

HCL Informix 14.10 - .NET Provider Reference Guide



Contents

| | |
|--|----------|
| Chapter 1. Informix® .NET Provider Guide..... | 3 |
| Overview of HCL Informix® .NET Provider..... | 3 |
| What is the Informix® .NET Provider?..... | 3 |
| Supported programming environments..... | 3 |
| Support for IPv6..... | 4 |
| Installing the HCL Informix® .NET Provider..... | 4 |
| Overview of the .NET provider class library..... | 5 |
| Thread-safety of provider types..... | 6 |
| Namespace requirements..... | 6 |
| Connecting to a database..... | 7 |
| The ? parameter markers..... | 9 |
| Call stored procedures..... | 10 |
| IfxProviderFactory objects to write database- independent code..... | 10 |
| Distributed transactions..... | 10 |
| The OUT and INOUT Parameters..... | 10 |
| Generic coding with the ADO.NET common base classes..... | 13 |
| Error messages..... | 14 |
| Mapping data types..... | 14 |
| Retrieve data..... | 15 |
| Set data types for a parameter..... | 16 |
| Display format of FLOAT, DECIMAL, or MONEY data types..... | 17 |
| Type reference..... | 17 |
| Supported public .NET interfaces..... | 17 |
| Supported Public .NET base classes..... | 18 |
| Prototype syntax..... | 19 |
| IfxBlob class..... | 20 |
| IfxClob class..... | 25 |
| IfxCommand class..... | 30 |
| IfxCommandBuilder class..... | 35 |
| IfxConnection class..... | 37 |
| IfxConnectionStringBuilder class..... | 45 |
| IfxDataAdapter class..... | 47 |
| IfxDataReader class..... | 50 |
| IfxDataSourceEnumerator class..... | 53 |
| IfxDateTime structure..... | 57 |
| IfxDecimal structure..... | 65 |
| IfxError class..... | 71 |
| IfxErrorCollection class..... | 71 |
| IfxException class..... | 72 |
| IfxMonthSpan structure..... | 72 |
| IfxParameter class..... | 77 |
| IfxParameterCollection class..... | 80 |
| IfxProviderFactory class..... | 82 |

| | |
|---|------------|
| IfxSmartLOBCreateTimeFlags enumeration..... | 83 |
| IfxSmartLOBFileLocation enumeration..... | 84 |
| IfxSmartLOBLocator class..... | 85 |
| IfxSmartLOBLockMode enumeration..... | 85 |
| IfxSmartLOBOpenMode enumeration..... | 85 |
| IfxSmartLOBWhence enumeration..... | 86 |
| IfxTimeSpan structure..... | 86 |
| IfxTimeUnit enumeration..... | 92 |
| IfxTransaction class..... | 93 |
| IfxType enumeration..... | 94 |
| Sample programs..... | 95 |
| Demonstration programs..... | 95 |
| Informix® .NET Provider examples..... | 95 |
| Index..... | 104 |

Chapter 1. Informix® .NET Provider Guide

These topics contain the information you need in order to use the Informix® .NET Provider to access and manipulate data in HCL Informix® databases.

These topics assume you are familiar with the Microsoft™ .NET specification, object-oriented programming principles, and using HCL Informix® servers and databases.

Microsoft™ provides information about programming with .NET on its website. For more information about working with HCL Informix®, see the release notes in your server documentation set.

These topics are taken from *IBM® Informix® .NET Provider Reference Guide*.

Overview of HCL Informix® .NET Provider

The topics in this overview describe the Informix® .NET Provider and provide information on the environment in which to use it. These topics also provide installation and connection information and general information to help you get started using the Informix® .NET Provider.

What is the Informix® .NET Provider?

The Informix® .NET Provider is a .NET assembly that lets .NET applications access and manipulate data in 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 Informix® .NET Provider is more efficient than accessing the Informix® database through either of these two methods:

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

Related information

[Differences between .NET Providers on page](#)

Supported programming environments

The Informix® .NET Provider can be used by any application that can be run by the Microsoft™ .NET Framework.

The following list includes examples of supported programming environments:

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

The Informix® .NET Provider runs on all Microsoft™ Windows™ operating systems that support all .NET Framework features. If you want to use the Informix® .NET Provider that implements ADO.NET interfaces, you must have the corresponding version of Microsoft™ .NET Framework installed on your system. The Informix® .NET Provider that implements ADO.NET interfaces is available with HCL® Informix® Client Software Development Kit (Client SDK) 3.0 and later releases.

Support for IPv6

The 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 HCL Informix® .NET Provider

You can install the Informix® .NET Provider with the HCL® Informix® Client Software Development Kit (Client SDK) through a typical or custom installation.

The Informix® .NET Provider installation registers strong-named assemblies in the Global Assembly Cache (GAC). If your application is not running in a debugger, the application automatically finds the assembly that is registered in the GAC. If you are upgrading .NET applications on 64-bit Windows™, you must clear the GAC to load the new version of the Informix® .NET Provider DLL. Clear the GAC with the gacutil utility using the following command:

```
gacutil /u IBM.Data.Informix
```

You can perform this operation before upgrading HCL Informix® .NET Provider.

The locations of the HCL Informix® .NET Framework assemblies are shown in the following table:

Table 1. Locations of the .NET Framework assemblies

| Framework that the assembly implements | Location |
|--|--------------------------|
| Informix® .NET Provider 2.0 Framework | %INFORMIXDIR%\bin\netf20 |
| Informix® .NET Provider 4.0 Framework | %INFORMIXDIR%\bin\netf40 |

Use the Informix® .NET Framework 2.0 Provider for .NET Framework 2.0, 3.0, and 3.5.

Use the Informix® .NET Framework 4.0 Provider for .NET Framework 4.0 and 4.5.

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

1. In the **Solution Explorer** window, right-click **References**.
2. Click **Browse** and go to the assembly. The file name of the assembly is `IBM.Data.Informix.dll`. Select the file from the folder for the version that you want to use.

- %INFORMIXDIR%\bin\netf11
- %INFORMIXDIR%\bin\netf20
- %INFORMIXDIR%\bin\netf40

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**.

Related information

[Installing Client Products on UNIX, Linux, Mac OS X on page](#)

[Installing Client Products on Windows Systems on page](#)

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 the `IfxDotNetIntrinsicModule.dll` file.

Set your **PATH** environment variable to include the path to `IfxDotNetIntrinsicModule.dll` as follows:

- %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.

Prepare the database server

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

Overview of the .NET provider class library

The Informix® .NET Provider supports all of the .NET public classes and base classes that are needed to access the Informix® database.

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 the `DataSet` object 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 stored on the client. Keep your data sets as small as possible. If you do not need to return all the data, use a projection clause in the SELECT statement to limit the returned rows.

Following are the main Informix® .NET Provider classes that provide data access:

- `IfxConnection`: Connection to a database and management of transactions.
- `IfxCommand`: Issues SQL commands.
- `IfxDataReader`: Reads a forward-only stream of data records.
- `IfxTransaction`: Controls transactions.
- `IfxDataAdapter`: Pushes data into a data set and for reconciling changes in a data set with the database.

You can use the following .NET Provider classes to develop provider-independent code:

- `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 update the correct row in the table. For single-table updates, you can use the `IfxCommandBuilder` class to automatically reconcile changes in the data set with the database.

Related reference

[Supported public .NET interfaces on page 17](#)

[Supported Public .NET base classes on page 18](#)

Related information

[Reconcile DataSet changes with the database on page 8](#)

[IfxCommandBuilder class on page 35](#)

[Projection Clause on page](#)

Thread-safety of provider types

Only static members of the 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 Informix® .NET Provider is: `IBM.Data.Informix`. This means that the full name of the objects in the Informix® .NET Provider all begin with `IBM.Data.Informix`.

For example, 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;
```

The Informix® .NET Provider supports the `System.Transaction` namespace, but does not support the `System.EnterpriseServices` namespace.

Connecting to a database

You connect to a database by using the `Open` method of an `IfxConnection` object.

About this task

You define information about how to connect to the database (such as the machine and server where the database is located) 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 the topic [IfxConnection class on page 37](#).

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. Complete the items of the dialog to provide the connection information.

Results

The following fragment shows a simple connection to a database called **testdb** on the HCL Informix® server that is called `testserver` that is located on a machine named `berry`:

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

An `IfxConnection` object can also determine the connection properties from the environment variables. If you set the properties in the connection string, the `IfxConnection` object uses those values. If you do not set the properties in the connection string, the `IfxConnection` object uses the values that are set by the environment.

However, you can set the value for the DELIMIDENT property in the registry by using the Setnet utility. If you do not set the value for the DELIMIDENT property in the connection string or in the environment, but you set it by using the Setnet utility, the `IfxConnection` object uses the value that you set with the Setnet utility.

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

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

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

Connection string attribute names are not case-sensitive, but often their values are not.

Related information

[The DELIMIDENT Environment Variable on page](#)

[DELIMIDENT environment variable on page](#)

Reconcile 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.

[Retrieve data into a DataSet on page 98](#) includes an example that demonstrates how to use `IfxDataAdapter` objects. [IfxCommandBuilder object to reconcile changes with the database on page 99](#) 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 `InvalidOperationException` 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 [Type reference on page 17](#).

The connection pool

Connection pooling allows client applications to reuse connections instead of creating a new one each time the 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.

The `Idle Timeout` internal parameter is the standard protocol for removing connections from the pool and prevents connections from remaining active indefinitely in the server. `Idle Timeout` has a value of 120 seconds, which cannot be changed by setting a new value in the connection string. With `Idle Timeout`, when a connection is unused in the connection pool for more than 120 seconds, the connection is closed and removed from the pool.

Set FullTrust permission

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

The ? parameter markers

You can use the question mark symbol (?) to mark a parameter's place in an SQL statement or stored procedure.

Because the 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.

Call stored procedures

To use stored procedures in your applications, you must set some properties of the `IfxCommand` object.

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 topic [Call a stored procedure on page 100](#) includes an example that shows how to run a stored procedure and read any results that it returns.

IfxProviderFactory objects to write database-independent code

Starting with Informix® .NET Provider Version 2, you can use the `IfxProviderFactory` class to write database-independent code.

For more information about the `IfxProviderFactory` class, see [IfxProviderFactory class on page 82](#).

Distributed transactions

Your application can enlist a connection for distributed transactions by setting the `Enlist` connection string attribute to `true`, `yes`, or `1`.

You should set the `Pooling` connection string attribute to `true`, `yes`, or `1` while working with distributed transactions.

[Distributed transactions on page 100](#) includes an example of how to use distributed transactions with your application.

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.

The OUT and INOUT Parameters

As of Version 3.50.xC4, HCL Informix® Client Software Development Kit supports the use of OUT and INOUT parameters during execution of SPL.

The following data types are supported:

- BIGINT
- BLOB
- BOOLEAN
- DATETIME
- CHAR
- CLOB
- DECIMAL
- FLOAT
- INT8
- INTEGER
- INTERVAL
- LVARCHAR
- MONEY
- NCHAR
- NVARCHAR
- SMALLFLOAT
- SMALLINT
- VARCHAR

These restrictions exist when using OUT or INOUT parameters in SPL execution:

- Collection data types such as LIST, MULTISSET, ROW and SET are not supported.
- Returning result sets is not supported. After executing SPL with OUT or INOUT parameters, you cannot call **SQLFetch** or **SQL GetData**.
- Only one value can be returned; that is, only one set of OUT or INOUT parameters can be returned per individual SPL execution.

The following SPL execution example creates one OUT, one INOUT, and one IN (default) parameter and one return value.

```
create procedure myproc(OUT intparam INT, INOUT charparam char(20),
  inparam int) returns int
<body of SPL>
end procedure;
```

The following code example shows how to use OUT and INOUT parameters.

```
using System;
using System.Data;
using System.IO;
using IBM.Data.Informix;

namespace SPLInOutParamTest
{
  class Program
```

```

{
    static void Main(string[] args)
    {
        /* Build connection string and create connection object */
        IfxConnection conn = new IfxConnection("Server=oL_ids1150;
        Database=common_db1;UID=informix;PWD=informix");

        /* Create command object */
        IfxCommand cmd = new IfxCommand();

        /* Connect to the server */
        conn.Open();

        /* Associate connection object to command object */
        cmd.Connection = conn;

        try
        {
            try
            {
                /* Drop the procedure */
                cmd.CommandText = "DROP PROCEDURE test_proc;";
                cmd.ExecuteNonQuery();
            }
            catch { /* Ignore the exception */ };

            /* Create procedure with INOUT params */
            cmd.CommandText = "CREATE PROCEDURE test_proc
            (INOUT arg1 int, OUT arg2 int, INOUT arg3 int) " +
                "returning int " +
                "define ret int; " +
                "let ret = (arg1 + arg3);" +
                "let arg1 = 1; " +
                "let arg2 = 2; " +
                "let arg3 = 3;" +
                "return ret; " +
                "end procedure;";

            cmd.ExecuteNonQuery();

            cmd.CommandText = "{? = call test_proc(?,?,?)};";

            /* Bind the required parameters */
            IfxParameter p1 = cmd.Parameters.Add("ID", IfxType.Integer);
            IfxParameter p2 = cmd.Parameters.Add("ID1", IfxType.Integer);
            IfxParameter p3 = cmd.Parameters.Add("ID2", IfxType.Integer);
            IfxParameter p4 = cmd.Parameters.Add("ID3", IfxType.Integer);

            /* Initialize the values for the parameters */
            p1.Value = 0;
            p2.Value = 5;
            p3.Value = 4;
            p4.Value = 10;

            /* Bind the appropriate direction */
            p1.Direction = ParameterDirection.Output;
            p2.Direction = ParameterDirection.InputOutput;

```


The `DbConnection` object can connect to Informix® databases, just as a `IfxConnection` object, which is inherited from the `DbConnection` object.

By 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 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 HCL Informix® server are represented as Informix® .NET Provider exceptions.

Tracing

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

0

No tracing

1

Tracing of API entry and exit, with return code

2

Tracing of API entry and exit, with return code, plus tracing of parameters to the API

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

Error checking during data transfer

The **IFX_LOB_XFERSIZE** environment variable is used to specify the number of kilobytes in a CLOB or BLOB to transfer from a client application to the database server before checking whether an error has occurred. The error check occurs each time the specified number of kilobytes is transferred.

If an error occurs, the remaining data is not sent and an error is reported. If no error occurs, the file transfer will continue until it finishes.

The valid range for **IFX_LOB_XFERSIZE** is from 1 to 9223372036854775808 kilobytes. The **IFX_LOB_XFERSIZE** environment variable is set on the client.

For more information on **IFX_LOB_XFERSIZE**, see the *HCL® Informix® Guide to SQL: Reference*.

Mapping data types

These topics describe how data types are mapped between HCL Informix® databases and the .NET Framework.

The information on mapping 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)

Retrieve data

Each HCL Informix® data type can fit in a .NET Framework data type.

The following table shows each HCL 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 the Informix® database. The types recommended are the most efficient and least likely to change the value.

Table 2. Best-fit types for retrieving Informix® data types

| Informix® data type | Recommended type | Best-fit native .NET data type |
|--------------------------------|--------------------------|--------------------------------|
| BIGINT | <code>Int64</code> | <code>Int64</code> |
| BIGSERIAL | <code>Int64</code> | <code>Int64</code> |
| BLOB | <code>IfxBlob</code> | <code>Byte[]</code> |
| BOOLEAN | <code>Boolean</code> | <code>Boolean</code> |
| BYTE | <code>Byte[]</code> | <code>Byte[]</code> |
| CHAR | <code>String</code> | <code>String</code> |
| CHAR(1) | <code>String</code> | <code>String</code> |
| CLOB | <code>IfxClob</code> | <code>Byte[]</code> |
| DATE | <code>IfxDateTime</code> | <code>DateTime</code> |
| DATETIME | <code>IfxDateTime</code> | <code>DateTime</code> |
| DECIMAL(p<=28) fixed scale | <code>IfxDecimal</code> | <code>Decimal</code> |
| DECIMAL (p<=28) floating point | <code>IfxDecimal</code> | <code>Double</code> |
| DECIMAL (p>28) | <code>IfxDecimal</code> | <code>String</code> |
| DOUBLE | <code>Double</code> | <code>Double</code> |
| FLOAT | <code>Double</code> | <code>Double</code> |
| IDSSECURITYLABEL | <code>Int64[]</code> | <code>Int64[]</code> |
| INTEGER | <code>Int32</code> | <code>Int32</code> |
| INT8 | <code>Int64</code> | <code>Int64</code> |

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

| Informix® data type | Recommended type | Best-fit native .NET data type |
|------------------------|------------------|------------------------------------|
| 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 |

For the format of Informix® data types, DECIMAL, MONEY, DATETIME, and INTERVAL returned using `IfxDataReader.GetString` method see the section about the Literal Row segment in *HCL® Informix® Guide to SQL: Syntax*.

The ROW and TEXT types and the collection types, LIST, MULTISSET, 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 *HCL® 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.

Set data types for a parameter

Your application should set the type for a parameter as the HCL Informix® type whenever possible (using the `IfxType` argument of `IfxParameter` constructor).

[IfxType enumeration on page 94](#) shows the `IfxType` enumeration.

You can specify a parameter type as a .NET `DbType` instead, and the 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 Informix® .NET Provider may not choose the data type you intend.

If you do not specify either the Informix® data type or a .NET DbType, the Informix® .NET Provider attempts to infer the 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 HCL 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 *HCL® Informix® Guide to SQL: Reference*.

Type reference

All of the classes described in these topics 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

The Informix® .NET Provider implements specific Microsoft™ .NET interfaces.

For more information about the Microsoft™ public interfaces and classes, see the Microsoft™ .NET Framework SDK documentation. If the Informix® .NET Provider does not support a particular .NET class or method, that class or method is implemented as no-operation.

Table 3. Interfaces implemented by 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 Informix® database |

Table 3. Interfaces implemented by Informix® .NET Provider classes (continued)

| Class | Extends | Description |
|------------------------|------------------------|--|
| IfxConnection | IDbConnection | Represents an application's unique session with a data source |
| IfxDataAdapter | IDbDataAdapter | Enables an application to run SQL commands against the database, fill data sets, and reconcile changes in the data set with the database |
| IfxDataReader | IDbDataReader | Allows forward-only, read-only, access to a stream of data from the database |
| IfxError | | Represents an instance of a warning or an error that is generated by the 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 the 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

The Informix® .NET Provider implements specific Microsoft™ .NET base classes.

For more information about the Microsoft™ public classes, see the Microsoft™ .NET Framework SDK documentation. If the Informix® .NET Provider does not support a particular .NET base class, that class is implemented as no-operation.

Table 4. Base classes implemented by 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 Informix® database. |
| IfxConnection | DbConnection | Represents an application's unique session with a data source. |

Table 4. Base classes implemented by 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 run 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 | Allows 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. The TransactionScope class is not supported by this .NET provider |

Prototype syntax

Because the objects of the 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:

```

      .-,-----
      v                                |
>>+-----+--returntype--methodname--(-----+-----)-><
    '-static-'                          '-parmtype--parmlabel-'

```

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 shown in this topic 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 85](#) 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.

Create an IfxBlob

You can get existing `IfxBlob` objects or you can create them.

You can get `IfxBlob` objects from these methods:

- `IfxConnection.GetIfxBlob`
- `IfxDataReader.GetIfxBlob`

You can 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 5. 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 <code>IfxSmartLOBCreateTimeFlags</code> enumeration. See IfxSmartLOBCreateTimeFlags enumeration on page 83 for details. |
| IsNull | System.Boolean | read-only | Returns true if the instance is null; otherwise it returns false. |

Table 5. IfxBlob public properties (continued)

| Property | Type | Access notes | Description |
|----------------------|----------------|------------------|--|
| 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. |
| 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 86](#).

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 85](#) 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 86](#).

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 86](#).

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 86](#).

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 86](#).

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 code set 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 85](#) 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.

Create 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 6. 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. |

Table 6. IfxClob public properties (continued)

| Property | Type | Access notes | Description |
|----------------------|----------------|------------------|--|
| | | | See IfxSmartLOBCreateTimeFlags enumeration on page 83 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. |
| 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. |

Table 6. IfxClob public properties (continued)

| Property | Type | Access notes | Description |
|----------|--------------|--------------|---|
| 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 86](#).

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 85](#) 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 86](#).

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 86](#).

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 86](#).

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 86](#).

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.

Create an IfxCommand

You can create an *IfxCommand* by using the constructors or by using the *IfxConnection.CreateCommand* method.

See [IfxConnection.CreateCommand on page 44](#).

To write provider-independent code, you can use the *CreateCommand()* method of the *DbProviderFactory* class to create a provider-specific instance of *DbCommand*.

Related information

[IfxProviderFactory class on page 82](#)

IfxCommand constructors

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

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 7. IfxCommand public 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 HCL Informix® ODBC Driver supports is also supported by the 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. |
| RowFetchCount | System.Int32 | Sets the number of rows to fetch in each fetch operation. This value affects the performance of the fetch operation. The value is not applicable if the table contains columns of BLOB, CLOB, TEXT or BYTE data type. |
| Transaction | IfxTransaction | Gets or sets the transaction in which the IfxCommand object executes. |

Table 7. IfxCommand public properties (continued)

| Property | Type | Description |
|------------------|-----------------------------|--|
| 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)

Set the RowFetchCount property immediately after creation of the IfxCommand object or before invoking methods such as ExecuteReader that return an instance of DataReader. You can also set the RowFetchCount value by using the constructor. For example to set the RowFetchCount to 1, complete either of the following steps:

- `IfxCommand SelCmd = new IfxCommand(SQLcom, conn1); SelCmd.RowFetchCount = 1`
- `int rowFetchCount = 1; IfxCommand SelCmd = IfxCommand(cmdText, connection, rowFetchCount)`

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.Data.CommandBehavior 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 result set 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");
```

```
IfxConnection conn = new IfxConnection
("Database=stores7;Server=ol_sigaram_11;UID=informix;Password=ids4data");
IfxCommand cmd = new IfxCommand("select col1 from tbltest", conn);
conn.Open();
IfxDataReader myReader = cmd.ExecuteReader();
while (myReader.Read())
{
    Console.WriteLine("{0}", myReader.GetString(0));
}
myReader.Close();
myReader.Dispose();
cmd.Dispose();
conn.Close();
conn.Dispose();
```

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 the HCL 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 [Reconcile DataSet changes with the database on page 8](#).

Create 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 8. IfxCommandBuilder public properties

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

Create an IfxConnection

Some methods of other objects create `IfxConnection` objects implicitly. To create an `IfxConnection` object 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`.

Related information

[IfxProviderFactory class on page 82](#)

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 9. IfxConnection public 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 on page 38 for more information. |

Table 9. IfxConnection public properties (continued)

| Property | Type | Description |
|---------------------------|------------------------------|--|
| 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 HCL 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 41 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 of the ConnectionString property is an empty string. The Server attribute is mandatory in all situations.

The minimum required connection string attributes that must be set for non-DSN connections are Server, Protocol, Service, and Host name. If any of those are missing, the server ignores the remaining attributes and next checks the environmental variables, followed by the values in specified in setnet32.

The following table shows the connection string attributes.

Table 10. 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 Informix® .NET Provider does not automatically recognize when these commands are issued. Using these statements without proper management can lead to unexpected results. | "" (Empty string) |
| 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. | "" (Empty string) |

Table 10. Connection string attributes (continued)

| Attribute | Description | Default value |
|-------------------------------------|--|-------------------|
| Packet Size, Fetch Buffer Size, FBS | The size in bytes of the buffers used to send data to or from the server. Maximum value is 2147483648 (2GB). | 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. | "" (Empty string) |
| 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 CreateConnection() and the database server. | "" (Empty string) |
| Server | The name or alias of the instance of the Informix® server to which to connect. Required. | "" (Empty string) |
| Service | The service name or port number through which the server is listening for connection requests. | "" (Empty string) |
| 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. | "" (Empty string) |

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 `CreateConnection()` `ConnectionString` is not identical to the 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 11. Results for the UserDefinedTypeFormat setting with IfxDataReader access methods

| User Defined Type Format | Operation | Result |
|--------------------------|-----------|----------------------|
| string GetBytes() | | InvalidCastException |
| string GetString() | | Returns a string |
| bytes GetBytes() | | Returns bytes |
| bytes GetString() | | Returns the binary |

Table 11. Results for the UserDefinedTypeFormat setting with IfxDataReader access methods (continued)

| UserDefinedTypeFormat Setting | Operation | Result value as a hexadecimal decimal 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 IfxCommand 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 12. IfxConnection public 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. |

IfxConnection example

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"
);
IfxCommand cmd = new IfxCommand("select fname from customer", conn);
conn.Open();
IfxDataReader myReader = cmd.ExecuteReader();
while (myReader.Read())
{
    Console.WriteLine("{0}", myReader.GetString(0));
}
myReader.Close();
myReader.Dispose();
cmd.Dispose();
conn.Close();
conn.Dispose();

```

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

Create 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 13. IfxConnectionStringBuilder public 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.

Create 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 14. IfxDataAdapter public 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. |

Table 14. IfxDataAdapter public properties (continued)

| Property | Description |
|--------------------------------|---|
| 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. |
| ReturnProviderSpecificitytypes | 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 15. IfxDataReader public properties

| Property | Type | Description |
|----------|--------------|-------------------|
| Depth | System.Int32 | Always returns 0. |

Table 15. IfxDataReader public properties (continued)

| Property | Type | Description |
|-------------------|----------------|--|
| 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 NotSupportedException 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 NotSupportedException 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();
reader.Dispose(); //To prevent high memory usage, Dispose() method is called.
//Close Connection
```

IfxDataSourceEnumerator class

`IfxDataSourceEnumerator` allows .NET applications to read HCL Informix® SQLHOST entries programmatically. `Setnet32` is utility that provides a GUI interface to SQLHOST entries.

Create 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 16. IfxDataSourceEnumerator public properties

| Property | Description |
|----------|--------------------------|
| Instance | Retrieves an enumerator. |

IfxDataSourceEnumerator public methods

IfxDataSourceEnumerator.GetDataSources

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

Table 17. Columns of System.Data.DataTable

| Column name | Ordinal position | Description |
|-------------------------|------------------|--|
| IfxDataSourceEnumerator | 0 | Name of the server instance. |
| ata | | Multiple instances can exist on a single server. |

Table 17. Columns of System.Data.DataTable (continued)

| Column name | Ordinal position | Description |
|--------------|------------------|--|
| HostComputer | 1 | Host entry in SQLHOS table. |
| User | 2 | User name used to connect to the database. |
| Password | 3 | Password option |

Table 17. Columns of System.Data.DataTable (continued)

| Column name | Ordinal position | Description |
|-------------|------------------|---|
| Password | 4 | as stored through the host. |
| Protocol | 5 | Empty string. Protocol column used for communication. |

Table 17. Columns of System.Data.DataTable (continued)

| Column name | Ordinal position | Description |
|-------------|------------------|-----------------------------|
| 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 18. 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 92](#).

Create 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` 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.

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 19. 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. |

Table 19. IfxDateTime public properties (continued)

| Property | Type | Access notes | Description |
|---------------|--------------|------------------|---|
| 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. |
| 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.

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 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.

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 ifxDT2)
```

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

-1

ifxDT1 is earlier than *ifxDT2*

0

ifxDT1 and *ifxDT2* are the same time

1

ifxDT1 is later than *ifxDT2*

Objects in the 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 18: Default values for time units in IfxDateTime objects on page 58](#) 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` of the `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.GreaterThanOrEqualTo

```
static System.Boolean IfxDateTime.GreaterThanOrEqualTo(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.LessThanOrEqualTo

```
static System.Boolean IfxDateTime.LessThanOrEqualTo(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 about the syntax, refer to the description of the **DBTIME** environment variable in the *HCL® 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 *HCL® 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 HCL Informix® databases 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.

Create 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 20. 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`.

-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`

Objects in the 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)
```

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 ");
```

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

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 <code>FractionDigits</code> | Result of <code>IfxDecimal.Round(IfxDec1,FractionDigits)</code> |
|--------------------------------------|---|
| -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 *HCL® Informix® Enterprise Replication Guide*.

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 <i>FractionDigits</i> | Result of <code>IfxDecimal.Truncate(<i>IfxDec1</i>,<i>FractionDigits</i>)</code> |
|--------------------------------|--|
| -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 HCL Informix® database.

IfxError public properties

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

Table 21. IfxError public properties

| Property | Description |
|-------------|---|
| Message | Gets the description of the error. |
| NativeError | Gets the error code returned from the HCL 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 22. IfxErrorCollection public properties

| Property | Description |
|----------|--|
| Count | Returns the number of <code>IfxError</code> 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 HCL Informix® database.

IfxException public properties

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

Table 23. IfxException public properties

| Property | Description |
|----------------|--|
| Errors | Gets the collection of <code>IfxError</code> objects as an <code>IfxErrorCollection</code> 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 on page 92](#). For details about this enumeration see [IfxTimeUnit enumeration on page 92](#).

Create 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*.

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 24. IfxMonthSpan public 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. |
| 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*.

-1

ms1 is less than *ms2*

0

ms1 and *ms2* have the same value

1

ms1 is greater than *ms2*

Objects in the 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 *HCL@ 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 *HCL® 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.

Create 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 94](#) 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 25. IfxParameter public 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 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 on page 79). |
| 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 on page 79). |
| 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.

Create 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 26. IfxParameterCollection public 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 [Create an IfxParameterCollection on page 80](#) 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 HCL Informix® 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

This table 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 this table and assigned to the `IfxBlob.Flags` or `IfxClob.Flags` property.

| Member | Meaning |
|--------------------|---|
| DontKeepAccessTime | <p>If read this means that the smart large object does not keep track of the last time it was accessed.</p> <p>If written it tells the database server not to track the last access time for this smart large object.</p> <p>This flag overrides KeepAccessTime if both are given.</p> |
| KeepAccessTime | <p>If read this means that the smart large object keeps track of the last time it was accessed.</p> <p>If written it tells the database server to track the last access time for this smart large object.</p> <p>Use of the access time tracking feature causes significant extra work for the database server. Consider carefully before turning it on.</p> |
| Log | <p>If read this means that the database server logs changes to this smart large object in the system log file.</p> <p>If written it tells the database server to log changes to this smart large object in the system log file.</p> <p>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.</p> |
| NoLog | <p>If read this means that changes to this smart large object are not logged in the system log file.</p> <p>If written it tells the database server not to log changes to this smart large object in the system log file.</p> <p>This flag overrides Log if both are given.</p> |

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 uncommitted 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 92](#).

Create 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.

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`.

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 27. IfxTimeSpan public properties

| Property | Type | Access notes | Description |
|-------------|--------------|--------------|---|
| Days | System.Int64 | read-only | The number of full days in the <code>IfxTimeSpan</code> . If the <code>IfxTimeSpan</code> is negative then this value will be negative. |
| EndTimeUnit | IfxTimeUnit | read-only | The smallest unit included in the <code>IfxTimeSpan</code> . |

Table 27. IfxTimeSpan public properties (continued)

| Property | Type | Access notes | Description |
|---------------|----------------|---------------------|--|
| | | |  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. |
| Ticks | System.Decimal | read-only | <p>The total number of ticks in the length of time represented by the IfxTimeSpan. There are 10 000 000 ticks in one second.</p> <p>If the IfxTimeSpan is negative then this value will be negative.</p> |
| 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*.

-1

ts1 is less than *ts2*

0

ts1 and *ts2* have the same value

1

ts1 is greater than *ts2*

Objects in the 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.GreaterThanOrEqualTo

```
static System.Boolean IfxTimeSpan.GreaterThanOrEqualTo(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.LessThanOrEqualTo

```
System.Boolean IfxTimeSpan.LessThanOrEqualTo(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 about the syntax, refer to the description of the **DBTIME** environment variable in the *HCL® Informix® Enterprise Replication Guide*.

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 *HCL® 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 28. 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 29. Fractional members of IfxTimeUnit

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

The HCL 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 30. IfxTransaction public properties

| Property | Description |
|-----------------|--|
| Connection | The IfxTransaction object to associate with the transaction. |
| IsolationLevel | The isolation level for this transaction. With HCL 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 runs a command for which you want to control the transaction, you must assign the active transaction that is used in a connection to the Transaction property of the `IfxCommand` object, as shown in the example that follows. 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 HCL Informix® database servers. The following table 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 *HCL® Informix® Guide to SQL: Reference*.

Table 31. 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 |
| Date | Date | DateTime |
| DateTime | DateTime | DateTime |
| Decimal | Decimal | Decimal |
| Float | Double | Double |

Table 31. IfxType enumeration (continued)

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

Sample programs

Demonstration programs

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

The %INFORMIXDIR%\demo\dotnetdemo\QuickStart folder contains sample programs for C#.NET and VB.NET. The %INFORMIXDIR%\demo\dotnetdemo\Samples folder contains sample programs for C# Windows™ Forms.

Informix® .NET Provider 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 examples 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 HCL Informix® database servers. The sample databases used are **stores_demo** and **superstores_demo**. Instructions on how to create these databases are in the *HCL® Informix® DB-Access User's Guide*.

Retrieve a single value

You can 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 30](#).

```
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 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
}
```

Retrieve multiple rows

You can use an `IfxDataReader` object for simple access to data when you do not have to write and do not have to move backward.

The following example connects to the **stores_demo** database and uses an `IfxDataReader` object to retrieve all of the first names from the customer table. For more information about the `IfxDataReader` class, see [IfxDataReader class on page 50](#).

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

Execute SQL that does not return data and using a transaction

You can use the `IfxCommand.ExecuteNonQuery` method to execute SQL statements that do not return any data

Types of SQL statements that do not return data include:

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

The example in this topic 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 about the `IfxCommand` class, see [IfxCommand class on page 30](#). For the details about the `IfxTransaction` class see [IfxTransaction class on page 93](#)

```

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
}

```

Retrieve data into a DataSet

You can use an `IfxDaDataAdapter` object to retrieve database information into a `System.Data.DataSet` object for further processing.

The following 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 about the `IfxDaDataAdapter` object, see [IfxDaDataAdapter class on page 47](#). For more information about data sets 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();

IfxDaDataAdapter da = new IfxDaDataAdapter(

```

```

        "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
}
}

```

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 HCL Informix® database.

The `IfxCommandBuilder` object facilitates easy reconciliation of changes made in your data set with the database.

For more information about the `IfxCommandBuilder` class, see [IfxCommandBuilder class on page 35](#). For more information about reconciling changes in the database, see [Reconcile DataSet changes with the database on page 8](#).

The following example shows how to use the `IfxCommandBuilder` object.

```

// 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
IfxDaataAdapter allDataAdapter = new IfxDaataAdapter();
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();

```

Call a stored procedure

You can use an `IfxCommand` object to call a stored procedure. You must set the `IfxCommand` object `CommandType` property to `StoredProcedure`.

The example in this topic 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 30](#). For more information about calling stored procedures, see [Call stored procedures on page 10](#).

```

// 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();

```

Distributed transactions

The example in this topic 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
}
}

```

Write CLOBs to files

The example in this topic 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 25](#).

```

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.IfSmartLOBOpenMode.ReadOnly);
        c.ToFile(
            "C:\\tmp\\" + num.ToString() + ".txt",
            System.IO.FileMode.Create,
            IfSmartLOBFileLocation.Client
        );
    }
}
dr.Close();

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

Index

Special Characters

- ? symbol, parameters 9
- .NET base classes 18
- .NET DbType types 16, 94
- .NET Framework types 94
- .NET interfaces 17

A

- Abs method
 - of `IfxDecimal` structure 66
- AcceptChangesDuringFill property
 - of `IfxDataAdapter` class 47
- AcceptChangesDuringUpdate property
 - of `IfxDataAdapter` class 47
- Add method
 - of `IfxConnectionStringBuilder` class 46
 - of `IfxDateTime` structure 60
 - of `IfxDecimal` structure 66
 - of `IfxMonthSpan` structure 74
 - of `IfxParameterCollection` class 81
 - of `IfxTimeSpan` structure 89
- Add value 47
- AddDays method
 - of `IfxDateTime` structure 60
- AddMilliseconds method
 - of `IfxDateTime` structure 60
- AddMinutes method
 - of `IfxDateTime` structure 60
- AddMonths method
 - of `IfxDateTime` structure 60
- AddSeconds method
 - of `IfxDateTime` structure 60
- AddWithKey value 47
- AddYears method
 - of `IfxDateTime` structure 60
- ADO.NET 5
- AnsiString type 16
- Append member
 - of `IfxSmartLOBOpenMode` enumeration 85
- AppendKeyValuePair method
 - of `IfxConnectionStringBuilder` class 46
- ArgumentException exception 38
- ASP.NET 3
- Assembly, strong-named 4
- Attributes, for connecting 7
- Automatic INSERT, DELETE and UPDATE 8, 35

B

- Begin member
 - of `IfxSmartLOBWhence` enumeration 86
- BeginTransaction method
 - of `IfxConnection` class 44
- Bigint member
 - of `IfxType` enumeration 94
- BIGINT type 15
- BigSerial member
 - of `IfxType` enumeration 94
- BIGSERIAL type 15
- Blob member
 - of `IfxType` enumeration 94
- BLOB type 15
- Boolean member
 - of `IfxType` enumeration 94
- BOOLEAN type 15
- Both value 31
- Buffer member
 - of `IfxSmartLOBOpenMode` enumeration 85
- Byte member

- of `IfxType` enumeration 94
- BYTE type 15

C

- Calling a stored procedure
 - example of 100
- Cancel method
 - of `IfxCommand` class 32
- casting, example of 96
- cdotnet.sql script 5
- Ceiling method
 - of `IfxDecimal` structure 66
- ChangeDatabase method
 - of `IfxConnection` class 44
- Char member
 - of `IfxType` enumeration 94
- CHAR type 15
- CHAR(1) type 15
- Char1 member
 - of `IfxType` enumeration 94
- Classes, unsupported 17, 18
- Clear method
 - of `IfxConnectionStringBuilder` class 46
 - of `IfxParameterCollection` class 81
- Client Locale connection string attribute 38
- Client member
 - of `IfxSmartLOBFileLocation` enumeration 84
- Client SDK 3
- ClientLocale property
 - of `IfxConnection` class 37
- Clob member
 - of `IfxType` enumeration 94
- CLOB type 15
- Clone method
 - of `IfxDecimal` structure 66
- Close method
 - of `IfxBlob` class 23
 - of `IfxClob` class 28
 - of `IfxConnection` class 44
 - of `IfxDataReader` class 51
- Close property
 - of `IfxConnection` use of 7
- CloseConnection value 32
- Closing connections 7
- Codes, errors 71
- COLLECTION type 15
- CommandText property 10
 - of `IfxCommand` 31
- CommandTimeout property 8
 - of `IfxCommand` 31
- CommandType property 10
 - of `IfxCommand` 31
 - of `IfxCommand` class example of use 100
- Commit method
 - of `IfxTransaction` class 93
- Compare method
 - of `IfxDateTime` structure 60
 - of `IfxDecimal` structure 66
 - of `IfxMonthSpan` structure 74
 - of `IfxTimeSpan` structure 89
- CompareTo method
 - of `IfxDateTime` structure 60
 - of `IfxDecimal` structure 66
 - of `IfxMonthSpan` structure 74
 - of `IfxTimeSpan` structure 89
- ConflictOption property
 - of `IfxCommandBuilder` class 35
- Connecting to databases 7
- Connection Lifetime connection string attribute 38
- Connection pool 9, 38, 38
- Connection property 8
 - of `IfxCommand` 31
 - of `IfxTransaction` class 93
- Connection string 7
 - attributes of 38
 - defining visually 7
 - example of 45
- ConnectionString property
 - attributes of 38
 - of `IfxConnection` use of 7
 - of `IfxConnection` class 37
 - of `IfxConnectionStringBuilder` class 45
- ConnectionString property of `IfxConnection` defining visually 7
- ConnectionTimeout property
 - of `IfxConnection` class 37
- Constructors
 - of `IfxBlob` class 21
 - of `IfxClob` class 26
 - of `IfxCommand` class 31
 - of `IfxCommandBuilder` class 35
 - of `IfxConnection` class 37
 - of `IfxDataAdapter` class 47
 - of `IfxDateTime` structure 58
 - of `IfxDecimal` structure 66
 - of `IfxMonthSpan` structures 73
 - of `IfxParameter` class 78
 - of `IfxTimeSpan` structure 87
- Contains method
 - of `IfxParameterCollection` class 81
- ContainsKey method
 - of `IfxConnectionStringBuilder` class 46
- CopyTo method
 - of `IfxParameterCollection` class 81
- Count property
 - of `IfxConnectionStringBuilder` class 45
 - of `IfxErrorCollection` class 71
 - of `IfxParameterCollection` class 80
- CREATE DATABASE... statements 38
- CreateCommand method
 - of `IfxConnection` class 44
 - of `IfxProviderFactory` class 82
- CreateCommandBuilder method
 - of `IfxProviderFactory` class 82
- CreateConnection method
 - of `IfxProviderFactory` class 82
- CreateConnectionStringBuilder method
 - of `IfxProviderFactory` class 82
- CreateDataAdapter method
 - of `IfxProviderFactory` class 82
- CreateDataSourceEnumerator method
 - of `IfxProviderFactory` class 82
- CreateParameter method
 - of `IfxCommand` class 32
 - of `IfxProviderFactory` class 82
- CreatePermission method
 - of `IfxProviderFactory` class 82
- Creating
 - `IfxCommand` class 30
 - `IfxCommandBuilder` class 35
 - `IfxConnection` class 37

- IfxConnectionStringBuilder class 45
- IfxDataAdapter class 47
- IfxDataSourceEnumerator class 53
- IfxDateTime structure 58
- IfxDecimal structure 65
- IfxMonthSpan structure 72
- IfxParameter class 78
- IfxParameterCollection class 80
- IfxTimeSpan structure 86
- Current member
 - of IfxSmartLOBWhence enumeration 86

D

- Data source 5
- Data transfer
 - error checking 14
- DataAdapter property
 - of IfxCommandBuilder class 35
- Database connection string attribute 38
- Database Locale connection string attribute 38
- Database property
 - of IfxConnection class 37
- DATABASE... statements 38
- DatabaseLocale property
 - of IfxConnection class 37
- Databases
 - connecting to 7
- DataColumn interface 5
- DataReader object 5
- DataRelation interface 5
- DataSet object 5, 15
 - example of use 98
- DataTable interface 5
- Date member
 - of IfxType enumeration 94
- Date property
 - of IfxDateTime structure 59
- DATE type 15
- DATETIME data type 57
- DateTime member
 - of IfxType enumeration 94
- DATETIME type 15
- Day member
 - of IfxTimeUnit enumeration 92
- Day property
 - of IfxDateTime structure 59
- Days property
 - of IfxTimeSpan structure 87
- DB connection string attribute 38
- DB_LOCALE connection string attribute 38
- DbProviderFactory class 82
- DbType property
 - of IfxParameter class 79
- Decimal member
 - of IfxType enumeration 94
- DECIMAL type 15
- Default value 32
- DELETE statements 35
- DELETE, automatic 8
- DeleteCommand property 8
 - of IfxDataAdapter class 47
- DELIMITED connections string attribute 38
- DELIMITED environment variable 7
 - default setting of 7
- Delimiters 8, 15, 35
- Demonstration programs
 - Informix
 - .NET Provider
 - 95
- Depth property

- of IfxDataReader class 50
- DeriveParameters method 10
 - of IfxCommandBuilder class 36
- Direction property
 - of IfxParameter class 79
- DirtyRead member
 - of IfxSmartLOBOpenMode enumeration 85
- Dispose method 8
- Disposed event
 - of IfxConnection class 44
- Distributed transactions 38, 100
 - example of use 100
- Divide method
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- DontKeepAccessTime member
 - of IfxSmartLOBCreateTimeFlags enumeration 83
- dotnetdemo folder 95
- DOUBLE type 15
- Double-quote characters 15
- Duration method
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- Dynamic SQL 9

E

- E property
 - of IfxDecimal structure 66
- Efficient SQL statements 8
- End member
 - of IfxSmartLOBWhence enumeration 86
- end time unit 57, 72
- EndTimeUnit property
 - of IfxDateTime structure 59
 - of IfxMonthSpan structure 73
 - of IfxTimeSpan structure 87
- Enlist attribute 10
- Enlist connection string attribute 38
- Environment variables
 - IFX_LOB_XFERSIZE 14
- Equals method
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- EquivalentTo method
 - of IfxConnectionStringBuilder class 46
- Error handling
 - checking during data transfer 14
- Error messages 14, 71
- Error value 47, 47
- Errors property
 - of IfxException class 72
- EstimatedSize property
 - of IfxBlob 21
 - of IfxClob 26
- Events
 - of IfxConnection class 44
- Example code
 - IfxCommand class 34
- Examples
 - calling a stored procedure 100
 - casting a data type to a new type 96
 - Informix
 - .NET Provider
 - 95, 95
 - Inserting rows 97
 - of a connection string 45
 - Reconciling changes in a DataSet with the database 99
 - retrieving a single value 96

- Retrieving data into a DataSet 98
 - retrieving multiple rows 96
 - use of ExecuteNonQuery 97
 - use of ExecuteScalar method 96
 - use of IfxCommand.CommandType property 100
 - use of IfxCommandBuilder 99
 - use of IfxDataAdapter 98
 - use of IfxDataReader class 96
 - use of IfxParameter class 79
 - use of IfxTransaction class 94
 - use of local transaction 97
 - use of System.DataSet 98
 - using distributed transactions 100

Exceptions 14

- Exclusive connection string attribute 38
- Exclusive member
 - of IfxSmartLOBLockMode enumeration 85
- ExecuteNonQuery method
 - example of use 97
 - of IfxCommand class 32
- ExecuteNonQuery() method 97
- ExecuteReader method
 - of IfxCommand class 32
- ExecuteScalar method
 - of IfxCommand class 32
- ExecuteScalar method, example of use 96
- ExtentSize property
 - of IfxBlob 21
 - of IfxClob 26

F

- FBS connection string attribute 38
- Fetch Buffer Size connection string attribute 38
- FetchBufferSize property
 - of IfxConnection class 37
- FieldCount property
 - of IfxDataReader class 50
- Fill method
 - of IfxDataAdapter class 49
- FillLoadOption property
 - of IfxDataAdapter class 47
- FillSchema method
 - of IfxDataAdapter class 49
- FirstReturnedRecord value 31
- Flags property
 - of IfxBlob 21
 - of IfxClob 26
- Float member
 - of IfxType enumeration 94
- FLOAT type 15
- Foreign keys 8
- Forward-only cursor 5, 38, 50, 96
- Fraction member
 - of IfxTimeUnit enumeration 92
- Fraction1 member
 - of IfxTimeUnit enumeration 92
- Fraction2 member
 - of IfxTimeUnit enumeration 92
- Fraction3 member
 - of IfxTimeUnit enumeration 92
- Fraction4 member
 - of IfxTimeUnit enumeration 92
- Fraction5 member
 - of IfxTimeUnit enumeration 92
- FromFile method
 - of IfxBlob class 23
 - of IfxClob class 28
- FullTrust permission 9

G

- GAC 4
- gacutil utility 4
- GetActiveConnectionsCount property
 - of IfxConnection class 37
- GetBoolean method
 - of IfxDataReader class 51
- GetByte method
 - of IfxDataReader class 51
- GetBytes method
 - of IfxDataReader class 51
- GetChar method
 - of IfxDataReader class 51
- GetChars method
 - of IfxDataReader class 51
- GetData method
 - of IfxDataReader class 51
- GetDataSources method
 - of IfxDataSourceEnumerator class 54
- GetDataTypeName method
 - of IfxDataReader class 51
- GetDateTime method
 - of IfxDataReader class 51
- GetDecimal method
 - of IfxDataReader class 51
- GetDeleteCommand method 8
 - of IfxCommandBuilder class 36
- GetDouble method
 - of IfxDataReader class 51
- GetEnumerator method
 - of IfxErrorCollection class 71
 - of IfxParameterCollection class 81
- GetFieldType method
 - of IfxDataReader class 51
- GetFillParameters method
 - of IfxDataAdapter class 49
- GetFloat method
 - of IfxDataReader class 51
- GetGuid method
 - of IfxDataReader class 51
- GetHashCode method
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- GetIdleConnectionsCount property
 - of IfxConnection class 37
- GetIxBlob method
 - of IfxConnection class 44
- GetIxClob method
 - of IfxConnection class 44
- GetInsertCommand method 8
 - of IfxCommandBuilder class 36
- GetInt16 method
 - of IfxDataReader class 51
- GetInt32 method
 - of IfxDataReader class 51
- GetInt64 method
 - of IfxDataReader class 51
- GetLocator method
 - of IfxBlob class 23
 - of IfxClob class 28
- GetName method
 - of IfxDataReader class 51
- GetOrdinal method
 - of IfxDataReader class 51
- GetSchemaTable method
 - of IfxDataReader class 51
- GetString method
 - of IfxDataReader class 51
- GetString() method 15
- GetTimeSpan method
 - of IfxDataReader class 51

- GetUpdateCommand method 8
- GetValue method
 - of IfxDataReader class 51
- GetValues method
 - of IfxDataReader class 51
- Global Assembly Cache (GAC) 4
- GreaterThan method
 - of IfxDateTime structure 60, 60
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- GreaterThanOrEqual method
 - of IfxDateTime structure 60
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- GUID type 16

H

- HCL
- Informix
 - .NET Provider
 - installing 4
 - overview 5
- HCL
 - Informix
 - Client SDK
 - 3
- HCL
 - Informix
 - ODBC Driver
 - 3
- HCL
 - Informix
 - OLE DB Provider
 - 3
- HelpLink property
 - of IfxException class 72
- Host connection string attribute 38
- Hostnames
 - using in sqlhosts 4
- Hour member
 - of IfxTimeUnit enumeration 92
- Hour property
 - of IfxDateTime structure 59
- Hours property
 - of IfxTimeSpan structure 87

I

- IBM.Data
 - Informix
 - 82
- IBM.Data
 - Informix
 - namespace
 - 6
- IDSSECURITYLABEL type 15
- IFX_LOB_XFERSIZE
 - environment variable 14
- IxBlob class 20, 23
 - Close method 23
 - constructors 21
 - creating 20
 - EstimatedSize property 21
 - ExtentSize property 21
 - Flags property 21
 - FromFile method 23
 - GetLocator method 23
 - IxBlob class
 - SBSpace property 21
 - Size property 21
 - internal cursor 20

- IsNull property 21
- IsOpen property 21
- LastAccessTime property 21
- LastChangeTime property 21
- LastModificationTime property 21
- Lock method 23
- MaxBytes property 21
- Null property 21
- Open method 23
- Position property 21
- properties 21
- Read method 23
- ReferenceCount property 21
- Release method 23
- SBSpace property
 - of IxBlob 21
- Seek method 23
- Size property
 - of IxBlob 21
- ToFile method 23
- Truncate method 23
- Unlock method 23
- Write method 23
- IxBlob class methods
 - Methods
 - of IxBlob class 23
- IxClob 25, 28
 - constructors 26
- IxClob class
 - Close method 28
 - creating 26
 - EstimatedSize property 26
 - ExtentSize property 26
 - Flags property 26
 - FromFile method 28
 - GetLocator method 28
 - IxClob class
 - SBSpace property 26
 - Size property 26
 - internal cursor of 25
 - IsNull property 26
 - IsOpen property 26
 - LastAccessTime property 26
 - LastChangeTime property 26
 - LastModificationTime property 26
 - Lock method 28
 - MaxBytes property 26
 - methods of 28
 - Null property 26
 - Open method 28
 - Position property 26
 - Read method 28
 - ReferenceCount property 26
 - SBSpace property
 - of IxClob 26
 - Seek method 28
 - Size property
 - of IxClob 26
 - Truncate method 28
 - Unlock method 28
 - Write method 28
- IxClob class properties
 - Properties
 - of IxClob class 26
- IfxCommand class 30
 - Cancel method 32
 - CommandText property 31
 - CommandTimeout property 31
 - CommandType property 31
 - Connection property 31
 - constructors 31

- CreateParameter method 32
- creating 30
- example code 34
- ExecuteNonQuery method 32
- ExecuteReader method 32
- ExecuteScalar method 32
- methods 32
- Parameters property 31
- Prepare method 32
- properties 31
- RowFetchCount property 31
- Transaction property 31
- UpdatedRowSource property 31
- IfxCommand.ExecuteNonQuery() method 97
- IfxCommandBuilder class 5, 8, 35
 - ConflictOption property 35
 - constructors 35
 - creating 35
 - DataAdapter property 35
 - DeriveParameters method 36
 - example of use 36, 99
 - GetDeleteCommand method 36
 - GetInsertCommand method 36
 - methods of 36
 - properties of 35
 - QuotePrefix property 35
 - QuoteSuffix property 35
 - RefreshSchema method 36
- IfxConnection class 37
 - BeginTransaction method 44
 - ChangeDatabase method 44
 - ClientLocale property 37
 - Close method 44
 - connecting to a database 7
 - ConnectionString property 37
 - ConnectionTimeout property 37
 - CreateCommand method 44
 - creating 37
 - creating visually 7
 - Database property 37
 - DatabaseLocale property 37
 - Disposed event 44
 - events of 44
 - FetchBufferSize property 37
 - GetActiveConnectionsCount property 37
 - GetIdleConnectionsCount property 37
 - GetIfxBlob method 44
 - GetIfxClob method 44
 - InfoMessage event 44
 - methods 44
 - Open method 7, 44
 - PacketSize property 37
 - properties 37
 - ServerVersion property 37
 - State property 37
 - StateChange event 44
 - UserDefinedTypeFormat property 37
- IfxConnection class constructors 37
- IfxConnection object
 - Close property
 - use of 7
 - ConnectionString property
 - defining visually 7
 - use of 7
- IfxConnectionStringBuilder class 45
 - Add method 46
 - AppendKeyValuePair method 46
 - Clear method 46
 - ConnectionString property 45
 - ContainsKey method 46
 - Count property 45
 - creating 45
 - EquivalentTo method 46
 - IsFixedSize property 45
 - IsReadOnly property 45
 - Keys property 45
 - methods 46
 - properties 45
 - Remove method 46
 - ToString method 46
 - TryGetValue method 46
 - Values property 45
- IfxDataAdapter
 - examples of use 49
- IfxDataAdapter class 8, 15, 35, 47
 - AcceptChangesDuringFill property 47
 - AcceptChangesDuringUpdate property 47
 - constructors 47
 - creating 47
 - DeleteCommand property 47
 - example of use 98
 - Fill method 49
 - FillLoadOption property 47
 - FillSchema method 49
 - GetFillParameters method 49
 - InsertCommand property 47
 - methods 49
 - MissingMappingAction property 47
 - MissingSchemaAction property 47
 - properties 47
 - ReturnProviderSpecificTypes property 47
 - SelectCommand property 47
 - TableMappings property 47
 - Update method 49
 - UpdateBatchSize property 47
 - UpdateCommand property 47
- IfxDataReader class 50, 96
 - Close method 51
 - Depth property 50
 - example of use 53, 96
 - FieldCount property 50
 - GetBoolean method 51
 - GetByte method 51
 - GetBytes method 51
 - GetChar method 51
 - GetChars method 51
 - IsClosed property 50
 - methods 51
 - properties 50
 - RecordsAffected property 50
 - VisibleFieldCount property 50
- IfxDataReader.GetString() method 15
- IfxDataSourceEnumerator class 53
 - creating 53
 - GetDataSources method 54
 - Instance property 53
 - methods 54
 - properties 53
- IfxDateTime structure 57
 - Add method 60
 - AddDays method 60
 - AddMilliseconds method 60
 - AddMinutes method 60
 - AddMonths method 60
 - AddSeconds method 60
 - AddYears method 60
 - Compare method 60
 - CompareTo method 60
 - constructors 58
 - creating 58
 - Date property 59
 - Day property 59
 - default values 57
 - EndTimeUnit property 59
 - GreaterThan method 60, 60
 - GreaterThanOrEqual method 60
 - Hour property 59
 - LessThan method 60
 - limits of 57
 - MaxValue property 59
 - methods 60
 - Millisecond property 59
 - Minute property 59
 - MinValue property 59
 - Month property 59
 - NotEquals method 60
 - Now property 59
 - Null property 59
 - Parse method 60
 - properties 59
 - Second property 59
 - StartTimeUnit property 59
 - Ticks property 59
 - time units in 57
 - Today property 59
 - ToString method 60
 - Year property 59
- IfxDcimal structure 65
 - Abs method 66
 - Add method 66
 - Ceiling method 66
 - Clone method 66
 - Compare method 66
 - CompareTo method 66
 - constructors 66
 - creating 65
 - E property 66
 - Equals method 66
 - GetHashCode method 66
 - GreaterThan method 66
 - GreaterThanOrEqual method 66
 - IsFloating property 66
 - IsNull property 66
 - IsPositive property 66
 - LessThan method 66
 - LessThanOrEqual method 66
 - Max method 66
 - MaxPrecision property 66
 - MaxValue property 66
 - methods 66
 - Min method 66
 - MinusOne property 66
 - MinValue property 66
 - Modulo method 66
 - Multiply method 66
 - Negate method 66
 - NotEquals method 66
 - Null property 66
 - One property 66
 - Parse method 66
 - Pi property 66
 - properties 66
 - Remainder method 66
 - Round method 66
 - Subtract method 66
 - ToString method 66
 - Truncate method 66
 - example of use 66
 - Zero property 66
- IFXDOTNETTRACE environment variable 14
- IFXDOTNETTRACEFILE environment variable 14
- IfxError class 71

- Message property 71
- NativeError property 71
- properties 71
- SQLState property 71
- IfxErrorCollection class 71
 - Count property 71
 - GetEnumerator method 71
 - methods 71
- IfxErrorCollection class properties 71
- IfxException class 72
 - Errors property 72
 - HelpLink property 72
 - InnerException property 72
 - Message property 72
 - properties 72
 - StackTrace property 72
 - TargetSite property 72
- IfxMonthSpan structure 72
 - Add method 74
 - Compare method 74
 - CompareTo method 74
 - constructors 73
 - creating 72
 - Divide method 74
 - Duration method 74
 - EndTimeUnit property 73
 - Equals method 74
 - GetHashCode method 74
 - GreaterThan method 74
 - GreaterThanOrEqual method 74
 - IsNull property 73
 - LessThan method 74
 - LessThanOrEqual method 74
 - limits of 72
 - MaxValue property 73
 - MinValue property 73
 - Months property 73
 - Multiply method 74
 - Negate method 74
 - NotEquals method 74
 - Null property 73
 - Parse method 74
 - properties 73
 - StartTimeUnit property 73
 - ToString method 74
 - TotalMonths property 73
 - Years property 73
 - Zero property 73
- IfxMonthSpan structure methods 74
- IfxParameter class 77
 - constructors 78
 - creating 78
 - DbType property 79
 - Direction property 79
 - example of use 79
 - IfxType property 79
 - IsNullable property 79
 - ParameterName property 79
 - properties 79
 - SourceColumn property 79
 - SourceVersion property 79
 - Value property 79
- IfxParameter constructors 16
- IfxParameterCollection class 80
 - Add method 81
 - Clear method 81
 - Contains method 81
 - CopyTo method 81
 - Count property 80
 - creating 80
 - GetEnumerator method 81
- IndexOf method 81
- Insert method 81
- Item property 80
- methods 81
- properties 80
- Remove method 81
- RemoveAt method 81
- IfxProviderFactory class 82
 - CreateCommand method 82
 - CreateCommandBuilder method 82
 - CreateConnection method 82
 - CreateConnectionStringBuilder method 82
 - CreateDataAdapter method 82
 - CreateDataSourceEnumerator method 82
 - CreateParameter method 82
 - CreatePermission method 82
 - methods 82
- IfxSmartLOBCreateTimeFlags enumeration 83
 - DontKeepAccessTime member 83
 - KeepAccessTime member 83
 - Log member 83
 - NoLog member 83
- IfxSmartLOBFileLocation enumeration 84
 - Client member 84
 - Server member 84
- IfxSmartLOBLockMode enumeration 85
 - Exclusive member 85
 - Shared member 85
- IfxSmartLOBOpenMode enumeration 85
 - Append member 85
 - Buffer member 85
 - DirtyRead member 85
 - LockAll member 85
 - LockRange member 85
 - NoBuffer member 85
 - ReadOnly member 85
 - ReadWrite member 85
 - WriteOnly member 85
- IfxSmartLOBWhence enumeration 86
 - Begin member 86
 - Current member 86
 - End member 86
- IfxTimeSpan structure 86
 - Add method 89
 - Compare method 89
 - CompareTo method 89
 - constructors 87
 - creating 86
 - Days property 87
 - Divide method 89
 - Duration method 89
 - EndTimeUnit property 87
 - Equals method 89
 - GetHashCode method 89
 - GreaterThan method 89
 - GreaterThanOrEqual method 89
 - Hours property 87
 - IsNull property 87
 - LessThan method 89
 - LessThanOrEqual method 89
 - MaxScale property 87
 - MaxValue property 87
 - methods 89
 - Milliseconds property 87
 - Minutes property 87
 - MinValue property 87
 - Negate method 89
 - NotEquals method 89
 - Null property 87
 - Parse method 89
 - properties 87
- Seconds property 87
- StartTimeUnit property 87
- Ticks property 87
- ToString methods 89
- Zero property 87
- IfxTimeUnit enumeration 92
 - Day member 92
 - Fraction member 92
 - Fraction1 member 92
 - Fraction2 member 92
 - Fraction3 member 92
 - Fraction4 member 92
 - Fraction5 member 92
 - Hour member 92
 - Minute member 92
 - Month member 92
 - Second member 92
 - Year member 92
- IfxTransaction class 93
 - Commit method 93
 - Connection property 93
 - example of use 94
 - IsolationLevel property 93
 - methods 93
 - properties 93
 - Rollback method 93
- IfxType argument 16
- IfxType enumeration 94
 - Bigint member 94
 - BigSerial member 94
 - Blob member 94
 - Boolean member 94
 - Byte member 94
 - Char member 94
 - Char1 member 94
 - Clob member 94
 - Date member 94
 - DateTime member 94
 - Decimal member 94
 - Float member 94
 - Int8 member 94
 - Integer member 94
 - IntervalDayFraction member 94
 - IntervalYearMonth member 94
 - List member 94
 - LVarChar member 94
 - Money member 94
 - MultiSet member 94
 - NChar member 94
 - NVarChar member 94
 - Row member 94
 - Serial member 94
 - Serial8 member 94
 - Set member 94
 - SmallFloat member 94
 - SmallInt member 94
 - SmartLOBLocator member 94
 - Text member 94
 - VarChar member 94
- IfxType property
 - of IfxParameter class 79
- Ignore value 47, 47
- Importing a namespace 6
- IN parameters
 - used during execution of SPL 10
- IndexOf method
 - of IfxParameterCollection class 81
- InfoMessage event
 - of IfxConnection class 44
- Informix
 - .NET Provider

- namespace of 6
- Informix
 - Client SDK
 - 3
 - data types
 - 79
 - ODBC Driver
 - 3
 - OLE DB Provider
 - 3
 - types
 - 94
- InnerException property
 - of IfxException class 72
- Input parameters 79
- Insert method
 - of IfxParameterCollection class 81
- INSERT statements 35
- INSERT, automatic 8
- InsertCommand property 8
 - of IfxDataAdapter class 47
- Inserting rows, example of 97
- Installing
 - HCL
 - .NET Provider
 - 4
- Instance property
 - of IfxDataSourceEnumerator class 53
- Int32 type 15
- Int64 type 15
- Int64[] type 15
- Int8 member
 - of IfxType enumeration 94
- INT8 type 15
- Integer member
 - of IfxType enumeration 94
- INTEGER type 15
- Internet protocol version 6 (IPv6) 4
- INTERVAL type 15
- IntervalDayFraction member
 - of IfxType enumeration 94
- IntervalYearMonth member
 - of IfxType enumeration 94
- IP address 38
- IP addresses
 - IPv6 format in sqlhosts 4
- IPv6. 4
- IsClosed property
 - of IfxDataReader class 50
- IsDBNull method
 - of IfxDataReader class 51
- IsFixedSize property
 - of IfxConnectionStringBuilder class 45
- IsFloating property
 - of IfxDecimal structure 66
- IsNull property
 - of IfxBlob 21
 - of IfxClob 26
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 73
 - of IfxTimeSpan structure 87
- IsNullable property
 - of IfxParameter class 79
- IsolationLevel property
 - of IfxTransaction class 93
- IsOpen property

- of IfxBlob 21
- of IfxClob 26
- IsPositive property
 - of IfxDecimal structure 66
- IsReadOnly property
 - of IfxConnectionStringBuilder class 45
- Item property
 - of IfxParameterCollection class 80
- J**
- JOIN operators 8
- K**
- KeepAccessTime member
 - of IfxSmartLOBCreateTimeFlags
 - enumeration 83
- KeyInfo value 32
- Keys property
 - of IfxConnectionStringBuilder class 45
- L**
- Language locale 38
- LastAccessTime property
 - of IfxBlob 21
 - of IfxClob 26
- LastChangeTime property
 - of IfxBlob 21
 - of IfxClob 26
- LastModificationTime property
 - of IfxBlob 21
 - of IfxClob 26
- LessThan method
 - of IfxDateTime structure 60
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- LessThanOrEqual method
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- List member
 - of IfxType enumeration 94
- LIST type 15
- Local transactions
 - example of use 97
- Locales 38, 38
- Lock method
 - of IfxBlob class 23
 - of IfxClob class 28
- LockAll member
 - of IfxSmartLOBOpenMode enumeration 85
- LockRange member
 - of IfxSmartLOBOpenMode enumeration 85
- Log member
 - of IfxSmartLOBCreateTimeFlags
 - enumeration 83
- LVarChar member
 - of IfxType enumeration 94
- LVARCHAR type 15
- M**
- Max method
 - of IfxDecimal structure 66
- Max Pool Size attribute 9
- Max Pool Size connection string attribute 38
- MAX ROWS parameter 5
- MaxBytes property
 - of IfxBlob 21
 - of IfxClob 26
- MaxPrecision property
 - of IfxDecimal structure 66
- MaxScale property

- of IfxTimeSpan structure 87
- MaxValue property
 - of IfxDateTime structure 59
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 73
 - of IfxTimeSpan structure 87
- Message property
 - of IfxError class 71
 - of IfxException class 72
- Messages 71
- Methods
 - GetData method 51
 - GetDataTypeName method 51
 - GetDateTime method 51
 - GetDecimal method 51
 - GetDouble method 51
 - GetFieldType method 51
 - GetFloat method 51
 - GetGuid method 51
 - GetInt16 method 51
 - GetInt32 method 51
 - GetInt64 method 51
 - GetName method 51
 - GetOrdinal method 51
 - GetSchemaTable method 51
 - GetString method 51
 - GetTimeSpan method 51
 - GetValue method 51
 - GetValues method 51
 - IsDBNull method 51
 - NextResult method 51
 - of IfxClob class 28
 - of IfxCommand class 32
 - of IfxCommandBuilder class 36
 - of IfxConnection class 44
 - of IfxConnectionStringBuilder class 46
 - of IfxDataAdapter class 49
 - of IfxDataReader class 51
 - of IfxDataSourceEnumerator class 54
 - of IfxDateTime structure 60
 - of IfxDecimal structure 66
 - of IfxErrorCollection class 71
 - of IfxMonthSpan structure 74
 - of IfxParameterCollection class 81
 - of IfxProviderFactory class 82
 - of IfxTimeSpan structure 89
 - of IfxTransaction class 93
 - Read method 51
- Methods, unsupported 17, 18
- Microsoft .NET Framework SDK 85
- Microsoft ODBC .NET 3
- Microsoft OLE DB .NET 3
- Microsoft public .NET base classes 18
- Microsoft public .NET interfaces 17
- Millisecond property
 - of IfxDateTime structure 59
- Milliseconds property
 - of IfxTimeSpan structure 87
- Min method
 - of IfxDecimal structure 66
- Min Pool Size attribute 9
- Min Pool Size connection string attribute 38
- MinusOne property
 - of IfxDecimal structure 66
- Minute member
 - of IfxTimeUnit enumeration 92
- Minute property
 - of IfxDateTime structure 59
- Minutes property
 - of IfxTimeSpan structure 87
- MinValue property

- of IfxDatetime structure 59
- of IfxDecimal structure 66
- of IfxMonthSpan structure 73
- of IfxTimeSpan structure 87
- MissingMappingAction property
 - of IfxDataAdapter class 47
- MissingSchemaAction property
 - of IfxDataAdapter class 47
- Modulo method
 - of IfxDecimal structure 66
- Money member
 - of IfxType enumeration 94
- MONEY type 15
- Month member
 - of IfxTimeUnit enumeration 92
- Month property
 - of IfxDatetime structure 59
- Months property
 - of IfxMonthSpan structure 73
- Multiply method
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
- MultiSet member
 - of IfxType enumeration 94
- MULTISET type 15

N

- Namespace
 - importing 6
 - of Informix .NET Provider 6
- namespaces, System.Transaction 5
- NativeError property
 - of IfxError class 71
- NChar member
 - of IfxType enumeration 94
- NCHAR type 15
- Negate method
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- Nested string literals 15
- NextResult method
 - of IfxDataReader class 51
- Nobuffer member
 - of IfxSmartLOBOpenMode enumeration 85
- NoLog member
 - of IfxSmartLOBCreateTimeFlags enumeration 83
- None value 31
- NotEquals method
 - of IfxDatetime structure 60
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- Now property
 - of IfxDatetime structure 59
- Null property
 - of IfxBlob 21
 - of IfxClob 26
 - of IfxDatetime structure 59
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 73
 - of IfxTimeSpan structure 87
- Null values 79
- NVarChar member
 - of IfxType enumeration 94

O

ODBC .NET 3

- ODBC connection string 38
- ODBC Driver 3
- OLE DB .NET 3
- OLE DB Provider 3
- One property
 - of IfxDecimal structure 66
- Open method
 - of IfxBlob class 23
 - of IfxClob class 28
 - of IfxConnection class 44
- Optimize OpenFetchClose connection string attribute 38
- OPTOFC connection string attribute 38
- OUT parameters
 - used during execution of SPL 10
- Output parameters 8, 79
- OutputParameters value 31

P

- Packet Size connection string attribute 38
- PacketSize property
 - of IfxConnection class 37
- Parameter types 16
- ParameterCollection.Add method 9
- ParameterName property
 - of IfxParameter class 79
- Parameters
 - used during execution of SPL 10
- Parameters property
 - of IfxCommand 31
- Parse method
 - of IfxDatetime structure 60
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- Passthrough value 47
- Password connection string attribute 38
- Performance 50
- Persist Security Info attribute 38, 38
- Persist Security Info connection string attribute 38
- Pi property
 - of IfxDecimal structure 66
- Placeholder symbols 9
- Platforms 3
- Pooling attribute 9, 10
- Pooling connection string attribute 38
- Port numbers 38
- Position property
 - of IfxBlob 21
 - of IfxClob 26
- Prepare method
 - of IfxCommand class 32
- Primary key 8, 8
- PRO connection string attribute 38
- Properties
 - IfxErrorCollection class 71
 - of IfxBlob class 21
 - of IfxCommand class 31
 - of IfxCommandBuilder class 35
 - of IfxConnection class 37
 - of IfxConnectionStringBuilder class 45
 - of IfxDataAdapter class 47
 - of IfxDataReader class 50
 - of IfxDatasourceEnumerator class 53
 - of IfxDatetime structure 59
 - of IfxDecimal structure 66
 - of IfxError class 71
 - of IfxException class 72
 - of IfxMonthSpan structure 73
 - of IfxParameter class 79

- of IfxParameterCollection class 80
- of IfxTimeSpan structure 87
- of IfxTransaction class 93
- Protocol connection string attribute 38
- Provider independence 5
- Public base classes 18
- Public interfaces 17
- Public interfaces, objects 5
- PWD connection string attribute 38

Q

- QuotePrefix property 8
 - of IfxCommandBuilder class 35
- QuoteSuffix property 8
 - of IfxCommandBuilder class 35

R

- range of time types 57, 72
- Read method
 - of IfxBlob class 23
 - of IfxClob class 28
 - of IfxDataReader class 51
- Read-only data 5, 50, 96
- ReadOnly member
 - of IfxSmartLOBOpenMode enumeration 85
- ReadWrite member
 - of IfxSmartLOBOpenMode enumeration 85
- REAL type 15
- Reconciling changed data 5, 8, 35, 47, 99
- RecordsAffected property
 - of IfxDataReader class 50
- ReferenceCount property
 - of IfxBlob 21
 - of IfxClob 26
- RefreshSchema method 8
 - of IfxCommandBuilder class 36
- Registry 7
- Relationships 5
- Release method
 - of IfxBlob class 23
- Remainder method
 - of IfxDecimal structure 66
- Remove method
 - of IfxConnectionStringBuilder class 46
 - of IfxParameterCollection class 81
- RemoveAt method
 - of IfxParameterCollection class 81
- Retrieving a single value, example of 96
- Retrieving data 96
- Retrieving data into a DataSet
 - example of 98
- Retrieving multiple rows
 - example of 96
- ReturnProviderSpecificTypes property
 - of IfxDataAdapter class 47
- Rollback method
 - of IfxTransaction class 93
- Round method
 - of IfxDecimal structure 66
- Row member
 - of IfxType enumeration 94
- RowFetchCount property
 - of IfxCommand 31
- RowUpdating events 35

S

- Sample programs
 - Informix .NET Provider 95
- SchemaOnly value 32
- Second member

- of IfxTimeUnit enumeration 92
- Second property
 - of IfxDateTime structure 59
- Seconds property
 - of IfxTimeSpan structure 87
- Seek method
 - of IfxBlob class 23
 - of IfxClob class 28
- SELECT statements 5
- SelectCommand property 8, 35
 - of IfxDataAdapter class 47
- SequentialAccess value 32
- Serial member
 - of IfxType enumeration 94
- SERIAL type 15
- Serial8 member
 - of IfxType enumeration 94
- Server connection string attribute 38
- Server member
 - of IfxSmartLOBFileLocation enumeration 84
- ServerVersion property
 - of IfxConnection class 37
- Service connection string attribute 38
- Set member
 - of IfxType enumeration 94
- SET type 15
- Setnet utility 7
- Shared member
 - of IfxSmartLOBLockMode enumeration 85
- Single-quote characters 15
- Single-table updates 35
- SingleResult value 32
- SingleRow value 32
- SmallFloat member
 - of IfxType enumeration 94
- SMALLFLOAT type 15
- SmallInt member
 - of IfxType enumeration 94
- SmartLOBLocator member
 - of IfxType enumeration 94
- SourceColumn property
 - of IfxParameter class 79
- SourceVersion property
 - of IfxParameter class 79
- Special characters 8
- sqlhosts
 - IPv6 IP addresses in 4
 - using hostnames in 4
- SQLState property
 - of IfxError class 71
- StackTrace property
 - of IfxException class 72
- start time unit 57, 72
- StartTimeUnit property
 - of IfxDateTime structure 59
 - of IfxMonthSpan structure 73
 - of IfxTimeSpan structure 87
- State property
 - of IfxConnection class 37
- StateChange event
 - of IfxConnection class 44
- Stored procedures 10, 36, 100
- StoredProcedure value 31
- String literals 15
- String type 15
- Strong-named assembly 4
- Subtract method
 - of IfxDecimal structure 66
- sysmaster database 5
- System.Data.Common namespace 13

- System.DataSet class
 - Reconciling changes with the database example of 99
- System.EnterpriseServices namespace 5
- System.Transaction namespace 5
- SystemException exception 47

T

- TableDirect value 31
- TableMappings property
 - of IfxDataAdapter class 47
- TargetSite property
 - of IfxException class 72
- Text member
 - of IfxType enumeration 94
- TEXT type 15
- Text value 31
- Thread-safety of provider 6
- Ticks property
 - of IfxDateTime structure 59
 - of IfxTimeSpan structure 87
- Time units
 - in IfxDateTime 57
- Today property
 - of IfxDateTime structure 59
- ToFile method
 - of IfxBlob class 23
- ToString method
 - of IfxConnectionStringBuilder class 46
 - of IfxDateTime structure 60
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 74
 - of IfxTimeSpan structure 89
- TotalMonths property
 - of IfxMonthSpan structure 73
- Tracing 14
- Transaction property 8, 93
 - of IfxCommand 31
- Transactions 93
 - distributed 10, 38, 100
 - local
 - example of use 97
- Truncate method
 - of IfxBlob class 23
 - of IfxClob class 28
 - of IfxDecimal structure 66
- TryGetValue method
 - of IfxConnectionStringBuilder class 46

U

- UID connection string attribute 38
- Unlock method
 - of IfxBlob class 23
 - of IfxClob class 28
- Unsupported methods 17, 18
- Update method
 - of IfxDataAdapter class 49
- UPDATE statements 35
- UPDATE, automatic 8
- UpdateBatchSize property
 - of IfxDataAdapter class 47
- UpdateCommand property 8
 - of IfxDataAdapter class 47
- UpdatedRowSource property
 - of IfxCommand 31
- Upgrading .NET applications 4
- User ID connection string attribute 38
- UserDefinedTypeFormat
 - attributes of 41
- UserDefinedTypeFormat property
 - attributes of 41
 - of IfxConnection class 37

V

- Value property
 - of IfxParameter class 79
- Values property
 - of IfxConnectionStringBuilder class 45
- VarChar member
 - of IfxType enumeration 94
- VARCHAR type 15
- VisibleFieldCount property
 - of IfxDataReader class 50
- Visual BASIC .NET 3
- Visual C# .NET 3
- Visual J# .NET 3

W

- Warning messages 71
- Write method
 - of IfxBlob class 23
 - of IfxClob class 28
- WriteOnly member
 - of IfxSmartLOBOpenMode enumeration 85

X

- XCL connection string attribute 38

Y

- Year member
 - of IfxTimeUnit enumeration 92
- Year property
 - of IfxDateTime structure 59
- Years property
 - of IfxMonthSpan structure 73

Z

- Zero property
 - of IfxDecimal structure 66
 - of IfxMonthSpan structure 73
 - of IfxTimeSpan structure 87