contiguous characters that is *range* characters long. The start of the locked range is determined by the values of *smartLOBOffset* and *whence*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxClob.Open

- `void IfxClob.Open( IfxSmartLOBOpenMode mode )`

Before an instance of IfxClob can be read from or written to it must be opened using this method. The value of *mode* determines what sort of access will be allowed to the CLOB. See "IfxSmartLOBOpenMode enumeration" on page 3-63 for a description of the different modes.

IfxClob.Read

- `System.Int64 IfxClob.Read( char[] buff )`
- `System.Int64 IfxClob.Read( char[] buff, System.Int64 buffOffset, System.Int64 numCharsToRead, System.Int64 smartLOBOffset, IfxSmartLOBWhence whence )`

Reads characters into *buff* from the CLOB represented by this instance. The number returned is how many bytes were successfully read into *buff*.

If only *buff* is given, then the CLOB is read into it starting at element 0. This version of the method will not write past the end of the array *buff*. The CLOB is truncated if it is longer than the buffer. The read begins at the current cursor position of the CLOB.

If the other arguments are provided then exactly *numCharsToRead* characters are read into *buff* starting at element *buffOffset*. An error is returned if this method is asked to write outside the bounds of the array.

Before the read occurs the cursor is moved according to the values of *whence* and *smartLOBOffset*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxClob.Release

- `void IfxClob.Release()`

Use this method to free database server resources used by this instance if the instance was never read from or written to a database. Do not call this method if you have written the CLOB to a database or if it was created because of a read from a database.

After calling this method do not use the instance.

IfxClob.Seek

- `System.Int64 IfxClob.Seek( System.Int64 offset, IfxSmartLOBWhence whence )`

Changes the position of the cursor within the CLOB. The value returned is the new value of IfxClob.Position.

The new position of the cursor is determined by the values of *offset* and *whence*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxClob.ToFile

- `System.String IfxClob.ToFile( System.String filename, System.IO.FileMode mode, IfxSmartLOBFileLocation fileLocation )`

Writes the contents of the CLOB to an operating system file named *filename*. The value of *fileLocation* determines whether the file will be written on the client or on the server. Server side files are not currently supported.

The value of *mode* determines how the output file is opened. Look up System.IO.FileMode in the *.NET Framework Class Library* for details on the available modes.

IfxClob.Truncate

- `void IfxClob.Truncate( System.Int64 offset )`

Deletes everything past *offset* bytes from the start of the CLOB.

IfxClob.Unlock

- `void IfxClob.Unlock( System.Int64 smartLOBOffset, IfxSmartLOBWhence whence, System.Int64 range )`

Use this method to remove all locks placed on a certain range of characters in the CLOB. The size of the range that is unlocked is *range* characters.

The values of *smartLOBOffset* and *whence* determine where the range starts. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

IfxClob.Write

- `System.Int64 IfxClob.Write( char[] buff )`
- `System.Int64 IfxClob.Write( char[] buff, System.Int64 buffOffset, System.Int64 numCharsToWrite, System.Int64 smartLOBOffset, IfxSmartLOBWhence whence )`

Writes characters from *buff* to the CLOB represented by this instance. The number returned is how many characters were successfully written.

If only *buff* is given, then the entire array is written to the CLOB starting at the current cursor position.

If the other arguments are provided then exactly *numCharsToWrite* characters are written to the CLOB from *buff* starting at array element *buffOffset*. An error is returned if *buffOffset* is outside the bounds of the array.

Before the write is performed the cursor is moved according to the values of *whence* and *smartLOBOffset*. How these values interact is describe in the section "IfxSmartLOBWhence enumeration" on page 3-63.

If the write starts beyond the current end of the CLOB then it will be padded with values of 0 from the current end to the point where the write begins.

IfxCommand class

The IfxCommand class represents an SQL statement that is to be executed in the database.

Creating an IfxCommand

You can create an IfxCommand by using the constructors or by using these methods of other objects:

- `IfxConnection.CreateCommand` (See page 3-28.)

To write provider-independent code, you can use the CreateCommand() method of the DbProviderFactory class to create a provider-specific instance of DbCommand. This capability is included in the .NET 2.0 framework.

IfxCommand constructors

- `IfxCommand()`
- `IfxCommand( System.String cmdText )`
- `IfxCommand( System.String cmdText, IfxConnection connection )`
- `IfxCommand( System.String cmdText, IfxConnection connection, IfxTransaction transaction )`

If *cmdText* is given it is used as the SQL statement of the command. The *connection* and *transaction* will be used when the command is executed if they are given.

IfxCommand public properties

The following table shows the public properties of the IfxCommand class.

Table 3-5. IfxCommand Properties

Property	Type	Description
CommandText	System.String	Gets or sets the text command to run against the data source. The CommandType property is used to interpret this property. All ODBC escape sequence syntax that the IBM Informix ODBC Driver supports is also supported by the IBM Informix .NET Provider.
CommandTimeout	System.Int32	Gets or sets the wait time before terminating the attempt to execute a command and generating an error.
CommandType	System.Data.CommandType	Indicates how the CommandText property is interpreted. The possible values of the CommandType property are described after the table.
Connection	IfxConnection	Gets or sets the IfxConnection object used by this IfxCommand object.
Parameters	IfxParameterCollection	Gets the IfxParameterCollection object.

Table 3-5. IfxCommand Properties (continued)

Property	Type	Description
Transaction	IfxTransaction	Gets or sets the transaction in which the IfxCommand object executes.
UpdatedRowSource	System.Data.UpdateRowSource	Gets or sets how command results are applied to the DataRow when used by the IfxDataAdapter.Update method. The possible values of the UpdatedRowSource property are described after the table.

The CommandType property can have any of the following values:

- StoredProcedure—The name of a stored procedure.
- TableDirect—When the CommandType property is set to TableDirect, the CommandText property should be set to the name of the table or tables to be accessed. If any table names contain special characters, you might need to dereference them, for example, by using escape-character syntax or including qualifying characters. All rows and columns of the named tables are returned when you call the ExecuteNonQuery, ExecuteScalar, or ExecuteReader methods of the IfxCommand class. To access multiple tables, use a comma delimited list, without spaces or padding, that contains the names of the tables to access. When the CommandText property specifies multiple tables, a join of those tables is returned.
- Text—An SQL text command (the default)

The UpdatedRowSource property can have any of the following values:

- Both—Both the output parameters and the first returned row are mapped to the changed row in the DataSet object.
- FirstReturnedRecord—The data in the first returned row is mapped to the changed row in the DataSet object.
- None—Any returned parameters or rows are ignored.
- OutputParameters—Output parameters are mapped to the changed row in the DataSet object.

IfxCommand public methods

IfxCommand.Cancel

- `void IfxCommand.Cancel()`

Attempts to cancel the execution of a command. If the attempt to cancel fails, no exception is generated.

IfxCommand.CreateParameter

- `IfxParameter IfxCommand.CreateParameter()`

Creates a new instance of an IfxParameter object.

IfxCommand.ExecuteNonQuery

- `System.Int32 IfxCommand.ExecuteNonQuery()`

Executes an SQL statement against the IfxConnection object. For UPDATE, INSERT, and DELETE statements the return value is the number of rows affected; for all other statements, it is -1. Returns the InvalidOperationException error if the connection does not exist or is not open.

IfxCommand.ExecuteReader

- `IfxDataReader IfxCommand.ExecuteReader()`
- `IfxDataReader IfxCommand.ExecuteReader( System.DataCommandBehavior` *`behavior`* `)`

Executes the command in the CommandText property against the IfxConnection object and builds an IfxDataReader object. The IfxDataReader object is built using the command behavior in *behavior*:

- CloseConnection—When the command is executed, the IfxConnection object is closed when the associated IfxDataReader object is closed.
- Default—The query can return multiple result sets. Execution of the query can affect the database state. The default sets no CommandBehavior flags.
- KeyInfo—The query returns column and primary key information. The query is executed without any locking on the selected rows.
- SchemaOnly—The query returns column information only and does not affect the database state.
- SequentialAccess—Provides a way for the IfxDataReader object to handle rows that contain columns with large binary values. Rather than loading the entire row, the SequentialAccess parameter enables the IfxDataReader object to load data as a stream. You can then use the IfxDataReader.GetBytes or IfxDataReader.GetChars method to specify a byte location to start the read operation, and to specify a limited buffer size for the data being returned. Specifying the SequentialAccess parameter does not limit you to reading the columns in the order they are returned. However, after you have read past a location in the returned stream of data, you can no longer read data from the IfxDataReader object at or before that location.
- SingleResult—The query returns a single result set. Execution of the query can affect the database state.
- SingleRow—The query is expected to return a single row. Execution of the query can affect the database state. If you expect your SQL statement to return only one row, specifying the SingleRow parameter can improve application performance.

IfxCommand.ExecuteScalar

- `System.Object IfxCommand.ExecuteScalar()`

Executes the query, and returns the first column of the first row in the resultset returned by the query. Extra columns or rows are ignored.

IfxCommand.Prepare

- `void IfxCommand.Prepare()`

Creates a prepared (or compiled) version of the command against the database. If the CommandType property is set to TableDirect, this method does nothing.

IfxCommand examples

The following example fills a data set, adds new customer information records, and then updates the database with the changes.

```
IfxDataAdapter idap = new IfxDataAdapter("select * from customer",con);
DataSet ds = new DataSet("customer");
idap.Fill(ds,"customer");
DataRow drow = ds.Tables["customer"].NewRow();
drow["lname"]="";
ds.Tables["customer"].Rows.Add(drow);
idap.InsertCommand = new IfxCommand();
idap.InsertCommand.CommandType = CommandType.Text;
idap.InsertCommand.CommandText = "execute procedure add_cust(?,?,?)";
idap.InsertCommand.Connection = con;
IfxParameter iparam1 = idap.InsertCommand.CreateParameter();
IfxParameter iparam2 = idap.InsertCommand.CreateParameter();
IfxParameter iparam3 = idap.InsertCommand.CreateParameter();
    iparam1.ParameterName = "fname";
    iparam1.Value = "Hoopla";
    iparam2.ParameterName = "lname";
    iparam2.Value = "MAuie";
    iparam3.ParameterName = "company";
    iparam3.Value = "Fredonia";
    idap.InsertCommand.Parameters.Add(iparam1);
    idap.InsertCommand.Parameters.Add(iparam2);
    idap.InsertCommand.Parameters.Add(iparam3);
    //Inform the command object that the update
    //results in data being returned and it must be
    //updated against the changed row in the
    //dataset. The source of the data is in the
    //first returned row
    idap.InsertCommand.UpdatedRowSource= UpdateRowSource.FirstReturnedRecord;
    idap.RowUpdated += new IfxRowUpdatedEventHandler(OnRowUpdated);
    idap.InsertCommand.Connection.Open();
    idap.Update(ds,"customer");
```

IfxCommandBuilder class

The IfxCommandBuilder class automatically generates single-table INSERT, DELETE, and UPDATE commands that are used to reconcile changes made in a data set with the associated instance of an IBM Informix database. An IfxCommandBuilder object is always associated with an IfxDataAdapter object (in a one-to-one relationship). The IfxDataAdapter object uses IfxCommand objects to execute SQL commands against the database, fill data sets, and reconcile changes in the data set with the database. Automatic generation of SQL statements for data reconciliation is initiated when you set the SelectCommand property of the IfxDataAdapter object. The SelectCommand property gets or sets an SQL SELECT statement to be run against the database. Then, when you create an IfxCommandBuilder object, it automatically generates SQL statements for single-table updates to reconcile changes in the data set with the database. (The IfxCommandBuilder object registers itself as a listener for RowUpdating events of the IfxDataAdapter object.)

For more information about using IfxCommandBuilder objects, see “Reconciling DataSet changes with the database” on page 1-6.

Creating an IfxCommandBuilder

Use the constructor to create an IfxCommandBuilder.

IfxCommandBuilder constructors

- `IfxCommandBuilder()`
- `IfxCommandBuilder( IfxDataAdapter` *`adapter`* `)`

Initializes a new instance of the IfxCommandBuilder class optionally associated with an IfxDataAdapter object.

IfxCommandBuilder public properties

The following table shows the public properties of the IfxCommandBuilder class.

Table 3-6. IfxCommandBuilder Properties

Property	Type	Description
ConflictOption		Specifies which ConflictOption is to be used by the IfxCommandBuilder.
DataAdapter	IfxDataAdapter	Gets or sets the IfxDataAdapter object for which the SQL statements are generated.
QuotePrefix	System.String	Gets or sets the beginning character to use when specifying IBM Informix server object names, (for example, tables or columns), that contain characters such as spaces. QuotePrefix should only be set to a quotation mark, not to an apostrophe or an empty or null string.
QuoteSuffix	System.String	Gets or sets the ending character to use when specifying IBM Informix server object names, (for example, tables or columns), that contain characters such as spaces. QuoteSuffix should only be set to a quotation mark, not to an apostrophe or an empty or null string.

IfxCommandBuilder public methods

IfxCommandBuilder.DeriveParameters

- `void IfxCommandBuilder.DeriveParameters( IfxCommand` *`command`* `)`

Retrieves information about parameters for the stored procedures specified by an IfxCommand object and overwrites the IfxParameterCollection object with this information.

IfxCommandBuilder.GetDeleteCommand

- `IfxCommand IfxCommandBuilder.GetDeleteCommand()`

Gets the automatically generated IfxCommand object required to perform deletions on the database when an application calls the IfxDataAdapter.Update method.

IfxCommandBuilder.GetInsertCommand

- `IfxCommand IfxCommandBuilder.GetInsertCommand()`

Gets the automatically generated IfxCommand object required to perform inserts on the database when an application calls the IfxDataAdapter.Update method.

IfxCommandBuilder.GetUpdateCommand

- `IfxCommand IfxCommandBuilder.GetUpdateCommand()`

Gets the automatically generated IfxCommand object required to perform updates on the database when an application calls the IfxDataAdapter.Update method.

IfxCommandBuilder.RefreshSchema

- `void IfxCommandBuilder.RefreshSchema`

Refreshes the database schema information used to generate INSERT, UPDATE, or DELETE statements.

IfxCommandBuilder examples

The first example shows you how to perform an update using an IfxCommandBuilder object.

```
// IfxConnection -- con
DataSet ds = new DataSet();
string sql = "select fname, lname from customer";
IfxDataAdapter da = new IfxDataAdapter(sql,con);
//Build new CommandBuilder
IfxCommandBuilder ifxbuilder = new IfxCommandBuilder(da);
Con.Open();
Da.Fill(ds,"customer");
//code to modify data in DataSet goes here
ds.Tables[0].Rows[0].ItemArray[0] = "William";
//the following line will fail without the IfxCommandBuilder
//as we have not explicitly set an UpdateCommand in the DataAdapter
da.Update(ds,"customer");
```

This example shows how to retrieve information about parameters for stored procedures.

```
// IfxConnection - con
IfxCommand cmd = new IfxCommand("SP_GETUSERINFO",con);
IfxCommandBuilder cb = new IfxCommandBuilder();
Con.Open();
Cb.DeriveParameters(cmd);
foreach (IfxParameter param in cmd.Parameters)
{
    Console.WriteLine(param.ParameterName);
}
con.Close();
```

IfxConnection class

The IfxConnection class represents a unique session with a data source, for example, a network connection to an IBM Informix server. This class cannot be inherited.

Creating an IfxConnection

Some methods of other objects create IfxConnection objects implicitly. To create one explicitly use one of its constructors.

To write provider-independent code, you can use the CreateConnection() method of the DbProviderFactory class to create a provider-specific instance of DbCommand. This capability is included in the .NET 2.0 framework.

IfxConnection constructors

- `IfxConnection( )`
- `IfxConnection( System.String` *`connectionString`* `)`

Initializes a new instance of the IfxConnection class using the information in the *connectionString* parameter, if provided.

IfxConnection public properties

The following table shows the public properties of the IfxConnection class.

Table 3-7. IfxConnection Properties

Property	Type	Description
ClientLocale	System.String	Gets or sets the locale used by the application.
ConnectionString	System.String	Gets or sets the string used to open a database. See "ConnectionString property" for more information.
ConnectionTimeout	System.Int32	Gets the time (in seconds) to wait while trying to establish a connection before terminating the attempt and generating an error.
Database	System.String	Gets the name of the current database or the database to be used after a connection is open.
DatabaseLocale	System.String	Gets the locale of the database. (Not valid if the connection is not open.)
FetchBufferSize	System.Int32	Gets or sets the default data transport buffer size used by commands created using this connection. Setting this property does not affect commands already created.
GetActiveConnectionsCount	System.Int32	Gets the number of opened, in-use connections.
GetIdleConnectionsCount	System.Int32	Gets the number of opened, unused connections.
PacketSize	System.Int32	Same as FetchBufferSize. The two settings are semantically equivalent; changes in one are reflected in the other.
ServerVersion	System.String	Gets a string containing the version of the instance of the IBM Informix server to which the client is connected.
State	System.Data.Connection.State	Gets the current state of the connection.
UserDefinedTypeFormat	System.String	Sets the mapping of user-defined types to either DbType.String or DbType.Binary. See "UserDefinedTypeFormat property" on page 3-26 for more information.

ConnectionString property

The value of the ConnectionString property is a connection string that includes the source database name and the parameters you need to establish the connection. The default value is an empty string. The only attribute that is mandatory in all situations is Server.

The following table shows the connection string attributes.

Table 3-8. Connection String Attributes

Attribute	Description	Default Value
Client Locale, Client_Locale	The language locale used on the client side of the client-server connection.	en_us.CP1252 (Windows)
Connection Lifetime	When a connection is returned to the pool, the creation time of the connection is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by connection lifetime.	0
Database, DB	The name of the database within the server instance. If no database is specified, a server-only connection is created. You can switch a server-only connection to a database connection by using the ChangeDatabase method. If you use DATABASE... or CREATE DATABASE... statements, you must manage their execution fully, because the IBM Informix .NET Provider does not automatically recognize when these commands are issued. Using these statements without proper management can lead to unexpected results.	
Database Locale, DB_LOCALE	The language locale of the database.	en_US.819
DELIMIDENT	When set to true or y for yes, 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. You can disable automatic enlistment in existing transactions by specifying Enlist=false as a connection string parameter.	true
Exclusive, XCL	The EXCLUSIVE keyword opens the database in exclusive mode and prevents access by anyone but the current user. If another user has already opened the database, exclusive access is denied, an error is returned, and the database is not opened. Valid values are No, 0, Yes, or 1.	No
Host	The name or IP address of the machine on which the Informix server is running. Required.	localhost
Max Pool Size	The maximum number of connections allowed in the pool.	100
Min Pool Size	The minimum number of connections allowed in the pool.	0
Optimize OpenFetchClose, OPTOFC	Reduces the number of round trips to the server for result-set queries. Recommended only for forward-only retrieval of data.	
Packet Size, Fetch Buffer Size, FBS	The size in bytes of the buffers used to send data to or from the server.	32767
Password, PWD	The password associated with the User ID. Required if the client machine or user account is not trusted by the host. Prohibited if a User ID is not given.	
Persist Security Info	When set to false, security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values, including the password.	’false’
Pooling	When set to true, the IfxConnection object is drawn from the appropriate pool, or if necessary, it is created and added to the appropriate pool.	’true’
Protocol, PRO	The communication protocol used between the IBM Informix .NET Provider and the database server.	

Table 3-8. Connection String Attributes (continued)

Attribute	Description	Default Value
Server	The name or alias of the instance of the Informix server to which to connect. Required.	
Service	The service name or port number through which the server is listening for connection requests.	
Single Threaded	If your application is single threaded, you might have better performance with this property. Do not use this option in an XA/MSDTC environment.	'false'
Skip Parsing	You can avoid SQL parsing overhead by setting this value to 'true'. However, you must be certain that your queries are correct, otherwise an error will result.	'false'
User ID, UID	The login account. Required, unless the client machine is trusted by the host machine.	

You can only set the ConnectionString property when the connection is closed. Some of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, except when an error is detected. In this case, none of the properties are updated. IfxConnection properties return those settings contained in the ConnectionString as well as default values or values gathered elsewhere.

Resetting the ConnectionString on a closed connection resets *all* connection string values and related properties, including the password. For example, if you set a connection string that includes ″Database=superstores″, and then reset the connection string to ″Server=myServer″, the Database property is no longer set to superstores.

The connection string is parsed immediately after being set. If errors in syntax are found when parsing, a runtime exception, ArgumentException, is returned. Other errors can be found only when an attempt is made to open the connection. If an attribute name occurs more than once in the connection string, the value associated with the last occurrence is used.

The IBM Informix .NET Provider ConnectionString is not identical to an IBM Informix ODBC connection string. The connection string that is returned is the same as the one set by the user. Neither the ODBC 'Driver' attribute or the OLE DB 'Provider' attribute are supported.

If you set the Persist Security Info attribute to false (the default), if the connection has ever been opened, the returned connection string will not contain any security information. If the connection has *not* been opened, the returned connection string *does* contain security information, regardless of the setting of Persist Security Info. If you set the Persist Security Info attribute to true, the returned connection string contains security information.

UserDefinedTypeFormat property

The UserDefinedTypeFormat property of IfxConnection and IfxCommand sets the mapping of user-defined types to either DbType.String or DbType.Binary. Use this property instead of FetchExtendedTypesAs.

To access user-defined types as String objects, set the UserDefinedTypeFormat attribute or the UserDefinedTypeFormat property to `string`, "", or null. UDT

columns and parameters are mapped to DbType.String. The shorthand, UDTFormat, is also a valid connection string attribute. These settings are not case-sensitive.

To access user-defined types as Byte[] objects, set the UserDefinedTypeFormat attribute or the UserDefinedTypeFormat property to `bytes`. UDT columns and parameters are mapped to DbType.Binary. The IfxType property of a parameter or column is not affected.

The following table shows what the IfxDataReader access methods, GetBytes() and GetString(), return depending on the setting of the UserDefinedTypeFormat property.

Table 3-9. Results for the UserDefinedTypeFormat Setting with IfxDataReader Access Methods

UserDefinedTypeFormat Setting	Operation	Result
string	GetBytes()	Invalid cast exception
string	GetString()	Returns a string
bytes	GetBytes()	Returns bytes
bytes	GetString()	Returns the binary value as a hexadecimal string

When an IfxCommand object is bound to a connection, the object takes the UserDefinedTypeFormat property of that connection. Later changes to the connection setting of the property do not affect the IfxCommand object. Use one of the following ways to associate a command with a connection:

- IfxConnection.CreateCommand()
- IfxCommand.Connection_set()
- IfxCommand.DbConnection_set()
- IfxCommand(string cmdText, IfxConnection connection)
- IfxCommand(string cmdText, IfxConnection connection, IfxTransaction transaction)

You can set the UserDefinedTypeFormat property of an IfxCommand independently from the UserDefinedTypeFormat property of its connection, but you cannot set it during the following times:

- When executing a command
- Between the first call of an IfxDataReader.Read() method and the closing of that data reader.

IfxConnection public methods

IfxConnection.BeginTransaction

- `IfxTransaction IfxConnection.BeginTransaction()`
- `IfxTransaction IfxConnection.BeginTransaction( System.Data.IsolationLevel` *`isoLevel`* `)`

Begins a database transaction.

IfxConnection.ChangeDatabase

- `void IfxConnection.ChangeDatabase( System.String value )`

Changes the current database for an open IfxConnection object.

IfxConnection.Close

- `void IfxConnection.Close()`

Closes the connection to the database.

IfxConnection.CreateCommand

- `IfxCommand IfxConnection.CreateCommand()`

Creates and returns an IfxConnection object associated with the connection.

IfxConnection.GetIfxBlob

- `IfxBlob IfxConnection.GetIfxBlob()`

Returns an IfxBlob structure based on this connection.

GetIfxClob

- `IfxClob IfxConnection.GetIfxClob()`

Returns an IfxClob structure based on this connection.

IfxConnection.EnlistTransaction

- `void IfxConnection.EnlistTransaction()`

Enlists in the specified transaction as a distributed transaction.

IfxConnection.Open

- `void IfxConnection.Open()`

Opens a database connection with the settings specified by the ConnectionString property of the IfxConnection object.

IfxConnection public events

The following table shows the public events of the IfxConnection class.

Table 3-10. IfxConnection Events

Event	Description
Disposed	Adds an event handler to listen to the Disposed event on the component.
InfoMessage	Occurs when the provider or server returns a warning or informational message.
StateChange	Occurs when the state of the connection changes.

Examples

The following C# example shows how to use a constructor to set the connection string.

```
IfxConnection conn = new IfxConnection(
        "User Id=me;Password=myPassword;" +
        "Host=ajax;Server=myServer;" +
        "Service=9401;Database=superstores"
        );
```

IfxConnectionStringBuilder class

Provides the base class from which strongly typed connection string builders derive. This class extends from the DbConnectionStringBuilder class. You can use an instance of IfxConnectionString to construct the connection strings

Creating an IfxConnectionStringBuilder

To create an IfxConnectionStringBuilder use one of the constructors.

To develop provider independent code, you can use the CreateConnectionStringBuilder() method from a provider-specific instance of the DbProviderFactory class to create a provider-specific instance of the DbConnectionStringBuilder class.

IfxConnectionStringBuilder public properties

The following table shows the public properties of the IfxConnectionStringBuilder class.

Table 3-11. IfxConnectionStringBuilder Properties

Property	Description
Count	Returns the number of keys that are contained within the connection string that is maintained by the IfxConnectionStringBuilder instance.
ConnectionString	Gets or sets the connection string that is associated with the IfxConnectionStringBuilder. Returns a semicolon-delimited list of key-value pairs stored in the collection that is maintained by the IfxConnectionStringBuilder. Each pair contains the key and value, which are separated by an equal sign.
IsFixedSize	Indicates whether the IfxConnectionStringBuilder has a fixed size. A value of true indicates that the IfxConnectionStringBuilder has a fixed size.
IsReadOnly	Indicates whether the IfxConnectionStringBuilder is read-only. A value of true indicates that the IfxConnectionStringBuilder is read only. A read-only collection prohibits adding, removing, or modifying elements after the collection is created.
Keys	Returns an ICollection that contains the keys that are in the IfxConnectionStringBuilder. The ICollection contains an unspecified order of values, but it is the same order as the associated values in the ICollection returned by the Values property.
Values	Returns an ICollection that contains the values in the DbConnectionStringBuilder. The ICollection contains an unspecified order of values, but it is the same order as the associated values in the ICollection returned by the Keys property.

IfxConnectionStringBuilder public methods

IfxConnectionStringBuilder.Add

Adds an entry with the specified key and value into the IfxConnectionStringBuilder.

IfxConnectionStringBuilder.AppendKeyValuePair

Appends a key and value to an existing StringBuilder object.

IfxConnectionStringBuilder.Clear

Clears the contents from the IfxConnectionStringBuilder instance.

IfxConnectionStringBuilder.ContainsKey

Indicates whether this IfxConnectionStringBuilder object contains a specific key.

IfxConnectionStringBuilder.EquivalentTo

Compares the connection information in this IfxConnectionStringBuilder object with the connection information in another object.

IfxConnectionStringBuilder.Remove

Removes the specified key from the IfxConnectionStringBuilder instance.

IfxConnectionStringBuilder.ToString

Returns the connection string that is associated with this IfxConnectionStringBuilder object.

IfxConnectionStringBuilder.TryGetValue

Retrieves a value that corresponds to the supplied key from this IfxConnectionStringBuilder object.

IfxDataAdapter class

The IfxDataAdapter object uses IfxCommand objects to execute SQL commands against the database, fill data sets, and reconcile changes in the data set with the database.

Creating an IfxDataAdapter

To create an IfxDataAdapter use one of the constructors.

IfxDataAdapter constructors

- `IfxDataAdapter()`
- `IfxDataAdapter( IfxCommand selectCommand )`
- `IfxDataAdapter( System.String selectCommandText, IfxConnection selectConnection )`
- `IfxDataAdapter( System.String selectCommandText, System.String selectConnectionString )`

An SQL query that returns rows and connection to a database can be provided as either .NET types or strings. The value of *selectCommandText* is the query written in SQL. The value of *selectConnectionString* is a connection string as used by the constructors for the IfxConnection object.

IfxDataAdapter public properties

The following table shows the public properties of the IfxDataAdapter class.

Table 3-12. IfxDataAdapter Properties

Property	Description
AcceptChangedDuringFill	Gets or sets a value indicating if AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
AcceptChangesDuringUpdate	Gets or sets whether AcceptChanges is called during an Update.
DeleteCommand	Gets or sets an SQL statement for deleting records from the database.
FillLoadOption	Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader.
InsertCommand	Gets or sets an SQL statement used to insert new records into the database.
MissingMappingAction	The action to be taken when incoming data does not have matching table or column data sets. Indicates or specifies whether unmapped source tables or columns are passed with their source names so that they can be filtered or to raise an error. The MissingMappingAction property can have any of the values from the MissingMappingAction enumeration, described after the table.
MissingSchemaAction	Indicates or specifies whether missing source tables, columns, and their relationships are added to the data set schema, ignored, or cause an error to be returned. The MissingSchemaAction property can have any of the MissingSchemaAction enumeration values described after the table.
ReturnProviderSpecifictypes	Gets or sets whether the Fill method should return provider-specific values or common CLS-compliant values.
SelectCommand	Gets or sets an SQL statement used to select records in the database.
TableMappings	Indicates how a source table is mapped to a data set table. The default table name of a DataTable is Table. The default DataTableMapping that uses the default DataTable name is also Table.
UpdateBatchSize	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Gets or sets an SQL statement used to update records in the database.

The MissingMappingAction property can have the following values:

- Error—A SystemException is generated.
- Ignore—A column or table without a mapping is ignored.
- Passthrough—The source column and table are created if they do not already exist and they are added to the DataSet. This is the default value.

The MissingSchemaAction property can have the following values:

- Add—Adds any columns necessary to complete the schema.
- AddWithKey—Adds the necessary columns and primary key information to complete the schema. By default, primary keys are not created in the DataSet unless this property is specified. Setting this value ensures that incoming records that match existing records are updated instead of getting appended, which could potentially result in multiple copies of the same row.
- Error—A SystemException is generated.
- Ignore—Ignores the extra columns.

IfxDataAdapter public methods

IfxDataAdapter.Fill

Adds or refreshes rows in the DataSet to match those in the database using the DataSet name, and creates a DataTable named Table.

IfxDataAdapter.FillSchema

Adds a DataTable to the specified DataSet and configures the schema to match that in the database based on the specified SchemaType.

IfxDataAdapter.GetFillParameters

Gets the parameters set by the user when executing an SQL SELECT statement.

IfxDataAdapter.Update

Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet from a DataTable.

IfxDataAdapter examples

The first example demonstrates the use of the TableMappings and MissingMappingAction properties.

```
// IfxConnection -- con
DataSet ds = new DataSet();
string sql = "select fname from customer";
IfxDataAdapter da = new IfxDataAdapter(sql,con);
// Default -- MissingMappingAction set to Passthrough.
// Database Column name is used as Data Set Column Name. This
//is the default setting
//da.MissingMappingAction = MissingMappingAction.Passthrough;
// MissingMappingAction set to Ignore
// The column or table not having a mapping is ignored. Returns a
// null reference . Will return error while accessing DataRow
da.MissingMappingAction = MissingMappingAction.Ignore;
// MissingMappingAction set to Error
// DataColumnMapping & DataTableMapping is not done
// then DataAdapter.Fill  returns Exception
da.MissingMappingAction = MissingMappingAction.Error;
// If set to Error, DataColumnMapping and DataTableMapping has to
// be done
DataColumnMapping dcFnm = new DataColumnMapping("fname", "FirstName");
DataTableMapping dtCus = new DataTableMapping  ("customer","CustomerTable");
dtCus.ColumnMappings.Add(dcFnm);
// Activates the Mapping
```

```
da.TableMappings.Add(dtCus);
da.Fill(ds,"customer");
foreach(DataRow dr in ds.Tables["CustomerTable"].Rows)
{
      Console.WriteLine(dr["FirstName"]);
}
//Close Connection
```

The next example demonstrates how to use the FillSchema method in conjunction with the MissingSchemaAction property.

```
// IfxConnection -- con
DataSet ds = new DataSet();
string sql = "select fname from customer";
IfxDataAdapter da = new IfxDataAdapter(sql,con);
//MissSchemaAction is set to error so Fill will return
// exception if Data Set and Customer table schema
// do not match
da.MissingSchemaAction = MissingSchemaAction.Error;
// Fills Data Set Schema with the customer table schema
da.FillSchema(ds,SchemaType.Source,"customer");
da.Fill(ds,"customer");
foreach(DataRow dr in ds.Tables["customer"].Rows)
{
      Console.WriteLine(dr["fname"]);
}
//Close Connection
```

The following example illustrates the use of the SelectCommand and UpdateCommand properties.

```
IfxDataAdapter adpt = new IfxDataAdapter();
adpt.SelectCommand = new IfxCommand("SELECT CustomerID, Name FROM Customers
    WHERE Country = ? AND City = ?", conn);
IfxParameter ifxp1 = new IfxParameter("Country",DbType.String);
IfxParameter ifxp2 = new IfxParameter("City",DbType.String);
Adpt.SelectCommand.Parameters.Add(ifxp1);
Adpt.SelectCommand.Parameters.Add(ifxp2);
//similarly for an UpdateCommand
//adpt.UpdateCommand.Parameters.Add("CustomerName",DbType.String);
adpt.UpdateCommand.Parameters.["CustomerName"] = "xyz";
```

IfxDataReader class

The IfxDataReader object is a forward-only cursor that allows read-only access to the data it retrieves from the database. The data source connection must remain active while your application accesses the IfxDataReader object.

In general, performance is better when you use an IfxDataReader object than when you use an IfxDataAdapter object.

IfxDataReader public properties

The following table shows the public properties of the IfxDataReader class.

Table 3-13. IfxDataReader Properties

Property	Type	Description
Depth	System.Int32	Always returns 0.
FieldCount	System.Int32	Gets the number of columns in the current row.
IsClosed	System.Boolean	Gets a value indicating whether the IfxDataReader object is closed.
RecordsAffected	System.Int32	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
VisibleFieldCount	System.Int32	Gets the number of fields in the DbDataReader that are not hidden.

IfxDataReader public methods

IfxDataReader.Close

Closes the IfxDataReader object.

IfxDataReader.GetBoolean

Gets the value of the specified column as a Boolean.

IfxDataReader.GetByte

Throws a NotSupported exception.

IfxDataReader.GetBytes

Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.

IfxDataReader.GetChar

Gets the character value of the specified column.

IfxDataReader.GetChars

Reads a stream of characters from the specified column offset into the buffer as an array, starting at the given buffer offset.

IfxDataReader.GetData

Throws a NotSupported exception.

IfxDataReader.GetDateTime

Gets the date and time data value of the specified field.

IfxDataReader.GetDataTypeName

Gets the data type information for the specified field.

IfxDataReader.GetDecimal

Gets the fixed-point numeric value of the specified field.

IfxDataReader.GetDouble

Gets the double-precision floating point number of the specified field.

IfxDataReader.GetFieldType

Gets the Type information for the object returned by GetValue.

IfxDataReader.GetFloat
Gets the single-precision floating point number of the specified field.

IfxDataReader.GetGuid
Returns the GUID value of the specified field.

IfxDataReader.GetInt16
Gets the 16-bit signed integer value of the specified field.

IfxDataReader.GetInt32
Gets the 32-bit signed integer value of the specified field.

IfxDataReader.GetInt64
Gets the 64-bit signed integer value of the specified field.

IfxDataReader.GetName
Gets the name for the field to find.

IfxDataReader.GetOrdinal
Returns the index of the named field.

IfxDataReader.GetSchemaTable
Returns a DataTable object that describes the column metadata of the IfxDataReader object.

IfxDataReader.GetString
Gets the string value of the specified field.

IfxDataReader.GetTimeSpan
Gets the time span value of the specified field.

IfxDataReader.GetValue
Returns the value of the specified field.

IfxDataReader.GetValues
Gets all the attribute fields in the collection for the current record.

IfxDataReader.IsDBNull
Returns whether the specified field is set to null.

IfxDataReader.NextResult
When reading the results of batch SQL statements, advances the IfxDataReader object to the next result.

IfxDataReader.Read
Advances the IfxDataReader object to the next record.

IfxDataReader example
The following example demonstrates how to use the properties and methods of the IfxDataReader class.

```
// IfxConnection - con
string sql = "select stock_num,manu_code,description from stock";
con.Open();
IfxCommand  selectCommand = new IfxCommand(sql,con);
IfxDataReader reader = selectCommand.ExecuteReader(CommandBehavior.Default);
```

```
//schema for Dataset can be created by GetSchemaTable()
DataTable schema = reader.GetSchemaTable();
//read to use reader properties.
reader.Read();
Console.WriteLine("Depth is ");
Console.WriteLine(reader.Depth);
Console.WriteLine("Number of Columns are");
Console.WriteLine(reader.FieldCount);
Console.WriteLine("Number of Rows Changed");
Console.WriteLine(reader.RecordsAffected);
Console.WriteLine("Is Data Reader Closed ?");
Console.WriteLine(reader.IsClosed);
do
{
    while (reader.Read())
    {
        Int32 num = reader.GetInt32(0);
        Console.WriteLine(num );
        String string1 = reader.GetString(1);
        Console.WriteLine(string1);
        String string2 = reader.GetString(2);
        Console.WriteLine(string2);
    }
} while (reader.NextResult());
reader.Close();
//Close Connection
```

IfxDataSourceEnumerator class

IfxDataSourceEnumerator allows .NET applications to read Informix SQLHOST entries programatically. Setnet32 is utility that provides a GUI interface to SQLHOST entries.

Creating an IfxDataSourceEnumerator

You can create an IfxDataSourceEnumerator by using the following method:

- `DbDataSourceEnumerator enum = factory.CreateDataSourceEnumerator();`

Where `factory` is a provider-specific instance of DbProviderFactory.

IfxDataSourceEnumerator public properties

The following table shows the public properties of the IfxDataSourceEnumerator class.

Table 3-14. IfxDataSourceEnumerator Properties

Property	Description
Instance	Retrieves an enumerator.

IfxDataSourceEnumerator public methods

IfxDataSourceEnumerator.GetDataSources

Returns a DataTable. Each DataRecord in the DataTable represents a client server entry that is configured on the computer.

Table 3-15. Columns of System.Data.DataTable

Column name	Ordinal position	Description
IfxDatabaseServer	0	Name of the server instance. Multiple instances can exist on a single server.
HostComputer	1	Host entry in SQLHOSTS.
UserName	2	User name used to connect to the database.
PasswordOption	3	Password option as stored through SetNet32 for the host.
Password	4	Empty string.
Protocol	5	Protocol used for communication.
Service	6	Service name.
Option	7	Option field from SetNet32.

IfxDateTime structure

An IfxDateTime represents a single moment in the span of time from midnight on 1 January 0001 to 11:59:59.99999 pm. on 31 December 9999.

An IfxDateTime is treated as if it were made up of a separate value for each of these time units:

- Year
- Month
- Day
- Hour
- Minute
- Second
- Fractions of a second

You can create an IfxDateTime that uses only a subset of these time units. This is allowed in order to mimic the behavior of the database server's DATETIME data type. It does not save any space in memory when you use fewer time units in an IfxDateTime.

The largest time unit of an IfxDateTime is called the *start time unit*. The smallest time unit of an IfxDateTime is called the *end time unit*. The start time unit, the end time unit, and all time units in between are called the *range* of the IfxDateTime.

Example: If an IfxDateTime uses the year, month, and day portions then the start time unit is year, the end time unit is day, and the range is year to day.

Time units that are not included in the range of the IfxDateTime are assumed to have a default value as listed in this table.

Table 3-16. Default values for time units in IfxDateTime objects

Time unit	Default value
Year	1200
Month	1
Day	1
Hour	0
Minute	0
Second	0
Fraction	0

When creating an IfxDateTime you specify time units using the members of the IfxTimeUnit enumeration. For details about this enumeration see "IfxTimeUnit enumeration" on page 3-69.

Creating an IfxDateTime

All values for time units other than ticks are assumed to be numeric representations for the unit in an actual date.

Example: If you use a value of 13 for a month then you will get an error because there are only twelve months in a year. The 13 will not be converted to one year and one month.

IfxDateTime constructors

- `IfxDateTime( System.Int64 ticks )`

The new instance is set to a value equal to *ticks* ticks since midnight on 1 Jan 0001. There are 10 000 000 ticks in one second.

The range of the new instance is Year to Fraction5.

Note: Ticks are more precise than Fraction5. The extra precision is ignored by all methods and operators.

- `IfxDateTime( System.DateTime dt )`

The new instance is set to the same value as *dt*. The range of the new instance is Year to Fraction5.

- `IfxDateTime( System.Int32 numUnits, IfxTimeUnit unit )`

The instance has a range of *unit* to *unit*. The value is set to *numUnits* units past midnight on 1 Jan 0001.

- `IfxDateTime( System.Int32 numUnits1, System.Int32 numUnits2, IfxTimeUnit end )`
- `IfxDateTime( System.Int32 numUnits1, System.Int32 numUnits2, System.Int32 numUnits3, IfxTimeUnit end )`
- `IfxDateTime( System.Int32 numUnits1, System.Int32 numUnits2, System.Int32 numUnits3, System.Int32 numUnits4, IfxTimeUnit end )`

- `IfxDateTime( System.Int32 numUnits1, System.Int32 numUnits2, System.Int32 numUnits3, System.Int32 numUnits4, System.Int32 numUnits5, IfxTimeUnit end )`
- `IfxDateTime( System.Int32 numUnits1, System.Int32 numUnits2, System.Int32 numUnits3, System.Int32 numUnits4, System.Int32 numUnits5, System.Int32 numUnits6, IfxTimeUnit end )`
- `IfxDateTime( System.Int32 numUnits1, System.Int32 numUnits2, System.Int32 numUnits3, System.Int32 numUnits4, System.Int32 numUnits5, System.Int32 numUnits6, System.Int32 numUnits7, IfxTimeUnit end )`

If *numUnits1* through *numUnits7* are given then there is no *start* parameter because the start time unit is automatically assumed to be Year; otherwise the range of the new instance is *start* to *end*. The *end* time unit is always required because it determines the precision of the fractional portion.

Values must be provided for all units in the range. The *numUnits1* parameter is interpreted as the value for the start time unit. The rest of the values are interpreted as the values of the other time units in the range in order.

IfxDateTime public properties

These are the public properties of the IfxDateTime object.

Table 3-17. IfxDateTime public properties

Property	Type	Access Notes	Description
Date	IfxDateTime	read-only	An IfxDateTime that has the same value as the instance but has a range of Year to Day.
Day	System.Int32	read-only	The day portion of the value.
EndTimeUnit	IfxTimeUnit	read-only	The end time unit of the instance.
Hour	System.Int32	read-only	The hour portion of the value.
MaxValue	IfxDateTime	read-only static	An IfxDateTime that has the largest value possible in an IfxDateTime. The range of the IfxDateTime is Year to Fraction5.
Millisecond	System.Int32	read-only	The number of whole milliseconds in the fractional portion of the instance. There are 1000 milliseconds in one second.
MinValue	IfxDateTime	read-only static	An IfxDateTime that has the smallest value possible in an IfxDateTime. The range of the IfxDateTime is Year to Fraction5.
Minute	System.Int32	read-only	The minute portion of the value.
Month	System.Int32	read-only	The month portion of the value.
Now	IfxDateTime	read-only static	An IfxDateTime that is set to the current date and time and has a range of Year to Fraction5.
Null	IfxDateTime	read-only static	An IfxDateTime that is set to null.
Second	System.Int32	read-only	The seconds portion of the value.
StartTimeUnit	IfxTimeUnit	read-only	The start time unit of the instance.

Table 3-17. IfxDateTime public properties (continued)

Property	Type	Access Notes	Description
Ticks	System.Int64	read-only	The number of ticks from midnight on 1 Jan 0001 to the time in this instance. There are 10 000 000 ticks in one second.
Today	IfxDateTime	read-only static	An IfxDateTime set to the current time and having a range of Year to Day.
Year	System.Int32	read-only	The year portion of the value.

IfxDateTime public methods

IfxDateTime.Add

- `IfxDateTime IfxDateTime.Add( IfxTimeSpan` *`ifxTS`* `)`
- `IfxDateTime IfxDateTime.Add( IfxMonthSpan` *`ifxMS`* `)`

Returns a new IfxDateTime set to the value of the instance plus the amount of time represented by *ifxTS* or *ifxMS*. The new IfxDateTime has the same range as the instance. The instance itself is not changed.

Adding an IfxMonthSpan is not the same as adding a well defined span of time because there are a varying number of days in a month. When you add an IfxMonthSpan the addition is performed only on the year and month portions of the IfxDateTime. All other time units will be the same as they are in the instance.

IfxDateTime.AddDays

- `IfxDateTime IfxDateTime.AddDays( System.Double` *`days`* `)`

Returns a new IfxDateTime set to the same value as this instance plus the number of days in *days*. The value of *days* can be negative. Fractional values are permitted. The instance itself is not changed.

The new IfxDateTime has the same range as the instance. If the resulting date will not fit in that range an error is given.

IfxDateTime.AddMilliseconds

- `IfxDateTime IfxDateTime.AddMilliseconds( System.Double` *`milliseconds`* `)`

Returns a new IfxDateTime set to the same value as this instance plus the number of milliseconds in *milliseconds*. The value of *milliseconds* can be negative. It must be greater than -1 000 000 000 and less than 1 000 000 000. Fractional values are permitted. The instance itself is not changed.

The new IfxDateTime has the same range as the instance. If the resulting date will not fit in that range an error is given.

IfxDateTime.AddMinutes

- `IfxDateTime IfxDateTime.AddMinutes( System.Double` *`minutes`* `)`

Returns a new IfxDateTime set to the same value as this instance plus the number of minutes in *minutes*. The value of *minutes* can be negative. It must be greater than -1 000 000 000 and less than 1 000 000 000. Fractional values are permitted.

The new IfxDateTime has the same range as the instance. The instance itself is not changed. If the resulting date will not fit in that range an error is given.

IfxDateTime.AddMonths

- `IfxDateTime IfxDateTime.AddMonths( System.Double` *`months`* `)`

Returns a new IfxDateTime set to the same value as this instance except that the months portion has the value of *months* added to it. The value of *months* can be negative. The instance itself is not changed.

Note: This is not the same as adding a well defined span of time because the length of a month varies. Units smaller than month will never be affected by the addition.

The new IfxDateTime has the same range as the instance. If the resulting date will not fit in that range an error is given. You will also get an error if the resulting date is invalid, such as February 31.

IfxDateTime.AddSeconds

- `IfxDateTime IfxDateTime.AddSeconds( System.Double` *`seconds`* `)`

Returns a new IfxDateTime set to the same value as this instance plus the number of seconds in *seconds*. The value of *seconds* can be negative. It must be greater than -1 000 000 000 and less than 1 000 000 000. Fractional values are permitted. The instance itself is not changed.

The new IfxDateTime has the same range as the instance. If the resulting date will not fit in that range an error is given.

IfxDateTime.AddYears

- `IfxDateTime IfxDateTime.AddYears( System.Int32` *`years`* `)`

Returns a new IfxDateTime set to the same value as this instance except that the years portion has the value of *years* added to it. The value of *years* can be negative. The instance itself is not changed.

Note: This is not the same as adding a well defined span of time because the length of a year varies. Units smaller than year will never be affected by the addition.

The new IfxDateTime has the same range as the instance. If the resulting date will not fit in that range an error is given. You will also get an error if the resulting date is February 29 on a non-leap year.

IfxDateTime.Compare

- `static IfxDateTime IfxDateTime.Compare( IfxDateTime` *`ifxDT1`*`, IfxDateTime` *`ifxDT1`* `)`

Returns a value based on the relative values of *ifxDT1* and *ifxDT2*:

Value	Meaning
-1	*ifxDT1* is earlier than *ifxDT2*
0	*ifxDT1* and *ifxDT2* are the same time
1	*ifxDT1* is later than *ifxDT2*

Note: Objects in the IBM Informix .NET Provider consider two null values to be equal to each other. They also consider a null value to be less than any non-null value.

Any two IfxDateTime objects can be compared. Default values are used for any time units that are not in the range of the IfxDateTime. See Table 3-16 on page 3-38 for the default values that are used.

IfxDateTime.CompareTo

- `System.Int32 IfxDateTime.CompareTo( System.Object obj )`

The object *obj* must be an IfxDateTime.

This is equivalent to calling IfxDateTime.Compare with this instance as *ifxDT1* and *obj* as *ifxDT2*.

IfxDateTime.DaysInMonth

- `static System.Int32 IfxDateTime.DaysInMonth( System.Int32 year, System.Int32 month )`

Returns the number of days in the month *month* of the year *year*.

IfxDateTime.Equals

- `static System.Boolean IfxDateTime.Equals( IfxDateTime ifxDT1, IfxDateTime ifxDT2 )`

This method returns true if IfxDateTime.Compare(*ifxDT1*, *ifxDT2*) would return 0. If not it returns false.

IfxDateTime.GetHashCode

- `System.Int32 IfxDateTime.GetHashCode()`

Returns the hash code for this instance.

The hash code will be the same for any two IfxDateTime objects that have the same value but might also be the same for two IfxDateTime objects with different values.

See the description of the Object.GetHashCode method in the *.NET Framework Class Library* for details about hash codes.

IfxDateTime.GreaterThan

- `static System.Boolean IfxDateTime.GreaterThan( IfxDateTime ifxDT1, IfxDateTime ifxDT2 )`

This method returns true if IfxDateTime.Compare(*ifxDT1*, *ifxDT2*) would return 1. If not it returns false.

IfxDateTime.GreaterThanOrEqual

- `static System.Boolean IfxDateTime.GreaterThanOrEqual( IfxDateTime ifxDT1, IfxDateTime ifxDT2 )`

This method returns true if IfxDateTime.Compare(*ifxDT1*, *ifxDT2*) would return 0 or 1. If not, it returns false.

IfxDateTime.LessThan

- `static System.Boolean IfxDateTime.LessThan( IfxDateTime ifxDT1, IfxDateTime ifxDT2 )`

This method returns true if IfxDateTime.Compare(*ifxDT1*, *ifxDT2*) would return -1. If not it returns false.

IfxDateTime.LessThanOrEqual

- `static System.Boolean IfxDateTime.LessThanOrEqual( IfxDateTime ifxDT1, IfxDateTime ifxDT2 )`

This method returns true if IfxDateTime.Compare(*ifxDT1*, *ifxDT2*) would return 0 or -1. If not it returns false.

IfxDateTime.NotEquals

- `static System.Boolean IfxDateTime.NotEquals( IfxDateTime ifxDT1, IfxDateTime ifxDT2 )`

This method returns true if IfxDateTime.Compare(*ifxDT1*, *ifxDT2*) would return 1 or -1. If not it returns false.

IfxDateTime.Parse

- `static IfxDateTime IfxDateTime.Parse( System.String dateTimeString )`
- `static IfxDateTime IfxDateTime.Parse( System.String dateTimeString, IfxTimeUnit start, IfxTimeUnit end )`
- `static IfxDateTime IfxDateTime.Parse( System.String dateTimeString, System.String format, IfxTimeUnit start, IfxTimeUnit end )`

Returns a new IfxDateTime with a value based on *dateTimeString*. If *format* is not given, the *dateTimeString* string must be in this format:

```
Y-M-DD hh:mm:ss.f
```

Y An integer indicating the year.

M The number of the month in the range 1 to 12.

D The number of day in the month.

h The hour of the day in the range 0 to 23.

m The minute of the hour in the range 0 to 59.

s The second of the minute in the range 0 to 59.

f The fractional portion of the seconds. Precision beyond 5 decimal places is ignored.

The range of the new IfxDateTime is *start* to *end*. If *start* and *end* are not given the range is Day to Fraction5.

All time units in the range must be present in *dateTimeString*, even if they are zero. If *format* is provided then time units outside the range are optional. If they are present they are ignored. If *format* is not provided then time units outside the range are not allowed.

The *format* string uses the same syntax as the DBTIME environment variable. For the details of the syntax, refer to the description of the DBTIME environment variable in the *IBM Informix Guide to SQL: Reference*.

IfxDateTime.ToString

- `System.String IfxDateTime.ToString()`
- `System.String IfxDateTime.ToString ( System.String format )`

Returns the value of the instance as a string. If *format* is not present the format used is:

YYYY-MM-DD hh:mm:ss.f

YYYY
Four digit year

MM
Two digit month

DD
Two digit day

hh Two digit hour in range of 00 to 23

mm
Two digit minute

ss Two digit second

f The fractional portion of the seconds

Portions outside the range of the instance are not included in the string.

If *format* is provided the output is formatted in the way indicated in that string. The *format* string uses the same syntax as the DBTIME environment variable. For the details of the syntax, refer to the description of the DBTIME environment variable in the *IBM Informix Guide to SQL: Reference*.

IfxDecimal structure

An IfxDecimal object represents a decimal number with up to 32 significant digits. The range of valid values is from 10^{-129} to 10^{129}.

The DECIMAL data type in an Informix database can represent a larger range of values than will fit in any of the .NET Framework data types. It can also be a floating point number rather than a fixed point number. This is why you should use an IfxDecimal object to store values that are stored on the database as a DECIMAL or its equivalent.

You can use IfxDecimal to preserving numerical value, but not for preserving the scale. For example, a value 100 can be displayed as 100.0. Similarly, 100.00 can be displayed as 100.0. IfxDecimal always stores a floating point value. To preserve scale, use Decimal instead of IfxDecimal.

Creating an IfxDecimal

IfxDecimal objects are created automatically by some methods of other objects. You can create them explicitly by using one of the constructors.

IfxDecimal constructors

- `IfxDecimal( System.Double d )`
- `IfxDecimal( System.Decimal d )`
- `IfxDecimal( System.Int64 d )`

- `IfxDecimal( System.Int32 i32 )`

The new instance will have the value of the parameter.

IfxDecimal properties

Table 3-18. IfxDateTime public properties

Property	Type	Access Notes	Description
E	IfxDecimal	read-only static	The value of the irrational number e.
IsFloating	System.Boolean	read-only	This is true if the instance is a floating point number; otherwise it is false.
IsNull	System.Boolean	read-only	This is true if the instance is null otherwise it is false.
IsPositive	System.Boolean	read-only	This is true if the instance is a positive number; otherwise it is false.
MaxPrecision	System.Byte	read-only static	The highest precision (number of significant digits) supported by an IfxDecimal. This is currently 32.
MaxValue	IfxDecimal	read-only static	The largest value that can be held in an IfxDecimal.
MinusOne	IfxDecimal	read-only static	The value -1.
MinValue	IfxDecimal	read-only static	The smallest value that can be held in an IfxDecimal.
Null	IfxDecimal	read-only static	An IfxDecimal that is set to null.
One	IfxDecimal	read-only static	The value 1.
Pi	IfxDecimal	read-only static	The value of the irrational number pi.
Zero	IfxDecimal	read-only static	The value 0.

IfxDecimal methods

IfxDecimal.Abs

- `static IfxDecimal IfxDecimal.Abs( IfxDecimal IfxDec_ )`

Creates a new IfxDecimal that has a value equal to the absolute value of *IfxDec_*.

IfxDecimal.Add

- `static IfxDecimal IfxDecimal.Add( IfxDecimal IfxDec1, IfxDecimal IfxDec2 )`

Creates a new IfxDecimal that has a value equal to the sum of *IfxDec1* and *IfxDec2*.

IfxDecimal.Ceiling

- `static IfxDecimal IfxDecimal.Ceiling( IfxDecimal IfxDec )`

Creates a new IfxDecimal that is the smallest integer that is not less than *IfxDec*.

IfxDecimal.Clone

- IfxDecimal IfxDecimal.Clone()

Creates a new IfxDecimal that is a duplicate of this instance.

IfxDecimal.Compare

- `static System.Int32 IfxDecimal.Compare( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

The value returned is based on the relative values of *IfxDec1* and *IfxDec2*.

Value	Meaning
-1	The value of *IfxDec1* is less than the value of *IfxDec2*
0	The IfxDecimal objects have the same value
1	The value of *IfxDec1* is greater than the value of *IfxDec2*

Note: Objects in the IBM Informix .NET Provider consider two null values to be equal to each other. They also consider a null value to be less than any non-null value.

IfxDecimal.CompareTo

- `System.Int32 IfxDecimal.CompareTo( System.Object` *`obj`* `)`

This is the same as calling IfxDecimal.Compare with the instance as *IfxDec1* and *obj* as *IfxDec2*.

The object *obj* must be an IfxDecimal object. You will get an error if you call this method from an instance that is null.

IfxDecimal.Divide

- `static IfxDecimal IfxDecimal.Divide( IfxDecimal` *`Dividend`*`, IfxDecimal` *`Divisor`* `)`

Creates a new IfxDecimal that is the result of dividing *Dividend* by *Divisor*.

IfxDecimal.Equals

- `static System.Boolean IfxDecimal.Equals( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`
- `System.Boolean IfxDecimal.Equals( System.Object` *`obj`* `)`

Returns true if IfxDecimal.Compare(*IfxDec1, IfxDec2*) would return 0.

If just *obj* is given then the instance is used as *IfxDec1* and *obj* is used as *IfxDec2*. The *obj* object must be an IfxDecimal.

IfxDecimal.Floor

- `static IfxDecimal IfxDecimal.Floor( IfxDecimal` *`IfxDec`* `)`

Creates a new IfxDecimal whose value is the largest integer not larger than the value of *IfxDec*.

IfxDecimal.GetHashCode

- `System.Int32 IfxDecimal.GetHashCode()`

Returns the hash code for this instance.

The hash code will be the same for any two IfxDecimal objects that have the same value but might also be the same for two IfxDecimal objects with different values.

See the description of the Object.GetHashCode method in the *.NET Framework Class Library* for details about hash codes.

IfxDecimal.GreaterThan

- `static System.Boolean IfxDecimal.GreaterThan( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

Returns true if IfxDecimal.Compare(*IfxDec1*, *IfxDec2*) would return 1; otherwise returns false.

IfxDecimal.GreaterThanOrEqual

- `static System.Boolean IfxDecimal.GreaterThanOrEqual( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

Returns true if IfxDecimal.Compare(*IfxDec1*, *IfxDec2*) would return 0 or 1; otherwise returns false.

IfxDecimal.LessThan

- `static System.Boolean IfxDecimal.LessThan( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

Returns true if IfxDecimal.Compare(*IfxDec1*, *IfxDec2*) would return -1; otherwise returns false.

IfxDecimal.LessThanOrEqual

- `static System.Boolean IfxDecimal.LessThanOrEqual( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

Returns true if IfxDecimal.Compare(*IfxDec1*, *IfxDec2*) would return 0 or -1; otherwise returns false.

IfxDecimal.Max

- `static IfxDecimal IfxDecimal.Max( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

Creates a new IfxDecimal with a value equal to the value of *IfxDec1* or *IfxDec2*, whichever is larger.

Neither *IfxDec1* nor *IfxDec2* can be null.

IfxDecimal.Min

- `static IfxDecimal IfxDecimal.Min( IfxDecimal` *`IfxDec1`*`, IfxDecimal` *`IfxDec2`* `)`

Creates a new IfxDecimal with a value equal to the value of *IfxDec1* or *IfxDec2*, whichever is smaller.

Neither *IfxDec1* nor *IfxDec2* can be null.

IfxDecimal.Modulo

- `static IfxDecimal IfxDecimal.Modulo( IfxDecimal a, IfxDecimal b )`

Synonym for IfxDecimal.Remainder.

IfxDecimal.Multiply

- `static IfxDecimal IfxDecimal.Multiply( IfxDecimal IfxDec1, IfxDecimal IfxDec2 )`

Creates a new IfxDecimal that has a value equal to *IfxDec1* times *IfxDec2*.

IfxDecimal.Negate

- `static IfxDecimal IfxDecimal.Negate( IfxDecimal IfxDec )`

Creates a new IfxDecimal that is the result of reversing the sign (positive or negative) of this instance.

IfxDecimal.NotEquals

- `static System.Boolean IfxDecimal.NotEquals( IfxDecimal IfxDec1, IfxDecimal IfxDec2 )`

Returns true if IfxDecimal.Compare(*IfxDec1*, *IfxDec2*) would return 1 or -1.

IfxDecimal.Parse

- `static IfxDecimal IfxDecimal.Parse( System.String s )`

Creates a new IfxDecimal with a value equal to the decimal value represented in *s*.

Leading and trailing spaces in *s* are ignored. You can include an optional exponent by placing an `e` or `E` between the decimal value and the exponent. Exponents can be negative.

Example: This C# statement creates an IfxDecimal named d that has a value of -0.000032:

```
IfxDecimal d = IfxDecimal.Parse(" -3.2e-5  ");
```

IfxDecimal.Remainder

- `static IfxDecimal IfxDecimal.Remainder( IfxDecimal a, IfxDecimal b )`

Creates a new IfxDecimal the value of which is the remainder of the integer division of *a* by *b*. Integer division in this case means that *b* goes into *a* an integral (whole) number of times and what is left over is the remainder.

The sign (positive or negative) of the remainder will always match the sign of *a*.

IfxDecimal.Round

- `static IfxDecimal IfxDecimal.Round( IfxDecimal IfxDec1, System.Int32 FractionDigits )`

Returns a new IfxDecimal that has the value of *IfxDec1* rounded to *FractionDigits* digits to the right of the decimal point. If *FractionDigits* is 0 the value is rounded to the ones place. If *FractionDigits* is -1 the value is rounded to the tens place, and so on.

Example: If *IfxDec1* is an IfxDecimal with a value of 123.45 then this table gives the results of rounding to different positions.

Value of *FractionDigits*	**Result of IfxDecimal.Round(***IfxDec1*,*FractionDigits***)**
-2	100.0
-1	120.0
0	123.0
1	123.5
2	123.45

IfxDecimal.Subtract

- `static IfxDecimal IfxDecimal.Subtract( IfxDecimal IfxDec1, IfxDecimal IfxDec2 )`

Creates a new IfxDecimal that has a value of *IfxDec1* minus *IfxDec2*.

IfxDecimal.ToString

- `System.String IfxDecimal.ToString()`
- `System.String IfxDecimal.ToString ( System.String format )`

Returns the value of the instance as a string. If *format* is not present the format used is that of an ordinary decimal number, no exponent is used.

If *format* is provided, the output is formatted in the way indicated in that string. The syntax of *format* is as described in the section "Formatting Numeric Strings" in chapter 5 of *IBM Informix ESQL/C Programmer's Manual*.

IfxDecimal.Truncate

- `static IfxDecimal IfxDecimal.Truncate( IfxDecimal IfxDec1, System.Int32 FractionDigits )`

Like IfxDecimal.Round(*IfxDec1*, *FractionDigits*) except that all digits to the right of the indicated digit are set to zero rather than rounded.

Example: This table gives the results of truncating an IfxDecimal named *IfxDec1* that has a value of 999.99.

Value of *FractionDigits*	**Result of IfxDecimal.Truncate(***IfxDec1*,*FractionDigits***)**
-2	900.0
-1	990.0
0	999.0
1	999.9
2	999.99

IfxError class

The IfxError class represents an instance of a warning or an error generated by the IBM Informix database.

IfxError public properties

The following table shows the public properties of the IfxError class.

Table 3-19. IfxError Properties

Property	Description
Message	Gets the description of the error.
NativeError	Gets the error code returned from the IBM Informix database.
SQLState	Gets the five-character error code that follows the ANSI SQL standard for the database.

IfxErrorCollection class

The IfxErrorCollection class represents a collection of IfxError object occurrences in an IfxException object.

IfxErrorCollection public properties

The following table shows the public properties of the IfxErrorCollection class.

Table 3-20. IfxErrorCollection Properties

Property	Description
Count	Returns the number of IfxError occurrences in the collection.

IfxErrorCollection public methods

IfxErrorCollection.GetEnumerator

Returns an Enumerator (IEnumerator) to this collection.

IfxException class

The IfxException class represents an exception that is generated when a warning or error is returned by the IBM Informix database.

IfxException public properties

The following table shows the public properties of the IfxException class.

Table 3-21. IfxException Public Properties

Property	Description
Errors	Gets the collection of IfxError objects as an IfxErrorCollection object.
HelpLink	Gets the help link URL for the exception errors, if available.
InnerException	Gets the exception that caused the current exception.
Message	Gets the text describing the exception.
StackTrace	Gets a string representation of the frames on the call stack when the exception occurred.
TargetSite	Gets the method that returned the exception.

IfxMonthSpan structure

An IfxMonthSpan represents an offset of a particular number of months and years. A positive IfxMonthSpan represents an offset forward in time and a negative IfxMonthSpan represents an offset backward in time. An IfxMonthSpan can hold values from -11 999 999 999 months to 11 999 999 999 months (11 999 999 999 months = 999 999 999 years and 11 months).

An IfxMonthSpan is treated as if it were made up of a separate value for years and months.

You can create an IfxMonthSpan that uses only years or only months. This is allowed to mimic the behavior of the database server's INTERVAL data type. It does not save any space in memory to use only one time unit.

The largest time unit of an IfxMonthSpan is called the *start time unit*. The smallest time unit of an IfxMonthSpan is called the *end time unit*. The start time unit and the end time unit together are called the *range* of the IfxMonthSpan.

Example: If an IfxMonthSpan uses years and months, start time unit is year, the end time unit is month, and the range is year to month. If only months are used then the range is month to month and both the start time unit and end time unit are month.

When creating an IfxMonthSpan you specify time units using the members of the enumeration IfxTimeUnit enumeration. For details about this enumeration see "IfxTimeUnit enumeration" on page 3-69.

Creating an IfxMonthSpan

IfxMonthSpan constructors

- `IfxMonthSpan( System.Int32 val, IfxTimeUnit timeUnit )`

The new instance has only one time unit and it is set to the value *val*.

Note: The StartTimeUnit and EndTimeUnit are both set to *timeUnit*.

- `IfxMonthSpan( System.Int32 _years, System.Int32 _months )`

The new instance has a value equal to the sum of *_years* years and *_months* months. Negative values are allowed.

IfxMonthSpan public properties

Table 3-22. IfxMonthSpan properties

Property	Type	Access Notes	Description
EndTimeUnit	IfxTimeUnit	read-only	An IfxTimeUnit enumeration element indicating the end time unit of this instance.
IsNull	System.Boolean	read-only	Returns true if the instance is null; otherwise false.
MaxValue	IfxMonthSpan	read-only static	An IfxMonthSpan set to the largest value that it can hold.
MinValue	IfxMonthSpan	read-only static	An IfxMonthSpan set to the smallest value that it can hold.

Table 3-22. IfxMonthSpan properties (continued)

Property	Type	Access Notes	Description
Months	System.Int32	read-only	Returns the remainder of dividing the total number of months in the IfxTimeSpan by 12. If the IfxMonthSpan is negative then this value will be negative.
Null	IfxMonthSpan	read-only static	An IfxMonthSpan set to null.
StartTimeUnit	IfxTimeUnit	read-only	The largest unit included in the IfxMonthSpan.
TotalMonths	System.Int64	read-only	The total number of months in the IfxMonthSpan. If the IfxMonthSpan is negative then this value will be negative.
Years	System.Int32	read-only	The number of full years in the IfxMonthSpan. If the IfxMonthSpan is negative then this value will be negative.
Zero	IfxMonthSpan	read-only static	An IfxMonthSpan set to 0.

IfxMonthSpan public methods

IfxMonthSpan.Add

- `IfxMonthSpan IfxMonthSpan.Add( IfxMonthSpan` *`ms`* `)`

Returns a new IfxMonthSpan set to the value of the this instance plus the amount of time in *ms*.

The resulting IfxMonthSpan has the same range as this instance. This instance is not changed.

IfxMonthSpan.Compare

- `static System.Int32 IfxMonthSpan.Compare( IfxMonthSpan` *`ms1`*`, IfxMonthSpan` *`ms2`* `)`

This method does not compare the relative sizes of the spans, rather the IfxTimeSpan objects are compared as if they were both numbers. This means, for instance, that a span of -12 years is less than a span of 2 months.

Returns a value based on the relative values of *ms1* and *ms2*.

Value	Meaning
-1	*ms1* is less than *ms2*
0	*ms1* and *ms2* have the same value
1	*ms1* is greater than *ms2*

Note: Objects in the IBM Informix .NET Provider consider two null values to be equal to each other. They also consider a null value to be less than any non-null value.

IfxMonthSpan.CompareTo

- `System.Boolean IfxMonthSpan.CompareTo( System.Object obj )`

The object *obj* must be an IfxMonthSpan.

This is equivalent to calling IfxMonthSpan.Compare with this instance as *ms1* and *obj* as *ms2*.

IfxMonthSpan.Divide

- `IfxMonthSpan IfxMonthSpan.Divide( Decimal val )`

Returns a new IfxMonthSpan set to the value of this instance divided by *val*.

- `IfxMonthSpan IfxMonthSpan.Divide( IfxMonthSpan ms )`

Returns the number of spans of time that are the size of *ms* that will fit in the span of time represented by this instance. The result is negative if one of the IfxMonthSpan objects is negative and the other is not.

IfxMonthSpan.Duration

- `IfxMonthSpan IfxMonthSpan.Duration()`

Returns a new IfxMonthSpan with a value that is the absolute value of this instance.

IfxMonthSpan.Equals

- `static Boolean IfxMonthSpan.Equals( IfxMonthSpan ms1, IfxMonthSpan ms2 )`

Returns true if *ms1* and *ms2* have the same value; otherwise returns false.

- `Boolean IfxMonthSpan.Equals( System.Object obj )`

Returns true if *obj* is an IfxMonthSpan that represents the same time offset as this instance; otherwise it returns false.

IfxMonthSpan.GetHashCode

- `System.Int32 IfxMonthSpan.GetHashCode()`

Returns the hash code for this instance.

The hash code will be the same for any two IfxMonthSpan objects that have the same value but might also be the same for two IfxMonthSpan objects with different values.

See the description of the Object.GetHashCode method in the *.NET Framework Class Library* for details about hash codes.

IfxMonthSpan.GreaterThan

- `static System.Boolean IfxMonthSpan.GreaterThan( IfxMonthSpan ms1, IfxMonthSpan ms2 )`

Returns true if IfxMonthSpan.Compare(*ms1, ms2*) would return 1; otherwise, it returns false.

IfxMonthSpan.GreaterThanOrEqual

- `static System.Boolean IfxMonthSpan.GreaterThanOrEqual( IfxMonthSpan ms1, IfxMonthSpan ms2 )`

Returns true if IfxMonthSpan.Compare(*ms1*, *ms2*) would return either 1 or 0; otherwise, it is false.

IfxMonthSpan.LessThan

- `System.Boolean IfxMonthSpan.LessThan( IfxMonthSpan ms1, IfxMonthSpan ms2 )`

Returns true if IfxMonthSpan.Compare(*ms1*, *ms2*) would return -1; otherwise, it returns false.

IfxMonthSpan.LessThanOrEqual

- `System.Boolean IfxMonthSpan.LessThanOrEqual( IfxMonthSpan ms1, IfxMonthSpan ms2 )`

Returns true if IfxMonthSpan.Compare(*ms1*, *ms2*) would return -1 or 0; otherwise, it returns false.

IfxMonthSpan.Multiply

- `IfxMonthSpan IfxMonthSpan.Multiply( Decimal val )`

Returns a new IfxMonthSpan set to the value of this instance multiplied by *val*.

IfxMonthSpan.Negate

- `IfxMonthSpan IfxMonthSpan.Negate()`

Returns a new IfxMonthSpan with a value equal to this instance but with opposite sign (positive or negative).

IfxMonthSpan.NotEquals

- `static System.Boolean IfxMonthSpan.NotEquals( IfxMonthSpan ms1, IfxMonthSpan ms2 )`

Returns true if IfxMonthSpan.Compare(*ms1*, *ms2*) would return -1 or 1; otherwise, it returns false.

IfxMonthSpan.Parse

- `static IfxTimeSpan IfxMonthSpan.Parse( System.String val )`
- `static IfxTimeSpan IfxMonthSpan.Parse( System.String val, IfxTimeUnit start, IfxTimeUnit end )`
- `static IfxTimeSpan IfxMonthSpan.Parse( System.String val, System.String format, IfxTimeUnit start, IfxTimeUnit end )`

Returns a new IfxMonthSpan with a value based on *val*. If *format* is not given, the *val* string must be in this format:

`[s]y- m`

s Optional sign. If present this can be either + or -. The default is +. The brackets ([]) are not part of the time span. They indicate that the sign is optional.

y The number of whole years in the span. This must be an integer in the range 0 to 999 999 999.

m The number of months. This must be an integer in the range 0 to 11.

The range of the new IfxMonthSpan is *start* to *end*. Only Year and Month are allowed in an IfxMonthSpan. If *start* and *end* are not given, the range is Year to Month. Values each unit in the range must be present in *val*, even if one or both are zero. Values outside the range must not be present. If only one time unit is used then the - is not used.

The *format* string uses the same syntax as the DBTIME environment variable except that the only placeholders it can include are %Y and %m. The %Y placeholder in this context accepts the number of years in a range from 0 to 999 999 999. All units for which there are placeholders must be present . For the details of the syntax, refer to the description of the DBTIME environment variable in the *IBM Informix Guide to SQL: Reference*.

If both year and month are given in *val* and accepted in *format*, then they are both used even if the range is year to year or month to month. If a IfxMonthSpan has a range of year to year and its value includes a total number of months that is not evenly divisible by 12 the extra months are ignored.

Example: The string output by this command is 1.

```
IfxMonthSpan.Parse("1-11","%Y-%m",
System.Double.Year,System.Double.Year).ToString()
```

If the range of an IfxMonthSpan is month to month and both years and months are given in *val* and accepted by *format* then the years are converted to months.

Example: The string output by this command is 23.

```
IfxMonthSpan.Parse("1-11","%Y-%m",
System.Double.Month,System.Double.Month).ToString()
```

IfxMonthSpan.Subtract

- `IfxMonthSpan IfxMonthSpan.Subtract( IfxMonthSpan` *`ms`* `)`

Returns a new IfxMonthSpan set to the value of this instance minus the value of *ms*.

IfxMonthSpan.ToString

- `System.String IfxMonthSpan.ToString()`
- `System.String IfxMonthSpan.ToString ( System.String` *`format`* `)`

Returns the value of the instance as a string. If *format* is not present the format used is:

sy-m

s Optional sign. A minus sign is shown here if the IfxMonthSpan is negative. Nothing is shown for positive values.

y The number of whole years in the value

m The number of months left over after calculating *y*

If the IfxMonthSpan has only one time unit then only that time unit is output and the dash that goes between the year and month is omitted.

If *format* is provided, the output is formatted in the way indicated in that string. The *format* string uses the same syntax as the DBTIME environment variable except

that only the %m, %y and %Y placeholders are allowed. The %y and %Y placeholders work the same way in this string. For the details of the syntax, refer to the description of the DBTIME environment variable in the *IBM Informix Guide to SQL: Reference*.

IfxParameter class

The IfxParameter class represents a parameter to an IfxCommand object. It represents a single parameter stored in a collection that is represented by an IfxParameterCollection object.

Creating an IfxParameter class

IfxParameter constructors

- `IfxParameter()`
- `IfxParameter( System.String name, System.Object value )`
- `IfxParameter( System.String name, IfxType type )`
- `IfxParameter( System.String name, IfxType type, System.Int32 size )`
- `IfxParameter( System.String name, IfxType type, System.Int32 size, System.String sourceColumn )`
- `IfxParameter( System.String name, IfxType type, System.Int32 size, System.Data.ParameterDirection parameterDirection, System.Boolean isNullable, System.Byte precision, System.Byte scale, System.String sourceColumn, System.Data.DataRowVersion srcVersion, System.Object value )`

The parameters are as follows:

name
The name of the parameter.

value
The value that will be assigned to the parameter.

type
The informix type of the parameter. See "IfxType enumeration" on page 3-71 for details.

size
The size of the parameter.

parameterDirection
Whether this parameter is an input, output, or input/output parameter. Look up System.Data.ParameterDirection in the *.NET Framework Class Library* for details on the directions.

sourceColumn
The source column for the parameter.

isNullable
Set to true if the parameter can accept null values; otherwise false.

precision
The precision of the parameter.

scale
The scale of the parameter.

srcVersion
The source version of the parameter.

IfxParameter public properties

The following table shows the public properties of the IfxParameter class.

Table 3-23. IfxParameter Properties

Property	Description
DbType	Gets or sets the DbType of the parameter. The DbType property specifies the data type of the IfxParameter object.
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. If the direction is output, and execution of the associated IfxCommand does not return a value, the IfxParameter contains a null value. After the last row from the last result set is read, Output, InputOut, and ReturnValue parameters are updated. The possible values for the Direction property are shown after the table.
IfxType	Gets or sets the IfxType of the parameter. The IfxType property specifies the data type enumeration of the IBM Informix .NET Provider that maps to the Informix data type.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
ParameterName	Gets or sets the name of the parameter. The ParameterName is used to reference the parameter in the parameter collection.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the value. The SourceColumn property can be passed as an argument to the IfxParameter constructor, or set as a property of an existing IfxParameter object (See "IfxParameter examples").
SourceVersion	Gets or sets the DataRowVersion to use when loading value. The SourceVersion specifies which DataRow version the IfxDataAdapter object uses to retrieve the value. The SourceVersion property can be passed as an argument to the IfxParameter constructor, or set as a property of an existing IfxParameter (See "IfxParameter examples").
Value	Gets or sets the value of the parameter.

IfxParameter examples

The first example creates an IfxParameter object.

```
//illustrates example of creating and using IfxParameter
//assume we have obtained a connection
IfxDataAdapter adpt = new IfxDataAdapter();
adpt.SelectCommand = new IfxCommand("SELECT CustomerID, Name FROM Customers
    WHERE Country = ? AND City = ?", conn);
IfxParameter ifxp1 = new IfxParameter("Country",DbType.String);
IfxParameter ifxp2 = new IfxParameter("City",DbType.String);
//add parameter to the Parameter collection
//since our provider does not support named parameters, the order of parameters
//added to the collection is important.
Adpt.SelectCommand.Parameters.Add(ifxp1);
Adpt.SelectCommand.Parameters.Add(ifxp2);
```

```
//the above method of creating and adding a parameter can also be done in a
//single step as shown
//adpt.UpdateCommand.Parameters.Add("CustomerName",DbType.String);
//assign value to the parameter
adpt.UpdateCommand.Parameters.["CustomerName"] = "xyz";
```

The next example demonstrates the use of the SourceVersion and SourceColumn properties:

```
//The following assumptions have been made:
// 1.We have obtained a connection (conn) to our data source
//2. We have a filled a DataSet using a DataAdapter(custDA) that has the
//following SelectCommand:
// "SELECT CustomerID, CompanyName FROM Customers WHERE Country = ? AND City =
// ?";
// 3. following is the update statement for the UpdateCommand:
// string updateSQL = "UPDATE Customers SET CustomerID = ?, CompanyName = ? " +
//                 "WHERE CustomerID = ? ";
// 4.The CustomerID column in the DataRow being used has been modified with a
// new value.
custDA.UpdateCommand = new IfxCommand(updateSQL,conn);
//The customer id column is being used as a source for 2 parameters.
//(set CustomerID = ?, and //where CustomerID = ?)
//the last parameter to the Add command specifies the SourceColumn for the
//parameter
IfxParameter myParam1 = custDA.UpdateCommand.Parameters.Add(
    "CustomerID", IfxType.Char,5,"CustomerID");
//The following line of code is implied as default, but is provided for
//illustrative purposes
//We want to update CustomerID with the current value in the DataRow.
myParam1.SourceVersion = DataRowVersion.Current;
//Current is the default value
custDA.UpdateCommand.Parameters.Add("CompanyName", IfxType.VarChar);
//The last parameter to the Add command specifies the SourceColumn for the
//parameter
IfxParameter myParm2 = custDA.UpdateCommand.Parameters.Add(
    "OldCustomerID", IfxType.Char,5,"CustomerID");
//We want to use in our search filter, the original value of CustomerID in
//the DataRow
MyParm2.SourceVersion = DataRowVersion.Original;
CustDA.Update();
```

IfxParameterCollection class

The IfxParameterCollection class represents the parameters for an IfxCommand object.

Creating an IfxParameterCollection

You do not create an IfxParameterCollection directly. It is created automatically as part of an IfxCommand. To access it use the IfxCommand.Parameters property.

IfxParameterCollection public properties

The following table shows the public properties of the IfxParameterCollection class.

Table 3-24. IfxParameterCollection Properties

Property	Description
Count	Returns the number of parameters in the collection.
Item	Gets the parameter at the specified index.

IfxParameterCollection public methods

IfxParameterCollection.Add

- `IfxParameter Add( IfxParameter value )`

Adds the IfxParameter object *value* to the IfxParameterCollection.

- `IfxParameter Add( System.Object value )`
- `IfxParameter Add( System.String parameterName, System.Object value )`
- `IfxParameter Add( System.String parameterName, IfxType IfxType )`
- `IfxParameter Add( System.String parameterName, IfxType ifxType, System.Int32 size )`
- `IfxParameter Add( System.String parameterName, IfxType ifxType, System.Int32 size, System.String sourceColumn )`

Create an IfxParameter object using the parameters given, then add it to the IfxParameterCollection. See "Creating an IfxParameterCollection" on page 3-58 for information on what each parameter does.

This method returns the IfxParameter that was added.

IfxParameterCollection.Clear

- `void IfxParameterCollection.Clear()`

Removes all the elements in the IfxParameterCollection object.

IfxParameterCollection.Contains

- `System.Boolean IfxParameterCollection.Contains( System.Object value )`
- `System.Boolean IfxParameterCollection.Contains( System.String value )`

Gets a value indicating whether a parameter in the collection has the specified source table name.

IfxParameterCollection.CopyTo

- `void IfxParameterCollection.CopyTo( System.Array array, System.Int32 index )`

Copies the elements of a collection into an array at a specified index.

IfxParameterCollection.GetEnumerator

- `System.Collections.IEnumerator IfxParameterCollection.GetEnumerator()`

Returns an enumerator to the collection.

IfxParameterCollection.IndexOf

- `System.Int32 IfxParameterCollection.IndexOf( System.Object value )`
- `System.Int32 IfxParameterCollection.IndexOf( System.String value )`

Gets the location of the IfxParameter object within the collection.

IfxParameterCollection.Insert

- `void IfxParameterCollection.Insert( System.Int32 index, System.Object value)`

Inserts a parameter at a specified location.

IfxParameterCollection.Remove

- `void IfxParameterCollection.RemoveAt( System.Object value )`

Removes the IfxParameter object from the collection.

IfxParameterCollection.RemoveAt

- `void IfxParameterCollection.RemoveAt( System.String parameterName )`
- `void IfxParameterCollection.RemoveAt( System.Int32 index )`

Removes the IfxParameter object named *parameterName* or at location *index* from the collection.

IfxProviderFactory class

You can use the IfxProviderFactory class to write provider-independent data access code. After getting an instance of the required provider factory, you can use that provider factory to create instances of the provider-specific data access classes. IfxProviderFactory exposes a series of methods that return these class instances.

You can use the DbProviderFactory class to create a DbProvider instance specifically for the Dynamic Server invariant, IBM.Data.Informix, as shown in the following example:

```
DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.Informix");
```

IfxProviderFactory public methods

IfxProviderFactory.CreateConnectionStringBuilder

- `IfxProviderFactory CreateConnectionStringBuilder( IfxProviderFactory  )`

Returns an instance of a DbConnectionStringBuilder that the application developers can use to create connection strings dynamically

IfxProviderFactory.CreateConnection

- `IfxProviderFactory.CreateConnection( IfxParameter value )`

Returns an instance of a DbConnection that the application developers can use to connect to a data store. The DbConnection class exposes a method CreateCommand() that returns a new DbCommand instance. the developers can use this instead of the DbProviderFactory.CreateCommand() method to create a command for that connection

IfxProviderFactory.CreateCommand

- `IfxProviderFactory.CreateCommand()`

developers can use to execute SQL statements and stored procedures. The DbCommand class exposes a method CreateParameter() that returns a new DbParameter instance. The developers can use this instead of the DbProviderFactory.CreateParameter() method to create parameters for that command.

IfxProviderFactory.CreateParameter

- `IfxProviderFactory.CreateParameter ( )`

Returns an instance of a DbParameter that the application developers can use to pass values into and out of SQL statements and stored procedures

IfxProviderFactory.CreateCommandBuilder

- `IfxProviderFactory.CreateCommandBuilder()`

Returns an instance of a DbCommandBuilder that the application developers can use to create the UPDATE, INSERT and DELETE SQL statements for a DataAdapter automatically

IfxProviderFactory.CreateDataAdapter

- `IfxProviderFactory.CreateDataAdapter( )`

Returns an instance of a DbDataAdapter that the application developers can use to fill or update a DataSet or DataTable.

IfxProviderFactory.CreateDataSourceEnumerator

- `IfxProviderFactory.CreateDataSourceEnumerator( )`

Returns an instance of a DbDataSourceEnumerator that the application developers can use to examine the data sources available through this DbProviderFactory instance.

IfxProviderFactory.CreatePermission (PermissionState)

- `IfxProviderFactory.CreatePermission ( PermissionState)`

Takes a value from the PermissionState enumeration and returns an instance of a CodeAccessPermission that you can use to ensure that callers have been granted appropriate permission for all the objects to which they require access.

IfxSmartLOBCreateTimeFlags enumeration

The table below indicates the flags that can be set while creating a CLOB or BLOB object. The logical OR operation can be performed on one or more enumeration members listed in the table below and assigned to the IfxBlob.Flags or IfxClob.Flags property.

Member	Meaning
DontKeepAccessTime	If read this means that the smart large object does not keep track of the last time it was accessed. If written it tells the database server not to track the last access time for this smart large object. This flag overrides KeepAccessTime if both are given.

Member	Meaning
KeepAccessTime	If read this means that the smart large object keeps track of the last time it was accessed. If written it tells the database server to track the last access time for this smart large object. Use of the access time tracking feature causes significant extra work for the database server. Consider carefully before turning it on.
Log	If read this means that the database server logs changes to this smart large object in the system log file. If written it tells the database server to log changes to this smart large object in the system log file. Consider the extra overhead for the database server and the extra information that will be placed in the system log file before turning this feature on.
NoLog	If read this means that changes to this smart large object are not logged in the system log file. If written it tells the database server not to log changes to this smart large object in the system log file. This flag overrides Log if both are given.

IfxSmartLOBFileLocation enumeration

This enumeration is used to indicate which computer a particular file is on (or should be created on).

Member	Lock
Client	The file is on the computer that is running the client application.
Server	The file is on the computer that is running the database server.

IfxSmartLOBLocator class

This is a lower-level class that holds information about where a smart large object is stored. It encapsulates the locator structure of ESQL/C. You should never have to create or access an instance of this class explicitly.

IfxSmartLOBLockMode enumeration

This enumeration is used to indicate a particular type of lock.

Member	Lock
Exclusive	Open for writing only.
Shared	Open for reading and writing. The data is buffered locally and only written to the database server when the smart large object is closed.

IfxSmartLOBOpenMode enumeration

This enumeration is used to indicate what mode an IfxBlob or IfxClob object should be opened in. You OR the members of your choice together to specify how the smart large object will be accessed.

Member	Meaning
Append	If used by itself the smart large object is opened for reading only. If used with either ReadWrite or Write then the cursor is moved to the end of the smart large object before every write so that writes are always appended.
Buffer	If this is part of the access mode then reads and writes will use the standard database server buffer pool
DirtyRead	Open for reading only. You are allowed to read uncommited pages in the smart large object.
LockAll	If this is part of the access mode then any locks placed on the smart large object will lock the entire smart large object.
LockRange	If this is part of the access mode then you are allowed to lock a range in the smart large object without locking the entire smart large object.
Nobuffer	If this is part of the access mode the reads and writes will use private buffers from the session pool of the database server.
ReadOnly	Open for reading only.
ReadWrite	Open for reading and writing.
WriteOnly	Open for writing only.

IfxSmartLOBWhence enumeration

This enumeration is used to specify the meaning of an offset value. It is only used by methods of an IfxBlob or an IfxClob (collectively known as smart large objects).

Member	Lock
Begin	The offset is considered to be from the start of the smart large object. In this case the offset cannot be negative.
Current	The offset is considered to be from the current position of the smart large object's internal cursor.
End	The offset is considered to be from the current end of the smart large object.

IfxTimeSpan structure

An IfxTimeSpan represents an offset of a particular length either forward or backward in time. A positive IfxTimeSpan represents an offset forward in time and a negative IfxTimeSpan represents an offset backward in time.

An IfxTimeSpan is treated as if it is made up of a separate value for each of these time units:

- Day
- Hour
- Minute

- Second
- Fraction of a second

You can create an IfxTimeSpan that uses only a subset of these time units. This is allowed in order to mimic the behavior of the database server's INTERVAL data type. It does not save any space in memory when you use fewer time units in an IfxTimeSpan.

The largest time unit of an IfxTimeSpan is called the *start time unit*. The smallest time unit of an IfxTimeSpan is called the *end time unit*. The start time unit, the end time unit, and all units in between are called the *range* of the IfxTimeSpan.

Example: If an IfxTimeSpan uses hour, minute, and second units then the start time unit is hour, the end time unit is second, and the range is hour to second.

When creating an IfxTimeSpan you specify time units using the members of the IfxTimeUnit enumeration. For details about this enumeration see "IfxTimeUnit enumeration" on page 3-69.

Creating an IfxTimeSpan

In constructors that accept values for multiple time units, the values do not have to make sense with each other the way that they do in the constructors for an IfxDateTime. The values for one or more of the time units can be negative. The value of the created IfxTimeSpan is the sum of the time represented by each of the units.

Example: If you create an IfxTimeSpan using values of 50 days, 27 hours, and -5 minutes. The resulting IfxTimeSpan will be set to 51 days, 2 hours, and 55 minutes.

IfxTimeSpan constructors

- `IfxTimeSpan( System.Int64 _ticks )`
- `IfxTimeSpan( System.Decimal _ticks )`

The new instance has a range of Day to Fraction5 and is set to a value of *_ticks* ticks.

There are 10 000 000 ticks in one second.

Note: Ticks are more precise than Fraction5. The extra precision is ignored by all methods and operators.

- `IfxTimeSpan( System.TimeSpan ts )`

The new instance has the same value as *ts* and a range of Day to Fraction5.

- `IfxTimeSpan( System.Int32 val, IfxTimeUnit timeUnit )`

The new instance has only one time unit and it is set to the value *val*.

Note: The StartTimeUnit and EndTimeUnit are both set to *timeUnit*.

- `IfxTimeSpan( System.Int32 val1, System.Int32 val2, IfxTimeUnit start, IfxTimeUnit end)`
- `IfxTimeSpan( System.Int32 val1, System.Int32 val2, System.Int32 val3, IfxTimeUnit start, IfxTimeUnit end)`

- `IfxTimeSpan( System.Int32 val1, System.Int32 val2, System.Int32 val3, System.Int32 val4, IfxTimeUnit start, IfxTimeUnit end)`
- `IfxTimeSpan( System.Int32 val1, System.Int32 val2, System.Int32 val3, System.Int32 val4, System.Int32 val5, IfxTimeUnit end)`

If *val1* through *val5* are given then there is no *start* parameter because the start time unit is automatically assumed to be Day; otherwise the range of the new instance is *start* to *end*. The *end* time unit is always required because it determines the precision of the fractional portion.

Values must be provided for all units in the range. The *val1* parameter is interpreted as the value for the start time unit. The rest of the values are interpreted as the values of the other time units in the range in order.

IfxTimeSpan public properties

Table 3-25. Properties of IfxTimeSpan.

Property	Type	Access Notes	Description
Days	System.Int64	read-only	The number of full days in the IfxTimeSpan. If the IfxTimeSpan is negative then this value will be negative.
EndTimeUnit	IfxTimeUnit	read-only	The smallest unit included in the IfxTimeSpan. **Example:** If the IfxTimeSpan uses Day to Minute then Minute is the EndTimeUnit.
Hours	System.Int64	read-only	Returns the remainder of dividing the number of full hours in the IfxTimeSpan by 24.
IsNull	System.Boolean	read-only	True if the IfxTimeSpan is null, otherwise False.
MaxScale	System.Int32	read-only static	The largest number of digits allowed in the fraction of a second portion of the value. This currently has a value of 5.
MaxValue	IfxTimeSpan	read-only static	An IfxTimeSpan set to the largest value that it can hold.
Milliseconds	System.Int64	read-only	Returns the remainder of dividing the number of full milliseconds in the IfxTimeSpan by 1000. If the IfxTimeSpan is negative then this value will be negative.
Minutes	System.Int64	read-only	The component of TimeSpan that indicates the number of minutes. The value ranges from -59 to 59. If the IfxTimeSpan is negative then this value will be negative.
MinValue	IfxTimeSpan	read-only static	The smallest value that can be held in an IfxTimeSpan.
Null	IfxTimeSpan	read-only static	An IfxTimeSpan set to null.
Seconds	System.Int64	read-only	Returns the remainder of dividing the number of full minutes in the instance by 60. If the instance is negative then this value will be negative.
StartTimeUnit	IfxTimeUnit	read-only	The largest unit included in the IfxTimeSpan.

Table 3-25. Properties of IfxTimeSpan. (continued)

Property	Type	Access Notes	Description
Ticks	System.Decimal	read-only	The total number of ticks in the length of time represented by the IfxTimeSpan. There are 10 000 000 ticks in one second. If the IfxTimeSpan is negative then this value will be negative.
Zero	IfxTimeSpan	read-only static	The value 0.

IfxTimeSpan public methods

These are the methods of the IfxTimeSpan object.

IfxTimeSpan.Add

- `IfxTimeSpan IfxTimeSpan.Add( IfxTimeSpan ts )`

Returns a new IfxTimeSpan set to the value of the this instance plus the amount of time in *ts*.

The resulting IfxTimeSpan has the same range as this instance. This instance is not changed.

IfxTimeSpan.Compare

- `static System.Int32 IfxTimeSpan.Compare( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

This method does not compare the relative sizes of the spans, rather the IfxTimeSpan objects are compared as if they were both numbers. This means, for instance, that a span of -12 hours is less than a span of 2 hours.

Returns a value based on the relative values of *ts1* and *ts2*.

Value	Meaning
-1	*ts1* is less than *ts2*
0	*ts1* and *ts2* have the same value
1	*ts1* is greater than *ts2*

Note: Objects in the IBM Informix .NET Provider consider two null values to be equal to each other. They also consider a null value to be less than any non-null value.

IfxTimeSpan.CompareTo

- `System.Boolean IfxTimeSpan.CompareTo( System.Object obj )`

The object *obj* must be an IfxTimeSpan.

This is equivalent to calling IfxTimeSpan.Compare with the IfxTimeSpan as *ts1* and *obj* as *ts2*.

IfxTimeSpan.Divide

- `IfxTimeSpan IfxTimeSpan.Divide( Decimal val )`

Returns a new IfxTimeSpan set to the original IfxTimeSpan divided by *val*.

- `IfxTimeSpan IfxTimeSpan.Divide( IfxTimeSpan ts )`

Returns the number of spans of time that are the size of *ts* that will fit in the span of time represented by this instance of IfxTimeSpan. The result is negative if one of the IfxTimeSpan objects is negative and the other is not.

IfxTimeSpan.Duration

- `IfxTimeSpan IfxTimeSpan.Duration()`

Returns a new IfxTimeSpan with a value that is the absolute value of this instance.

IfxTimeSpan.Equals

- `static Boolean IfxTimeSpan.Equals( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

Returns true if *ts1* and *ts2* have the same value; otherwise returns false.

- `Boolean IfxTimeSpan.Equals( System.Object obj )`

Returns true if *obj* is an IfxTimeSpan that has the same value as this instance; otherwise it returns false.

IfxTimeSpan.GetHashCode

- `System.Int32 IfxTimeSpan.GetHashCode()`

Returns the hash code for this IfxTimeSpan.

The hash code will be the same for any two IfxTimeSpan objects that have the same value but might also be the same for two IfxTimeSpan objects with different values.

See the description of the Object.GetHashCode method in the *.NET Framework Class Library* for details about hash codes.

IfxTimeSpan.GreaterThan

- `static System.Boolean IfxTimeSpan.GreaterThan( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

Returns true if IfxTimeSpan.Compare(*ts1*, *ts2*) would return 1. Otherwise, it returns false.

IfxTimeSpan.GreaterThanOrEqual

- `static System.Boolean IfxTimeSpan.GreaterThanOrEqual( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

Returns true if IfxTimeSpan.Compare(*ts1*, *ts2*) would return either 1 or 0. Otherwise, it is false.

IfxTimeSpan.LessThan

- `System.Boolean IfxTimeSpan.LessThan( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

Returns true if IfxTimeSpan.Compare(*ts1*, *ts2*) would return -1. Otherwise, it returns false.

IfxTimeSpan.LessThanOrEqual

- `System.Boolean IfxTimeSpan.LessThanOrEqual( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

Returns true if IfxTimeSpan.Compare(*ts1*, *ts2*) would return -1 or 0. Otherwise, it returns false.

IfxTimeSpan.Negate

- `IfxTimeSpan IfxTimeSpan.Negate()`

Returns a new IfxTimeSpan with a value equal to this instance but with opposite sign (positive or negative).

IfxTimeSpan.NotEquals

- `static System.Boolean IfxTimeSpan.NotEquals( IfxTimeSpan ts1, IfxTimeSpan ts2 )`

Returns true if IfxTimeSpan.Compare(*ts1*, *ts2*) would return -1 or 1. Otherwise, it returns false.

IfxTimeSpan.Parse

- `static IfxTimeSpan IfxTimeSpan.Parse( System.String _szTime )`
- `static IfxTimeSpan IfxTimeSpan.Parse( System.String _szTime, IfxTimeUnit start, IfxTimeUnit end )`
- `static IfxTimeSpan IfxTimeSpan.Parse( System.String _szTime, System.String format, IfxTimeUnit start, IfxTimeUnit end )`

Returns a new IfxTimeSpan with a value based on *_szTime*. If *format* is not given, the *_szTime* string must be in this format:

`[-]d h:m:s.f`

- `-` Optional sign. If this is present the IfxTimeSpan will be negative. The brackets ([]) indicate that the sign is optional. They are not part of the format.
- *d* An integer indicating the number of days. This must be an integer in the range 0 to 999 999 999.
- *h* The number of hours. This must be an integer in the range 0 to 23.
- *m* The number of minutes. This must be an integer in the range 0 to 59.
- *s* The number of whole seconds. This must be an integer in the range 0 to 59.
- *f* The fractional portion of the seconds. Precision beyond 5 decimal places is ignored.

The range of the new IfxTimeSpan is *start* to *end*. If *start* and *end* are not given the range is Day to Fraction5.

All time units in the range must be present in *_szTime*, even if they are zero. If *format* is provided then time units outside the range are optional. If they are present they are ignored. If *format* is not provided then time units outside the range are not allowed.

The *format* string uses the same syntax as the DBTIME environment variable except that it cannot contain placeholders for month or year. For the details of the syntax, refer to the description of the DBTIME environment variable in the *IBM Informix ESQL/C Programmer's Manual*.

IfxTimeSpan.ToString

- `System.String IfxTimeSpan.ToString()`
- `System.String IfxTimeSpan.ToString ( System.String format )`

Returns the value of the instance as a string. If *format* is not present the format used is:

```
D hh:mm:ss.f
```

D The number of whole days in the value

hh Two digit hour in range of 00 to 23

mm
Two digit minute

ss Two digit second

f The fractional portion of the seconds

Portions outside the range of the instance are not included in the string.

If *format* is provided the output is formatted in the way indicated in that string. The *format* string uses the same syntax as the DBTIME environment variable. For the details of the syntax, refer to the description of the DBTIME environment variable in the *IBM Informix Guide to SQL: Reference*.

IfxTimeUnit enumeration

IfxTimeUnit is an enumeration that holds the valid time units used with IfxDateTime, IfxMonthSpan, and IfxTimeSpan. IfxTimeUnit has members for each of the major time units.

Table 3-26. The non-fraction members of IfxTimeUnit

IfxTimeUnit member	Unit
Year	Years
Month	Months
Day	Days
Hour	Hours
Minute	Minutes
Second	Full seconds

It also has several members that represent fractions of a second at several different precisions.

Table 3-27. Fractional members of IfxTimeUnit

IfxTimeUnit member	Precision	Example
Fraction1	Tenths of a second	1962-04-16 11:35:10.1

Table 3-27. Fractional members of IfxTimeUnit (continued)

IfxTimeUnit member	Precision	Example
Fraction2	Hundredths of a second	1962-04-16 11:35:10.12
Fraction or Fraction3	Thousandths of a second	1962-04-16 11:35:10.123
Fraction4	Ten thousandths of a second	1962-04-16 11:35:10.1234
Fraction5	Hundred thousandths of a second	1962-04-16 11:35:10.12345

Note: The Informix time data types include properties that return the fractions of a second portion as milliseconds (thousandths of a second). They do not, however, include properties that return the fractions of a second in any of the other precisions.

IfxTransaction class

The IfxTransaction class represents the transaction to be performed with the database.

IfxTransaction public properties

The following table shows the public properties of the IfxTransaction class.

Table 3-28. IfxTransaction Properties

Property	Description
Connection	The IfxTransaction object to associate with the transaction.
IsolationLevel	The isolation level for this transaction. **Note:** With IBM Informix database servers the serializable isolation level is identical to the repeatable-read isolation level. If you set the isolation level to repeatable-read in .NET it will actually be set to serializable in the database server.

IfxTransaction public methods

IfxTransaction.Commit

Commits the database transaction.

IfxTransaction.Rollback

Rolls back a transaction from a pending state.

Before your application executes a command for which you want to control the transaction, you must assign the active transaction used in a connection to the Transaction property of the IfxCommand object, as shown in the example, below. If you do not do this, an exception is returned.

IfxTransaction example

The following example shows how to perform an insert within a local transaction. The command `MyCommand.Transaction = myTrans;` assigns the active transaction to the Transaction property of the IfxCommand object.

```
IfxConnection myConn = new IfxConnection("Host=ajax;Server=myServer;
  Service=9401;database=dotnet;user id=xxx;password=xxx");
```

```
myConn.open();
IfxTransaction myTrans = myConn.BeginTransaction();
IfxCommand myCommand = new IfxCommand();
MyCommand.Transaction = myTrans;
MyCommand.CommandText = "INSERT INTO mytab(custid,custname)
    values(1005,\"Name\");"
MyCommand.ExecuteNonQuery();
MyTrans.Commit();
MyConn.Close();
```

IfxType enumeration

This enumerator is used with the IfxParameter object. Each member represents a data type that is supported by Informix database servers. Table 3-29 on page 3-71 shows all of the members and how each maps to .NET DbType types and to .NET Framework types. For detailed information about Informix types, see the *IBM Informix Guide to SQL: Reference*.

Table 3-29. IfxType Enumeration

Member	.NET DbType (best fit)	.NET Framework Type (best fit)
Bigint	Int64	Int64
BigSerial	Int64	Int64
Blob	Binary	Byte[]
Boolean	Boolean	Boolean
Byte	Binary	Byte[]
Char	StringFixedLength	String
Char1	StringFixedLength	Char
Clob	String	String
Collection	String	String
Date	Date	DateTime
DateTime	DateTime	DateTime
Decimal	Decimal	Decimal
Float	Double	Double
Int8	Int64	Int64
Integer	Int32	Int32
IntervalDayFraction	String	TimeSpan
IntervalYearMonth	String	String
List	String	String
LVarChar	String	String
Money	Currency	Decimal
MultiSet	String	String
NChar	StringFixedLength	String
NVarChar	String	String
Row	String	String
Serial	Int32	Int32

Table 3-29. IfxType Enumeration (continued)

Member	.NET DbType (best fit)	.NET Framework Type (best fit)
Serial8	Int64	Int64
Set	String	String
SmallFloat	Single	Single
SmallInt	Int16	Int16
SmartLOBLocator	Binary	Byte[]
Text	String	String
VarChar	String	String

Chapter 4. Sample programs

Demonstration programs 4-1
Examples 4-1
Retrieving a single value. 4-1
Retrieving multiple rows. 4-2
Executing SQL that does not return data and using a transaction 4-3
Retrieving data into a DataSet 4-3
Using an IfxCommandBuilder object to reconcile changes with the database 4-4
Calling a stored procedure 4-5
Using distributed transactions 4-6
Writing CLOBs to files 4-7

Demonstration programs

Demonstration programs for the IBM Informix .NET Provider are available in the %INFORMIXDIR%\demo\dotnetdemo folder.

They cover the following application development domains:

- C# Windows Forms
- VB.NET Windows Forms
- ASP.NET Web Forms

Examples

This section contains short examples that demonstrate the use of particular objects or show how to perform particular database tasks. The examples are short, in order to enhance clarity. Therefore, they do not represent real-world, full-size applications. All of these example are assumed to be in console applications written in the C# language. They all assume that you have already imported the IBM.Data.Informix namespace by including this directive in the program:

```
using IBM.Data.Informix;
```

Many of the examples use one of the sample databases that are included with IBM Informix database servers. The sample databases used are stores_demo and superstores_demo. Instructions on how to create these databases are in the *IBM Informix DB–Access User's Guide*.

Retrieving a single value

Use the IfxCommand.ExecuteScalar method when you know that the SQL you want to execute will return a single value. The IfxCommand.ExecuteScalar method returns a System.Object. You must cast this to the type of data that you expect to be returned. This example returns the output of `COUNT(*)` which is a decimal value so the System.Object is cast to type System.Decimal.

For more information about the IfxCommand class, see "IfxCommand class" on page 3-18.

```
try
{
    // Open a connection
    IfxConnection conn =
        new IfxConnection(
        "Host=myhost;Service=1541;Server=myifxserver;Database=stores_demo;"
```

© Copyright IBM Corp. 2003, 2008

```
        + "User ID=mylogin;password=mypassword"
        );
    conn.Open();

    // Create an SQL command
    IfxCommand cmd = new IfxCommand(
        "SELECT COUNT(*) FROM customer",
        conn
        );
    Decimal ccount = (Decimal)cmd.ExecuteScalar();
        Console.WriteLine("There are " + ccount + " customers");

    // Close the connection
    conn.Close();
    Console.ReadLine(); // Wait for a Return
}
catch(IfxException e)
{
    Console.WriteLine(e.ToString());
    Console.ReadLine(); //Wait for a Return
}
```

Retrieving multiple rows

This example connects to the stores_demo database and uses an IfxDataReader object to retrieve all of the first names from the customer table.

Use an IfxDataReader object for simple access to data when you do not have to write and do not have to move backward. For more information about the IfxDataReader class, see "IfxDataReader class" on page 3-33.

```
try
{
    // Open a connection
    IfxConnection conn =
        new IfxConnection(
        "Host=myhost;Service=1541;Server=myifxserver;Database=stores_demo;"
        + "User ID=mylogin;password=mypassword"
        );
    conn.Open();

// Create an SQL command
IfxCommand cmd = new IfxCommand(
              "SELECT fname FROM customer",
              conn );
IfxDataReader dr = cmd.ExecuteReader();
// Write the data to the console
while (dr.Read())
{
    Console.WriteLine(dr["fname"].ToString());
}
Console.ReadLine(); // Wait for a Return
dr.Close();

// Close the connection
conn.Close();
}
catch(IfxException e)
{
    Console.WriteLine(e.ToString());
    Console.ReadLine(); // Wait for a Return
}
```

Executing SQL that does not return data and using a transaction

To execute SQL statements that do not return any data use the IfxCommand.ExecuteNonQuery method. Types of SQL statements that do not return data include:

- Inserts
- Updates
- Deletes
- Creating or altering database objects

This example shows how to use IfxCommand.ExecuteNonQuery to perform an insert and also how to execute an IfxCommand inside a local transaction. For this example to work, the stores_demo database must have transaction logging. To create a stores_demo database that has transaction logging run the dbaccessdemo command with the `-log` option.

For the details of the IfxCommand class, see "IfxCommand class" on page 3-18. For the details of the IfxTransaction class see "IfxTransaction class" on page 3-70

```
try
{
// Open a connection
IfxConnection conn = new IfxConnection(
        "Host=myhost;Service=1541;"
        + "Server=myifxserver;Database=stores_demo;"
        + "User ID=mylogin;password=mypassword"
);
conn.Open();

//Begin the transaction
IfxTransaction tx = conn.BeginTransaction();

//Create an IfxCommand that uses the connection and transaction
IfxCommand cmd = new IfxCommand(
                "INSERT INTO state VALUES('XX','No State')",
                conn,
                tx );

//Execute the command
cmd.ExecuteNonQuery();

//Commit the transaction
tx.Commit();

// Close the connection
conn.Close();
}
catch(IfxException e)
{
Console.WriteLine(e.ToString());
Console.ReadLine(); //Wait for a Return
}
```

Retrieving data into a DataSet

To retrieve database information into a System.Data.DataSet object for further processing, use an IfxDataAdapter object. This example creates a System.Data.DataSet and populates it with the first and last names from the customer table. Then, to show that it is populated, it outputs the System.Data.DataSet to the console in the form of XML.

For the details of the IfxDataAdapter object, see "IfxDataAdapter class" on page 3-30. For more information about datasets see your .NET or ADO documentation.

```
try
{
    // Open a connection
    IfxConnection conn =
        new IfxConnection(
        "Host=myhost;Service=1541;Server=myifxserver;Database=stores_demo;"
        + "User ID=mylogin;password=mypassword"
        );
    conn.Open();

    IfxDataAdapter da = new IfxDataAdapter(
        "SELECT fname, lname FROM customer",
        conn );
    System.Data.DataSet ds = new System.Data.DataSet("Names");

    //Fill the DataSet
    da.Fill(ds);

    //The DataSet is ready to use.
    //This example outputs the DataSet to the Console as XML
    //just to show that it is populated.
    ds.WriteXml(Console.Out);
    Console.ReadLine(); //Wait for a Return

    // Close the connection
    conn.Close();
}
catch(IfxException e)
{
    Console.WriteLine(e.ToString());
    Console.ReadLine(); //Wait for a Return
}
```

Using an IfxCommandBuilder object to reconcile changes with the database

You can use the IfxCommandBuilder object to retrieve data with an SQL SELECT statement, make changes in the data set, and then reconcile those changes with the IBM Informix database, as shown in the following example. The IfxCommandBuilder object facilitates easy reconciliation of changes made in your dataset with the database.

For more information about the IfxCommandBuilder class, see "IfxCommandBuilder class" on page 3-21. For more information about reconciling changes in the database, see "Reconciling DataSet changes with the database" on page 1-6.

```
// Add the IBM Informix namespace

using System.Data;

using IBM.Data.Informix;

// Create a connection

IfxConnection conn=new IfxConnection("Host=berry; Service=3500;
  Server=testserver; User ID=informix; password=ifxtest;
  Database=testdb");

// Create a DataAdapter object

IfxDataAdapter allDataAdapter = new IfxDataAdapter();

IfxCommand selCmd = new IfxCommand("SELECT * FROM students", conn);

allDataAdapter.SelectCommand = selCmd;

//Set up the CommandBuilder object
```

```
IfxCommandBuilder cbuild = new IfxCommandBuilder(allDataAdapter);
DataSet allDataSet = new DataSet ();
try
{
    // Open the connection
    conn.Open();
    allDataAdapter.Fill(allDataSet);
    // Change the age of a student
    DataRow chRow;
    chRow = allDataSet.Tables["Table"].Rows[5];
    chRow["age"] = 24;
    // Use IfxDataAdapter.Update() to reconcile changes with the database
    allDataAdapter.Update(allDataSet);
}
catch (Exception ex)
{
    // Use a messagebox to show any errors
    MessageBox.Show (ex.Message);
}
// Close the connection
conn.Close();
```

Calling a stored procedure

You use an IfxCommand object to call a stored procedure. You must set the IfxCommand object CommandType property to StoredProcedure. The following example shows how to run a stored procedure and read any results returned by the stored procedure using an IfxDataReader object.

For more information about the IfxCommand class, see "IfxCommand class" on page 3-18. For more information about calling stored procedures, see "Calling stored procedures" on page 1-7.

```
// Add the IBM Informix namespace
using System.Data;
using IBM.Data.Informix;
// Create a connection
IfxConnection conn=new IfxConnection("Host=berry; Service=3500;
  Server=testserver; User ID=informix; password=ifxtest;
  Database=testdb");
conn.Open();
//Create a command object for the stored procedure
IfxCommand spCmd = new IfxCommand("testproc", conn);
// Set the CommandType property to Storedprocedure
spCmd.CommandType = CommandType.StoredProcedure
IfxDataReader testDataReader;
try
{
    testDataReader = spCmd.ExecuteReader();
    testDataReader.Close();
}
catch (Exception ex)
```

```
{
    // Use a messagebox to show any errors
    MessageBox.Show (ex.Message);
}
// Close the connection
conn.Close();
```

Using distributed transactions

The following example uses pseudo-code to demonstrate how to use distributed transactions.

```
...
using System.EnterpriseServices;
using IBM.Data.Informix;
...
[assembly: AssemblyKeyFile("test.snk")]
...
    public static void Main()
    {
        ...
            /* The 'using' construct below results in a call to Dispose on
            exiting the curly braces. It is important to dispose of COM+
            objects as soon as possible, so that COM+ services such as
            Object Pooling work properly */

        using (TwoPhaseTxn txn = new TwoPhaseTxn)
        {
            txn.TestAutoComplete_Exception();
        }

        using (TwoPhaseTxn txn = new TwoPhaseTxn)
        {
            txn.TestAutoComplete_TransactionVote();
        }
        ...
    }

//Transaction attributes specify the type of transaction requested

[Transaction(TransactionOption.RequiresNew)]
 public class TwoPhaseTxn : ServicedComponent
 {
     [AutoComplete]
     public void TestAutoComplete_Exception()
     {
          IfxConnection ifxConn1 = new IfxConnection("db=db1;server=srv1;
             enlist=true;");
          IfxConnection ifxConn2 = new IfxConnection("db=db2;server=srv2;
             enlist=true;");

          try
          {
               // db operation on ifxConn1
          }
          catch
          {
               // throw exception
          }

          try
          {
               // db operation on ifxConn2
          }
          catch
```

```
          {
               // throw exception
          }
     }
[AutoComplete]
     public void TestAutoComplete_TransactionVote()
     {
          IfxConnection ifxConn1 = new IfxConnection("db=db1;server=srv1;
             enlist=true;");
          IfxConnection ifxConn2 = new IfxConnection("db=db2;server=srv2;
             enlist=true;");

          try
          {
             // db operation on ifxConn1
          }
          catch
          {
             // In case of any failure, flag abort
             ContextUtil.MyTransactionVote = TransactionVote.Abort
          }

          try
          {
             // db operation on ifxConn2
          }
          catch
          {
             // In case of any failure, flag abort
             ContextUtil.MyTransactionVote = TransactionVote.Abort
          }
     }
```

Writing CLOBs to files

This example connects to the superstores_demo database and writes all of the CLOBs in the table catalog into files in the directory C:\tmp. The same technique is used to write BLOBs to files.

Note that the IfxClob instance must be opened before it is accessed.

For more information about the IfxClob class see "IfxClob class" on page 3-13.

```
try
{
    // Open a connection
    IfxConnection conn =
        new IfxConnection(
        "Host=myhost;" +
        "Service=1576;" +
        "Server=mydbserver;"+
        "Database=superstores_demo;" +
        "User ID=mylogin;password=mypassword"
        );
    conn.Open();

    // Create an SQL command
    IfxCommand cmd = new IfxCommand(
        "SELECT advert_descr, catalog_num FROM catalog",
        conn
        );
    IfxDataReader dr = cmd.ExecuteReader();

    // Write any CLOBs to files in C:\tmp
    while (dr.Read())
    {
```

```
        if(!dr.IsDBNull(0)){
            IfxClob c = dr.GetIfxClob(0);
            long num = dr.GetInt64(1);

            c.Open(IBM.Data.Informix.IfxSmartLOBOpenMode.ReadOnly);
            c.ToFile(
                "C:\\tmp\\" + num.ToString() + ".txt",
                System.IO.FileMode.Create,
                IfxSmartLOBFileLocation.Client
                );
        }
    }
    dr.Close();

    // Close the connection
    conn.Close();
}
catch(Exception e)
{
    //This is assumed to be a console application
    Console.WriteLine(e.ToString());
}
```

Appendix. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix Dynamic Server

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility Features

The following list includes the major accessibility features in IBM Informix Dynamic Server. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Tip: The IBM Informix Dynamic Server Information Center and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard Navigation

This product uses standard Microsoft Windows navigation keys.

Related Accessibility Information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software. The syntax diagrams in our manuals are available in dotted decimal format.

You can view the publications for IBM Informix Dynamic Server in Adobe Portable Document Format (PDF) using the Adobe Acrobat Reader.

IBM and Accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

© Copyright IBM Corp. 2003, 2008

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

© Copyright IBM Corp. 2003, 2008

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

> © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

? symbol, parameters 1-7
.NET base classes 3-6
.NET DbType types 2-2, 3-71
.NET Framework types 3-71
.NET interfaces 3-5

A

Abs method
  of IfxDecimal structure 3-45
AcceptChangesDuringFill property
  of IfxDataAdapter class 3-31
AcceptChangesDuringUpdate property
  of IfxDataAdapter class 3-31
Accessibility A-1
  keyboard A-1
  shortcut keys A-1
Add method
  of IfxConnectionStringBuilder class 3-30
  of IfxDateTime structure 3-40
  of IfxDecimal structure 3-45
  of IfxMonthSpan structure 3-52
  of IfxParameterCollection class 3-59
  of IfxTimeSpan structure 3-66
Add value 3-32
AddDays method
  of IfxDateTime structure 3-40
AddMilliseconds method
  of IfxDateTime structure 3-40
AddMinutes method
  of IfxDateTime structure 3-40
AddMonths method
  of IfxDateTime structure 3-41
AddSeconds method
  of IfxDateTime structure 3-41
AddWithKey value 3-32
AddYears method
  of IfxDateTime structure 3-41
ADO.NET 1-3
AnsiString type 2-2
Append member
  of IfxSmartLOBOpenMode enumeration 3-63
AppendKeyValuePair method
  of IfxConnectionStringBuilder class 3-30
ArgumentException exception 3-26
Arrays 1-7
ASP.NET 1-1, 4-1
Assembly, strong-named 1-2
Attributes, for connecting 1-5
Automatic INSERT, DELETE and UPDATE 1-6, 3-21

B

Begin member
  of IfxSmartLOBWhence enumeration 3-63
BeginTransaction method
  of IfxConnection class 3-27
Bigint member
  of IfxType enumeration 3-71
BIGINT type 2-1
BigSerial member
  of IfxType enumeration 3-71
BIGSERIAL type 2-1
Blob member
  of IfxType enumeration 3-71
BLOB type 2-1
Boolean member
  of IfxType enumeration 3-71
BOOLEAN type 2-1
Both value 3-19
Buffer member
  of IfxSmartLOBOpenMode enumeration 3-63
Byte member
  of IfxType enumeration 3-71
BYTE type 2-1

C

Calling a stored procedure
  example of 4-5
Cancel method
  of IfxCommand class 3-19
casting, example of 4-1
cdotnet.sql script 1-3
Ceiling method
  of IfxDecimal structure 3-45
ChangeDatabase method
  of IfxConnection class 3-28
Char member
  of IfxType enumeration 3-71
CHAR type 2-1
CHAR(1) type 2-1
Char1 member
  of IfxType enumeration 3-71
Classes,unsupported 3-6
Clear method
  of IfxConnectionStringBuilder class 3-30
  of IfxParameterCollection class 3-59
Client Locale connection string attribute 3-25
Client member
  of IfxSmartLOBFileLocation enumeration 3-62
Client SDK 1-2
ClientLocale property
  of IfxConnection class 3-24
Clob member
  of IfxType enumeration 3-71
CLOB type 2-1
Clone method
  of IfxDecimal structure 3-46
Close method
  of IfxBlob class 3-10
  of IfxClob class 3-15
  of IfxConnection class 3-28
  of IfxDataReader class 3-34
Close property
  of IfxConnection
    use of 1-5
CloseConnection value 3-20
Closing connections 1-5
Codes, errors 3-50

© Copyright IBM Corp. 2003, 2008

Collection member
of IfxType enumeration 3-71
COLLECTION type 2-2
CommandText property 1-7
of IfxCommand 3-18
CommandTimeout property 1-6
of IfxCommand 3-18
CommandType property 1-7
of IfxCommand 3-18
of IfxCommand class
example of use 4-5
Commit method
of IfxTransaction class 3-70
Compare method
of IfxDateTime structure 3-41
of IfxDecimal structure 3-46
of IfxMonthSpan structure 3-52
of IfxTimeSpan structure 3-66
CompareTo method
of IfxDateTime structure 3-42
of IfxDecimal structure 3-46
of IfxMonthSpan structure 3-53
of IfxTimeSpan structure 3-66
ConflictOption property
of IfxCommandBuilder class 3-22
Connecting to databases 1-5
Connection Lifetime connection string attribute 3-25
Connection pool 1-7, 3-25
Connection property 1-6
of IfxCommand 3-18
of IfxTransaction class 3-70
Connection string 1-5
attributes of 3-24
defining visually 1-5
example of 3-28
ConnectionString property
attributes of 3-24
of IfxConnection
use of 1-5
of IfxConnection class 3-24
of IfxConnectionStringBuilder class 3-29
ConnectionString property of IfxConnection
defining visually 1-5
ConnectionTimeout property
of IfxConnection class 3-24
Constructors
of IfxBlob class 3-8
of IfxClob class 3-13
of IfxCommand class 3-18
of IfxCommandBuilder class 3-22
of IfxConnection class 3-24
of IfxDataAdapter class 3-30
of IfxDateTime structure 3-38
of IfxDecimal structure 3-44
of IfxMonthSpan structures 3-51
of IfxParameter class 3-56
of IfxTimeSpan structure 3-64
Contains method
of IfxParameterCollection class 3-59
ContainsKey method
of IfxConnectionStringBuilder class 3-30
CopyTo method
of IfxParameterCollection class 3-59
Count property
of IfxConnectionStringBuilder class 3-29
of IfxErrorCollection class 3-50
of IfxParameterCollection class 3-59
CREATE DATABASE... statements 3-25
CreateCommand method
of IfxConnection class 3-28
of IfxProviderFactory class 3-60
CreateCommandBuilder method
of IfxProviderFactory class 3-61
CreateConnection method
of IfxProviderFactory class 3-60
CreateConnectionStringBuilder method
of IfxProviderFactory class 3-60
CreateDataAdapter method
of IfxProviderFactory class 3-61
CreateDataSourceEnumerator method
of IfxProviderFactory class 3-61
CreateParameter method
of IfxCommand class 3-19
of IfxProviderFactory class 3-61
CreatePermission method
of IfxProviderFactory class 3-61
Creating
IfxCommand class 3-18
IfxCommandBuilder class 3-22
IfxConnection class 3-23
IfxConnectionStringBuilder class 3-29
IfxDataAdapter class 3-30
IfxDataSourceEnumerator class 3-36
IfxDateTime structure 3-38
IfxDecimal structure 3-44
IfxMonthSpan structure 3-51
IfxParameter class 3-56
IfxParameterCollection class 3-58
IfxTimeSpan structure 3-64
Current member
of IfxSmartLOBWhence enumeration 3-63

D

Data source 1-3
DataAdapter property
of IfxCommandBuilder class 3-22
Database connection string attribute 3-25
Database Locale connection string attribute 3-25
Database property
of IfxConnection class 3-24
DATABASE... statements 3-25
DatabaseLocale property
of IfxConnection class 3-24
Databases
connecting to 1-5
DataColumn interface 1-4
DataReader object 1-4
DataRelation interface 1-4
DataSet object 1-4, 2-1
example of use 4-3
DataTable interface 1-4
Date member
of IfxType enumeration 3-71
Date property
of IfxDateTime structure 3-39
DATE type 2-1
DATETIME data type 3-37
DateTime member
of IfxType enumeration 3-71
DATETIME type 2-1
Day member
of IfxTimeUnit enumeration 3-69

Day property
of IfxDateTime structure 3-39
Days property
of IfxTimeSpan structure 3-65
DaysInMonth method
of IfxDateTime structure 3-42
DB connection string attribute 3-25
DB_LOCALE connection string attribute 3-25
DbProviderFactory class 3-60
DbType property
of IfxParameter class 3-57
Decimal member
of IfxType enumeration 3-71
DECIMAL type 2-1
Default value 3-20
DELETE statements 3-21
DELETE, automatic 1-6
DeleteCommand property 1-6
of IfxDataAdapter class 3-31
DELIMIDENT connections string attribute 3-25
DELIMIDENT environment variable 1-5
default setting of 1-5
Delimiters 1-7, 2-2, 3-22
Demonstration programs 4-1
Depth property
of IfxDataReader class 3-34
DeriveParameters method 1-8
of IfxCommandBuilder class 3-22
Direction property
of IfxParameter class 3-57
DirtyRead member
of IfxSmartLOBOpenMode enumeration 3-63
Disability A-1
Dispose method 1-6
Disposed event
of IfxConnection class 3-28
Distributed transactions 3-25, 4-6
example of use 4-6
Divide method
of IfxDecimal structure 3-46
of IfxMonthSpan structure 3-53
of IfxTimeSpan structure 3-67
DontKeepAccessTime member
of IfxSmartLOBCreateTimeFlags enumeration 3-61
dotnetdemo folder 4-1
DOUBLE type 2-1
Double-quote characters 2-2
Duration method
of IfxMonthSpan structure 3-53
of IfxTimeSpan structure 3-67
Dynamic SQL 1-7

E

E property
of IfxDecimal structure 3-45
Efficient SQL statements 1-6
End member
of IfxSmartLOBWhence enumeration 3-63
end time unit 3-37, 3-51
EndTimeUnit property
of IfxDateTime structure 3-39
of IfxMonthSpan structure 3-51
of IfxTimeSpan structure 3-65
Enlist attribute 1-8
Enlist connection string attribute 3-25
Equals method
of IfxDateTime structure 3-42
of IfxDecimal structure 3-46
of IfxMonthSpan structure 3-53
of IfxTimeSpan structure 3-67
EquivalentTo method
of IfxConnectionStringBuilder class 3-30
Error messages 1-9, 3-49
Error value 3-31, 3-32
Errors property
of IfxException class 3-50
EstimatedSize property
of IfxBlob 3-9
of IfxClob 3-14
Events
of IfxConnection class 3-28
Example code
IfxCommand class 3-20
Examples 4-1
calling a stored procedure 4-5
casting a data type to a new type 4-1
Inserting rows 4-3
of a connection string 3-28
Reconciling changes in a DataSet with the database 4-4
retrieving a single value 4-1
Retrieving data into a DataSet 4-3
retrieving multiple rows 4-2
use of ExecuteNonQuery 4-3
use of ExecuteScalar method 4-1
use of IfxCommand.CommandType property 4-5
use of IfxCommandBuilder 4-4
use of IfxDataAdapter 4-3
use of IfxDataReader class 4-2
use of IfxParameter class 3-57
use of IfxTransaction class 3-70
use of local transaction 4-3
use of System.DataSet 4-3
using distributed transactions 4-6
Exceptions 1-9
Exclusive connection string attribute 3-25
Exclusive member
of IfxSmartLOBLockMode enumeration 3-62
ExecuteNonQuery method
example of use 4-3
of IfxCommand class 3-19
ExecuteNonQuery() method 4-3
ExecuteReader method
of IfxCommand class 3-20
ExecuteScalar method
of IfxCommand class 3-20
ExecuteScalar method, example of use 4-1
ExtentSize property
of IfxBlob 3-9
of IfxClob 3-14

F

FBS connection string attribute 3-25
Fetch Buffer Size connection string attribute 3-25
FetchBufferSize property
of IfxConnection class 3-24
FieldCount property
of IfxDataReader class 3-34
Fill method
of IfxDataAdapter class 3-32
FilLoadOption property
of IfxDataAdapter class 3-31

FillSchema method
of IfxDataAdapter class 3-32
FirstReturnedRecord value 3-19
Flags property
of IfxBlob 3-9
of IfxClob 3-14
Float member
of IfxType enumeration 3-71
FLOAT type 2-1
Floor method
of IfxDecimal structure 3-46
Foreign keys 1-7
Forward-only cursor 1-4, 3-25, 3-33, 4-2
Fraction member
of IfxTimeUnit enumeration 3-70
Fraction1 member
of IfxTimeUnit enumeration 3-69
Fraction2 member
of IfxTimeUnit enumeration 3-70
Fraction3 member
of IfxTimeUnit enumeration 3-70
Fraction4 member
of IfxTimeUnit enumeration 3-70
Fraction5 member
of IfxTimeUnit enumeration 3-70
FromFile method
of IfxBlob class 3-10
of IfxClob class 3-15
FullTrust permission 1-7

G

GAC.
See Global assembly cache.
GetActiveConnectionsCount property
of IfxConnection class 3-24
GetBoolean method
of IfxDataReader class 3-34
GetByte method
of IfxDataReader class 3-34
GetBytes method
of IfxDataReader class 3-34
GetChar method
of IfxDataReader class 3-34
GetChars method
of IfxDataReader class 3-34
GetData method
of IfxDataReader class 3-34
GetDataSources method
of IfxDataSourceEnumerator class 3-37
GetDataTypeName method
of IfxDataReader class 3-34
GetDateTime method
of IfxDataReader class 3-34
GetDecimal method
of IfxDataReader class 3-34
GetDeleteCommand method 1-6
of IfxCommandBuilder class 3-22
GetDouble method
of IfxDataReader class 3-34
GetEnumerator method
of IfxErrorCollection class 3-50
of IfxParameterCollection class 3-59
GetFieldType method
of IfxDataReader class 3-34
GetFillParameters method
of IfxDataAdapter class 3-32
GetFloat method
of IfxDataReader class 3-35
GetGuid method
of IfxDataReader class 3-35
GetHashCode method
of IfxDateTime structure 3-42
of IfxDecimal structure 3-47
of IfxMonthSpan structure 3-53
of IfxTimeSpan structure 3-67
GetIdleConnectionsCount property
of IfxConnection class 3-24
GetIfxBlob method
of IfxConnection class 3-28
GetIfxClob method
of IfxConnection class 3-28
GetInsertCommand method 1-6
of IfxCommandBuilder class 3-22
GetInt16 method
of IfxDataReader class 3-35
GetInt32 method
of IfxDataReader class 3-35
GetInt64 method
of IfxDataReader class 3-35
GetLocator method
of IfxBlob class 3-11
of IfxClob class 3-16
GetName method
of IfxDataReader class 3-35
GetOrdinal method
of IfxDataReader class 3-35
GetSchemaTable method
of IfxDataReader class 3-35
GetString method
of IfxDataReader class 3-35
GetString() method 2-2
GetTimeSpan method
of IfxDataReader class 3-35
GetUpdateCommand method 1-6
of IfxCommandBuilder class 3-22
GetValue method
of IfxDataReader class 3-35
GetValues method
of IfxDataReader class 3-35
Global assembly cache (GAC) 1-2
GreaterThan method
of IfxDateTime structure 3-42
of IfxDecimal structure 3-47
of IfxMonthSpan structure 3-53
of IfxTimeSpan structure 3-67
GreaterThanOrEqual method
of IfxDateTime structure 3-42
of IfxDecimal structure 3-47
of IfxMonthSpan structure 3-54
of IfxTimeSpan structure 3-67
GUID type 2-2

H

HelpLink property
of IfxException class 3-50
Host connection string attribute 3-25
Hostnames
using in sqlhosts 1-2
Hour member
of IfxTimeUnit enumeration 3-69
Hour property
of IfxDateTime structure 3-39

Hours property
of IfxTimeSpan structure 3-65

I

IBM Informix .NET Provider
installing 1-2
namespace of 1-4
overview 1-3
IBM Informix Client SDK 1-2
IBM Informix ODBC Driver 1-1
IBM Informix OLE DB Provider 1-1
IBM.Data.Informix 3-60
IBM.Data.Informix namespace 1-4
IDSSECURITYLABEL type 2-1
IfxBlob class 3-8, 3-13
Close method 3-10
constructors 3-8
creating 3-8
EstimatedSize property 3-9
ExtentSize property 3-9
Flags property 3-9
FromFile method 3-10
GetLocator method 3-11
internal cursor 3-8
IsNull property 3-9
IsOpen property 3-9
LastAccessTime property 3-9
LastChangeTime property 3-10
LastModificationTime property 3-10
Lock method 3-11
MaxBytes property 3-10
Null property 3-10
Open method 3-11
Position property 3-10
properties 3-9
Read method 3-11
ReferenceCount property 3-10
Release method 3-11
SBSpace property 3-10
Seek method 3-12
Size property 3-10
ToFile method 3-12
Truncate method 3-12
Unlock method 3-12
Write method 3-12
IfxBlob class methods
Methods
of IfxBlob class 3-10
IfxClob 3-13, 3-18
constructors 3-13
IfxClob class
Close method 3-15
creating 3-13
EstimatedSize property 3-14
ExtentSize property 3-14
Flags property 3-14
FromFile method 3-15
GetLocator method 3-16
internal cursor of 3-13
IsNull property 3-14
IsOpen property 3-14
LastAccessTime property 3-14
LastChangeTime property 3-15
LastModification property 3-15
Lock method 3-16
MaxBytes property 3-15
IfxClob class *(continued)*
methods of 3-15
Null property 3-15
Open method 3-16
Position property 3-15
Read method 3-16
ReferenceCount property 3-15
Release method 3-16
SBSpace property 3-15
Seek method 3-17
Size property 3-15
ToFile method 3-17
Truncate method 3-17
Unlock method 3-17
Write method 3-17
IfxClob class properties
Properties
of IfxClob class 3-14
IfxCommand class 3-18, 3-21
Cancel method 3-19
CommandText property 3-18
CommandTimeout property 3-18
CommandType property 3-18
Connection property 3-18
constructors 3-18
CreateParameter method 3-19
creating 3-18
example code 3-20
ExecuteNonQuery method 3-19
ExecuteReader method 3-20
ExecuteScalar method 3-20
methods 3-19
Parameters property 3-18
Prepare method 3-20
properties 3-18
Transaction property 3-19
UpdatedRowSource property 3-19
IfxCommand.ExecuteNonQuery() method 4-3
IfxCommandBuilder class 1-4, 1-6, 3-21, 3-23
ConflictOption property 3-22
constructors 3-22
creating 3-22
DataAdapter property 3-22
DeriveParameters method 3-22
example of use 3-23, 4-4
GetDeleteCommand method 3-22
GetInsertCommand method 3-22
GetUpdateCommand method 3-22
methods of 3-22
properties of 3-22
QuotePrefix property 3-22
QuoteSuffix property 3-22
RefreshSchema method 3-23
IfxConnection class 3-23, 3-29
BeginTransaction method 3-27
ChangeDatabase method 3-28
ClientLocale property 3-24
Close method 3-28
connecting to a database 1-5
ConnectionString property 3-24
ConnectionTimeout property 3-24
CreateCommand method 3-28
creating 3-23
creating visually 1-5
Database property 3-24
DatabaseLocale property 3-24
Disposed event 3-28

IfxConnection class *(continued)*
events of 3-28
FetchBufferSize property 3-24
GetActiveConnectionsCount property 3-24
GetIdleConnectionsCount property 3-24
GetIfxBlob method 3-28
GetIfxClob method 3-28
InfoMessage event 3-28
methods 3-27
Open method 1-5, 3-28
PacketSize property 3-24
properties 3-24
ServerVersion property 3-24
State property 3-24
StateChange event 3-28
UserDefinedTypeFormat property 3-24
IfxConnection class constructors 3-24
IfxConnection object
Close property
use of 1-5
ConnectionString property
defining visually 1-5
use of 1-5
IfxConnectionStringBuilder class 3-29, 3-30
Add method 3-30
AppendKeyValuePair method 3-30
Clear method 3-30
ConnectionString property 3-29
ContainsKey method 3-30
Count property 3-29
creating 3-29
EquivalentTo method 3-30
IsFixedSize property 3-29
IsReadOnly property 3-29
Keys property 3-29
methods 3-30
properties 3-29
Remove method 3-30
ToString method 3-30
TryGetValue method 3-30
Values property 3-29
IfxDataAdapter
examples of use 3-32
IfxDataAdapter class 1-6, 2-1, 3-21, 3-30, 3-33
AcceptChangesDuringFill property 3-31
AcceptChangesDuringUpdate property 3-31
constructors 3-30
creating 3-30
DeleteCommand property 3-31
example of use 4-3
Fill method 3-32
FillLoadOption property 3-31
FillSchema method 3-32
GetFillParameters method 3-32
InsertCommand property 3-31
methods 3-32
MissingMappingAction property 3-31
MissingSchemaAction property 3-31
properties 3-31
ReturnProviderSpecifictypes property 3-31
SelectCommand property 3-31
TableMappings property 3-31
Update method 3-32
UpdateBatchSize property 3-31
UpdateCommand property 3-31
IfxDataReader class 3-33, 3-36, 4-2
Close method 3-34
IfxDataReader class *(continued)*
Depth property 3-34
example of use 3-35, 4-2
FieldCount property 3-34
GetBoolean method 3-34
GetByte method 3-34
GetBytes method 3-34
GetChar method 3-34
GetChars method 3-34
GetData method 3-34
GetDataTypeName method 3-34
GetDateTime method 3-34
GetDecimal method 3-34
GetDouble method 3-34
GetFieldType method 3-34
GetFloat method 3-35
GetGuid method 3-35
GetInt16 method 3-35
GetInt32 method 3-35
GetInt64 method 3-35
GetName method 3-35
GetOrdinal method 3-35
GetSchemaTable method 3-35
GetString method 3-35
GetTimeSpan method 3-35
GetValue method 3-35
GetValues method 3-35
IsClosed property 3-34
IsDBNull method 3-35
methods 3-34
NextResult method 3-35
properties 3-34
Read method 3-35
RecordsAffected property 3-34
VisibleFieldCount property 3-34
IfxDataReader.GetString() method 2-2
IfxDataSourceEnumerator class 3-36, 3-37
creating 3-36
GetDataSources method 3-37
Instance property 3-36
methods 3-37
properties 3-36
IfxDateTime structure 3-37, 3-44
Add method 3-40
AddDays method 3-40
AddMilliseconds method 3-40
AddMinutes method 3-40
AddMonths method 3-41
AddSeconds method 3-41
AddYears method 3-41
Compare method 3-41
CompareTo method 3-42
constructors 3-38
creating 3-38
Date property 3-39
Day property 3-39
DaysInMonth method 3-42
default values 3-38
EndTimeUnit property 3-39
Equals method 3-42
GetHashCode method 3-42
GreaterThan method 3-42
GreaterThanOrEqual method 3-42
Hour property 3-39
LessThan method 3-43
LessThanOrEqual method 3-43
limits of 3-37

IfxDateTime structure *(continued)*
MaxValue property 3-39
methods 3-40
Millisecond property 3-39
Minute property 3-39
MinValue property 3-39
Month property 3-39
NotEquals method 3-43
Now property 3-39
Null property 3-39
Parse method 3-43
properties 3-39
Second property 3-39
StartTimeUnit property 3-39
Ticks property 3-40
time units in 3-37
Today property 3-40
ToString method 3-44
Year property 3-40
IfxDecimal structure 3-44, 3-49
Abs method 3-45
Add method 3-45
Ceiling method 3-45
Clone method 3-46
Compare method 3-46
CompareTo method 3-46
constructors 3-44
creating 3-44
Divide method 3-46
E property 3-45
Equals method 3-46
Floor method 3-46
GetHashCode method 3-47
GreaterThan method 3-47
GreaterThanOrEqual method 3-47
IsFloating property 3-45
IsNull property 3-45
IsPositive property 3-45
LessThan method 3-47
LessThanOrEqual method 3-47
Max method 3-47
MaxPrecision property 3-45
MaxValue property 3-45
methods 3-45
Min method 3-47
MinusOne property 3-45
MinValue property 3-45
Modulo method 3-48
Multiply method 3-48
Negate method 3-48
NotEquals method 3-48
Null property 3-45
One property 3-45
Parse method 3-48
Pi property 3-45
properties 3-45
Remainder method 3-48
Round method 3-48
Subtract method 3-49
ToString method 3-49
Truncate method 3-49
example of use 3-49
Zero property 3-45
IFXDOTNETTRACE environment variable 1-9
IFXDOTNETTRACEFILE environment variable 1-9
IfxError class 3-49, 3-50
Message property 3-50
IfxError class *(continued)*
NativeError property 3-50
properties 3-50
SQLState property 3-50
IfxErrorCollection class 3-50
Count property 3-50
GetEnumerator method 3-50
methods 3-50
IfxErrorCollection class properties 3-50
IfxException class 3-50, 3-51
Errors property 3-50
HelpLink property 3-50
InnerException property 3-50
Message property 3-50
properties 3-50
StackTrace property 3-50
TargetSite property 3-50
IfxMonthSpan structure 3-51, 3-56
Add method 3-52
Compare method 3-52
CompareTo method 3-53
constructors 3-51
creating 3-51
Divide method 3-53
Duration method 3-53
EndTimeUnit property 3-51
Equals method 3-53
GetHashCode method 3-53
GreaterThan method 3-53
GreaterThanOrEqual method 3-54
IsNull property 3-51
LessThan method 3-54
LessThanOrEqual method 3-54
limits of 3-51
MaxValue property 3-51
MinValue property 3-51
Months property 3-52
Multiply method 3-54
Negate method 3-54
NotEquals method 3-54
Null property 3-52
Parse method 3-54
properties 3-51
StartTimeUnit property 3-52
ToString method 3-55
TotalMonths property 3-52
Years property 3-52
Zero property 3-52
IfxMonthSpan structure methods 3-52
IfxParameter class 3-56, 3-58
constructors 3-56
creating 3-56
DbType property 3-57
Direction property 3-57
example of use 3-57
IfxType property 3-57
IsNullable property 3-57
ParameterName property 3-57
properties 3-57
SourceColumn property 3-57
SourceVersion property 3-57
Value property 3-57
IfxParameter constructors 2-2
IfxParameterCollection class 3-58, 3-60
Add method 3-59
Clear method 3-59
Contains method 3-59

IfxParameterCollection class *(continued)*
CopyTo method 3-59
Count property 3-59
creating 3-58
GetEnumerator method 3-59
IndexOf method 3-60
Insert method 3-60
Item property 3-59
methods 3-59
properties 3-59
Remove method 3-60
RemoveAt method 3-60
IfxProviderFactory class 3-60, 3-61
CreateCommand method 3-60
CreateCommandBuilder method 3-61
CreateConnection method 3-60
CreateConnectionStringBuilder method 3-60
CreateDataAdapter method 3-61
CreateDataSourceEnumerator method 3-61
CreateParameter method 3-61
CreatePermission method 3-61
methods 3-60
IfxSmartLOBCreateTimeFlags enumeration 3-61, 3-62
DontKeepAccessTime member 3-61
KeepAccessTime member 3-62
Log member 3-62
NoLog member 3-62
IfxSmartLOBFileLocation enumeration 3-62
Client member 3-62
Server member 3-62
IfxSmartLOBLockMode enumeration 3-62, 3-63
Exclusive member 3-62
Shared member 3-62
IfxSmartLOBOpenMode enumeration 3-63
Append member 3-63
Buffer member 3-63
DirtyRead member 3-63
LockAll member 3-63
LockRange member 3-63
NoBuffer member 3-63
ReadOnly member 3-63
ReadWrite member 3-63
WriteOnly member 3-63
IfxSmartLOBWhence enumeration 3-63
Begin member 3-63
Current member 3-63
End member 3-63
IfxTimeSpan structure 3-63, 3-69
Add method 3-66
Compare method 3-66
CompareTo method 3-66
constructors 3-64
creating 3-64
Days property 3-65
Divide method 3-67
Duration method 3-67
EndTimeUnit property 3-65
Equals method 3-67
GetHashCode method 3-67
GreaterThan method 3-67
GreaterThanOrEqual method 3-67
Hours property 3-65
IsNull property 3-65
LessThan method 3-67
LessThanOrEqual method 3-68
MaxScale property 3-65
MaxValue property 3-65
IfxTimeSpan structure *(continued)*
methods 3-66
Milliseconds property 3-65
Minutes property 3-65
MinValue property 3-65
Negate method 3-68
NotEquals method 3-68
Null property 3-65
Parse method 3-68
properties 3-65
Seconds property 3-65
StartTimeUnit property 3-65
Ticks property 3-66
ToString method 3-69
Zero property 3-66
IfxTimeUnit enumeration 3-69, 3-70
Day member 3-69
Fraction member 3-70
Fraction1 member 3-69
Fraction2 member 3-70
Fraction3 member 3-70
Fraction4 member 3-70
Fraction5 member 3-70
Hour member 3-69
Minute member 3-69
Month member 3-69
Second member 3-69
Year member 3-69
IfxTransaction class 3-70, 3-71
Commit method 3-70
Connection property 3-70
example of use 3-70
IsolationLevel property 3-70
methods 3-70
properties 3-70
Rollback method 3-70
IfxType argument 2-2
IfxType enumeration 3-71, 4-1
Bigint member 3-71
BigSerial member 3-71
Blob member 3-71
Boolean member 3-71
Byte member 3-71
Char member 3-71
Char1 member 3-71
Clob member 3-71
Collection member 3-71
Date member 3-71
DateTime member 3-71
Decimal member 3-71
Float member 3-71
Int8 member 3-71
Integer member 3-71
IntervalDayFraction member 3-71
IntervalYearMonth member 3-71
List member 3-71
LVarChar member 3-71
Money member 3-71
MultiSet member 3-71
NChar member 3-71
NVarChar member 3-71
Row member 3-71
Serial member 3-71
Serial8 member 3-72
Set member 3-72
SmallFloat member 3-72
SmallInt member 3-72

IfxType enumeration *(continued)*
SmartLOBLocator member 3-72
Text member 3-72
VarChar member 3-72
IfxType property
of IfxParameter class 3-57
Ignore value 3-31, 3-32
Importing a namespace 1-5
IndexOf method
of IfxParameterCollection class 3-60
InfoMessage event
of IfxConnection class 3-28
Informix Client SDK 1-2
Informix data types 3-57
Informix ODBC Driver 1-1
Informix OLE DB Provider 1-1
Informix types 3-71
InnerException property
of IfxException class 3-50
Input parameters 1-7, 3-57
Insert method
of IfxParameterCollection class 3-60
INSERT statements 3-21
INSERT, automatic 1-6
InsertCommand property 1-6
of IfxDataAdapter class 3-31
Inserting rows, example of 4-3
Installing
IBM Informix .NET Provider Add-in for Microsoft Visual Studio 1-2
Installing IBM Informix .NET Provider 1-2
Instsance property
of IfxDataSourceEnumerator class 3-36
Int32 type 2-2
Int64 type 2-2
Int64[] type 2-1
Int8 member
of IfxType enumeration 3-71
INT8 type 2-2
Integer member
of IfxType enumeration 3-71
INTEGER type 2-2
Internet protocol version 6 (IPv6) 1-2
INTERVAL type 2-2
IntervalDayFraction member
of IfxType enumeration 3-71
IntervalYearMonth member
of IfxType enumeration 3-71
IP address 3-25
IP addresses
IPv6 format in sqlhosts 1-2
IPv6.
See Internet protocol version 6.
IsClosed property
of IfxDataReader class 3-34
IsDBNull method
of IfxDataReader class 3-35
IsFixedSize property
of IfxConnectionStringBuilder class 3-29
IsFloating property
of IfxDecimal structure 3-45
IsNull property
of IfxBlob 3-9
of IfxClob 3-14
of IfxDecimal structure 3-45
of IfxMonthSpan structure 3-51
of IfxTimeSpan structure 3-65
IsNullable property
of IfxParameter class 3-57
IsolationLevel property
of IfxTransaction class 3-70
IsOpen property
of IfxBlob 3-9
of IfxClob 3-14
IsPositive property
of IfxDecimal structure 3-45
IsReadOnly property
of IfxConnectionStringBuilder class 3-29
Item property
of IfxParameterCollection class 3-59

J

JOIN operators 1-6

K

KeepAccessTime member
of IfxSmartLOBCreateTimeFlags enumeration 3-62
KeyInfo value 3-20
Keys property
of IfxConnectionStringBuilder class 3-29

L

Language locale 3-25
LastAccessTime property
of IfxBlob 3-9
of IfxClob 3-14
LastChangeTime property
of IfxBlob 3-10
of IfxClob 3-15
LastModificationTime property
of IfxBlob 3-10
of IfxClob 3-15
LessThan method
of IfxDateTime structure 3-43
of IfxDecimal structure 3-47
of IfxMonthSpan structure 3-54
of IfxTimeSpan structure 3-67
LessThanOrEqual method
of IfxDateTime structure 3-43
of IfxDecimal structure 3-47
of IfxMonthSpan structure 3-54
of IfxTimeSpan structure 3-68
List member
of IfxType enumeration 3-71
LIST type 2-2
Local transactions
example of use 4-3
Locales 3-25
Lock method
of IfxBlob class 3-11
of IfxClob class 3-16
LockAll member
of IfxSmartLOBOpenMode enumeration 3-63
LockRange member
of IfxSmartLOBOpenMode enumeration 3-63
Log member
of IfxSmartLOBCreateTimeFlags enumeration 3-62
LVarChar member
of IfxType enumeration 3-71
LVARCHAR type 2-2

M

Max method
of IfxDecimal structure 3-47
Max Pool Size attribute 1-7
Max Pool Size connection string attribute 3-25
MAX ROWS parameter 1-4
MaxBytes property
of IfxBlob 3-10
of IfxClob 3-15
MaxPrecision property
of IfxDecimal structure 3-45
MaxScale property
of IfxTimeSpan structure 3-65
MaxValue property
of IfxDateTime structure 3-39
of IfxDecimal structure 3-45
of IfxMonthSpan structure 3-51
of IfxTimeSpan structure 3-65
Message property
of IfxError class 3-50
of IfxException class 3-50
Messages 3-49
Methods
of IfxClob class 3-15
of IfxCommand class 3-19
of IfxCommandBuilder class 3-22
of IfxConnection class 3-27
of IfxConnectionStringBuilder class 3-30
of IfxDataAdapter class 3-32
of IfxDataReader class 3-34
of IfxDataSourceEnumerator class 3-37
of IfxDateTime structure 3-40
of IfxDecimal structure 3-45
of IfxErrorCollection class 3-50
of IfxMonthSpan structure 3-52
of IfxParameterCollection class 3-59
of IfxProviderFactory class 3-60
of IfxTimeSpan structure 3-66
of IfxTransaction class 3-70
Methods, unsupported 3-6
Microsoft .NET Framework SDK 1-2
Microsoft .NET specification vii
Microsoft ODBC .NET 1-1
Microsoft OLE DB .NET 1-1
Microsoft public .NET base classes 3-6
Microsoft public .NET interfaces 3-5
Microsoft Visual Studio Add-in
Installing 1-2
Millisecond property
of IfxDateTime structure 3-39
Milliseconds property
of IfxTimeSpan structure 3-65
Min method
of IfxDecimal structure 3-47
Min Pool Size attribute 1-7
Min Pool Size connection string attribute 3-25
MinusOne property
of IfxDecimal structure 3-45
Minute member
of IfxTimeUnit enumeration 3-69
Minute property
of IfxDateTime structure 3-39
Minutes property
of IfxTimeSpan structure 3-65
MinValue property
of IfxDateTime structure 3-39
of IfxDecimal structure 3-45
MinValue property *(continued)*
of IfxMonthSpan structure 3-51
of IfxTimeSpan structure 3-65
MissingMappingAction property 3-32
of IfxDataAdapter class 3-31
MissingSchemaAction property
of IfxDataAdapter class 3-31
Modulo method
of IfxDecimal structure 3-48
Money member
of IfxType enumeration 3-71
MONEY type 2-2
Month member
of IfxTimeUnit enumeration 3-69
Month property
of IfxDateTime structure 3-39
Months property
of IfxMonthSpan structure 3-52
Multiply method
of IfxDecimal structure 3-48
of IfxMonthSpan structure 3-54
MultiSet member
of IfxType enumeration 3-71
MULTISET type 2-2

N

Namespace
importing 1-5
of IBM Informix .NET Provider 1-4
NativeError property
of IfxError class 3-50
NChar member
of IfxType enumeration 3-71
NCHAR type 2-2
Negate method
of IfxDecimal structure 3-48
of IfxMonthSpan structure 3-54
of IfxTimeSpan structure 3-68
Nested string literals 2-2
NextResult method
of IfxDataReader class 3-35
Nobuffer member
of IfxSmartLOBOpenMode enumeration 3-63
NoLog member
of IfxSmartLOBCreateTimeFlags enumeration 3-62
None value 3-19
NotEquals method
of IfxDateTime structure 3-43
of IfxDecimal structure 3-48
of IfxMonthSpan structure 3-54
of IfxTimeSpan structure 3-68
Now property
of IfxDateTime structure 3-39
Null property
of IfxBlob 3-10
of IfxClob 3-15
of IfxDateTime structure 3-39
of IfxDecimal structure 3-45
of IfxMonthSpan structure 3-52
of IfxTimeSpan structure 3-65
Null values 3-57
NVarChar member
of IfxType enumeration 3-71

O

ODBC .NET 1-1
ODBC connection string 3-26
ODBC Driver 1-1
OLE DB .NET 1-1
OLE DB Provider 1-1
One property
  of IfxDecimal structure 3-45
Open method
  of IfxBlob class 3-11
  of IfxClob class 3-16
  of IfxConnection class 3-28
Optimize OpenFetchClose connection string attribute 3-25
OPTOFC connection string attribute 3-25
OPTOFC environment variable 1-5
Output parameters 1-6, 3-57
OutputParameters value 3-19

P

Packet Size connection string attribute 3-25
PacketSize property
  of IfxConnection class 3-24
Parameter arrays 1-7
Parameter types 2-2
ParameterCollection.Add method 1-7
ParameterName property
  of IfxParameter class 3-57
Parameters property
  of IfxCommand 3-18
Parse method
  of IfxDateTime structure 3-43
  of IfxDecimal structure 3-48
  of IfxMonthSpan structure 3-54
  of IfxTimeSpan structure 3-68
Passthrough value 3-31
Password connection string attribute 3-25
Performance 3-33
Persist Security Info attribute 3-26
Persist Security Info connection string attribute 3-25
Pi property
  of IfxDecimal structure 3-45
Placeholder symbols 1-7
Platforms 1-1
Pooling attribute 1-7, 1-8
Pooling connection string attribute 3-25
Port numbers 3-26
Position property
  of IfxBlob 3-10
  of IfxClob 3-15
Prepare method
  of IfxCommand class 3-20
Primary key 1-6
PRO connection string attribute 3-25
Properties
  IfxErrorCollection class 3-50
  of IfxBlob class 3-9
  of IfxCommand class 3-18
  of IfxCommandBuilder class 3-22
  of IfxConnection class 3-24
  of IfxConnectionStringBuilder class 3-29
  of IfxDataAdapter class 3-31
  of IfxDataReader class 3-34
  of IfxDataSourceEnumerator class 3-36
  of IfxDateTime structure 3-39
  of IfxDecimal structure 3-45
Properties *(continued)*
  of IfxError class 3-50
  of IfxException class 3-50
  of IfxMonthSpan structure 3-51
  of IfxParameter class 3-57
  of IfxParameterCollection class 3-59
  of IfxTimeSpan structure 3-65
  of IfxTransaction class 3-70
Protocol connection string attribute 3-25
Provider independence 1-4
Public base classes 3-6
Public interfaces 3-5
Public interfaces, objects 1-3
PWD connection string attribute 3-25

Q

QuotePrefix property 1-7
  of IfxCommandBuilder class 3-22
QuoteSuffix property 1-7
  of IfxCommandBuilder class 3-22

R

range of time types 3-37, 3-51
Read method
  of IfxBlob class 3-11
  of IfxClob class 3-16
  of IfxDataReader class 3-35
Read-only data 1-4, 3-33, 4-2
ReadOnly member
  of IfxSmartLOBOpenMode enumeration 3-63
ReadWrite member
  of IfxSmartLOBOpenMode enumeration 3-63
REAL type 2-2
Reconciling changed data 1-4, 1-6, 3-21, 3-30, 4-4
RecordsAffected property
  of IfxDataReader class 3-34
ReferenceCount property
  of IfxBlob 3-10
  of IfxClob 3-15
RefreshSchema method 1-6
  of IfxCommandBuilder class 3-23
Registry 1-5
Relationships 1-4
Release method
  of IfxBlob class 3-11
  of IfxClob class 3-16
Remainder method
  of IfxDecimal structure 3-48
Remove method
  of IfxConnectionStringBuilder class 3-30
  of IfxParameterCollection class 3-60
RemoveAt method
  of IfxParameterCollection class 3-60
Retrieving a single value, example of 4-1
Retrieving data 4-2
Retrieving data into a DataSet
  example of 4-3
Retrieving multiple rows
  example of 4-2
ReturnProviderSpecifictypes property
  of IfxDataAdapter class 3-31
Rollback method
  of IfxTransaction class 3-70

Round method
of IfxDecimal structure 3-48
Row member
of IfxType enumeration 3-71
RowUpdating events 3-21

S

Sample programs 4-1
SBSpace property
of IfxBlob 3-10
of IfxClob 3-15
SchemaOnly value 3-20
Second member
of IfxTimeUnit enumeration 3-69
Second property
of IfxDateTime structure 3-39
Seconds property
of IfxTimeSpan structure 3-65
Seek method
of IfxBlob class 3-12
of IfxClob class 3-17
SELECT statements 1-4
SelectCommand property 1-6, 3-21
of IfxDataAdapter class 3-31
SequentialAccess value 3-20
Serial member
of IfxType enumeration 3-71
SERIAL type 2-2
Serial8 member
of IfxType enumeration 3-72
Server connection string attribute 3-26
Server member
of IfxSmartLOBFileLocation enumeration 3-62
ServerVersion property
of IfxConnection class 3-24
Service connection string attribute 3-26
Set member
of IfxType enumeration 3-72
SET type 2-2
Setnet utility 1-5
Shared member
of IfxSmartLOBLockMode enumeration 3-62
Shortcut keys
keyboard A-1
Single-quote characters 2-2
Single-table updates 3-21
SingleResult value 3-20
SingleRow value 3-20
Size property
of IfxBlob 3-10
of IfxClob 3-15
SmallFloat member
of IfxType enumeration 3-72
SMALLFLOAT type 2-2
SmallInt member
of IfxType enumeration 3-72
SmartLOBLocator member
of IfxType enumeration 3-72
SourceColumn property
of IfxParameter class 3-57
SourceVersion property
of IfxParameter class 3-57
Special characters 1-7
sqlhosts
IPv6 IP addresses in 1-2
using hostnames in 1-2
SQLState property
of IfxError class 3-50
StackTrace property
of IfxException class 3-50
start time unit 3-37, 3-51
StartTimeUnit property
of IfxDateTime structure 3-39
of IfxMonthSpan structure 3-52
of IfxTimeSpan structure 3-65
State property
of IfxConnection class 3-24
StateChange event
of IfxConnection class 3-28
Stored procedures 1-7, 3-22, 4-5
StoredProcedure value 3-19
String literals 2-2
String type 2-1
Strong-named assembly 1-2
Subtract method
of IfxDecimal structure 3-49
sysmaster database 1-3
System.DataSet class
Reconciling changes with the database
example of 4-4
SystemException exception 3-31

T

TableDirect value 3-19
TableMappings property 3-32
of IfxDataAdapter class 3-31
TargetSite property
of IfxException class 3-50
Text member
of IfxType enumeration 3-72
TEXT type 2-2
Text value 3-19
Thread-safety of provider 1-4
Ticks property
of IfxDateTime structure 3-40
of IfxTimeSpan structure 3-66
Time units
in IfxDateTime 3-37
Today property
of IfxDateTime structure 3-40
ToFile method
of IfxBlob class 3-12
of IfxClob class 3-17
ToString method
of IfxConnectionStringBuilder class 3-30
of IfxDateTime structure 3-44
of IfxDecimal structure 3-49
of IfxMonthSpan structure 3-55
of IfxTimeSpan structure 3-69
TotalMonths property
of IfxMonthSpan structure 3-52
Tracing 1-9
Transaction property 1-6, 3-70
of IfxCommand 3-19
Transactions 3-70
distributed 1-8, 3-25, 4-6
local
example of use 4-3
Truncate method
of IfxBlob class 3-12
of IfxClob class 3-17
of IfxDecimal structure 3-49

TryGetValue method
of IfxConnectionStringBuilder class 3-30

U

UID connection string attribute 3-26
Unlock method
of IfxBlob class 3-12
of IfxClob class 3-17
Unsupported methods 3-6
Update method
of IfxDataAdapter class 3-32
UPDATE statements 3-21
UPDATE, automatic 1-6
UpdateBatchSize property
of IfxDataAdapter class 3-31
UpdateCommand property 1-6
of IfxDataAdapter class 3-31
UpdatedRowSource property
of IfxCommand 3-19
User ID connection string attribute 3-26
UserDefinedTypeFormat
attributes of 3-26
UserDefinedTypeFormat property
attributes of 3-26
of IfxConnection class 3-24

V

Value property
of IfxParameter class 3-57
Values property
of IfxConnectionStringBuilder class 3-29
VarChar member
of IfxType enumeration 3-72
VARCHAR type 2-2
VisibleFieldCount property
of IfxDataReader class 3-34
Visual BASIC .NET 1-1
Visual C# .NET 1-1
Visual J# .NET 1-1

W

Warning messages 3-49
Write method
of IfxBlob class 3-12
of IfxClob class 3-17
WriteOnly member
of IfxSmartLOBOpenMode enumeration 3-63

X

XCL connection string attribute 3-25

Y

Year member
of IfxTimeUnit enumeration 3-69
Year property
of IfxDateTime structure 3-40
Years property
of IfxMonthSpan structure 3-52

Z

Zero property
of IfxDecimal structure 3-45
of IfxMonthSpan structure 3-52
of IfxTimeSpan structure 3-66

Printed in USA

SC23-9425-01