

Contents

Chapter 1

Namespace

FirebirdSql.Data.Firebird

Namespace Contents

Page

Interfaces

Classes

FbCommand??	??
<i>Represents an SQL statement or stored procedure to execute against a data source. This class cannot be inherited.</i>	
FbCommandBuilder??	??
<i>Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated data source. This class cannot be inherited.</i>	
FbConnection??	??
<i>Represents an open connection to a Firebird database. This class cannot be inherited.</i>	
FbDataAdapter??	??
<i>Represents a set of data commands and a connection to a data source that are used to fill the DataSet and update the data source. This class cannot be inherited.</i>	
FbDataReader??	??
<i>Provides a means of reading a forward-only stream of rows from a Firebird database. This class cannot be inherited.</i>	
FbError??	??
<i>Collects information relevant to a warning or error returned by Firebird. This class cannot be inherited.</i>	
FbErrorCollection??	??
<i>Collects all errors generated by the Firebird .NET Data Provider. This class cannot be inherited.</i>	
FbException??	??
<i>The exception that is thrown when Firebird Server returns a warning or error. This class cannot be inherited.</i>	

FbInfoMessageEventArgs	??
<i>Provides data for the InfoMessage event. This class cannot be inherited.</i>	
FbParameter	??
<i>Represents a parameter to a FbCommand, and optionally, its mapping to DataSet columns. This class cannot be inherited.</i>	
FbParameterCollection	??
<i>Collects all parameters relevant to a FbCommand as well as their respective mappings to DataSet columns. This class cannot be inherited.</i>	
FbPermission	??
<i>Enables the Firebird .NET Data Provider to ensure that a user has a security level adequate to access an Firebird data source. This class cannot be inherited.</i>	
FbPermissionAttribute	??
<i>Associates a security action with a custom security attribute.</i>	
FbRowUpdatedEventArgs	??
<i>Provides data for the RowUpdated event. This class cannot be inherited.</i>	
FbRowUpdatingEventArgs	??
<i>Provides data for the RowUpdating event. This class cannot be inherited.</i>	
FbTransaction	??
<i>Represents a Firebird transaction to be made in a Firebird database. This class cannot be inherited.</i>	

1.1 Interfaces

1.2 Classes

1.2.1 CLASS FbCommand

Represents an SQL statement or stored procedure to execute against a data source. This class cannot be inherited. The FbCommand class provides the following methods for executing commands against a Firebird database:

MethodDescriptionExecuteReader Executes commands that return rows.
ExecuteNonQuery Executes commands such as SQL INSERT, DELETE, UPDATE, and SET statements.
ExecuteScalar Retrieves a single value (for example, an aggregate value) from a database.

DECLARATION

```
public class FbCommand
    : Component
```

PROPERTIES

- *CommandText*

```
public string CommandText { get; set; }
```

Gets or sets the SQL statement or stored procedure to execute against the data source.

– Usage

- * When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure. The user may be required to use escape character syntax if the stored procedure name contains any special characters. The command executes this stored procedure when you call one of the Execute methods.

The FirebirdSql.NET Data Provider support the question mark (?) placeholder and named parameters for passing parameters to a SQL Statement or a stored procedure called by a Command of CommandType.Text.

For example:

```
SELECT * FROM Customers WHERE CustomerID =
@CustomerID
```

or

```
SELECT * FROM Customers WHERE CustomerID = ?
```

- *CommandTimeout*

```
public int CommandTimeout { get; set; }
```

Gets or sets the wait time before terminating an attempt to execute a command and generating an error.

- **Usage**

- * A value of 0 indicates no limit, and should be avoided in a CommandTimeout because an attempt to execute a command will wait indefinitely. Not currently supported.

- *CommandType*

```
public System.Data.CommandType CommandType { get; set; }
```

Gets or sets a value indicating how the CommandText property is interpreted.

- **Usage**

- * When you set the CommandType property to StoredProcedure, you should set the CommandText property to the name of the stored procedure. The command executes this stored procedure when you call one of the Execute methods.

- *Connection*

```
public FirebirdSql.Data.Firebird.FbConnection Connection { get; set; }
```

Gets or sets the FbConnection used by this instance of the FbCommand.

- **Usage**

- * You cannot set the Connection, CommandType, and CommandText properties if the current connection is performing an execute or fetch operation.
If you set Connection while a transaction is in progress and the property is not null, an InvalidOperationException is generated. If you set Connection after the transaction has been committed or rolled back, and the Transaction property is not null, the Transaction property is then set to a null value.

- *Container*

```
public System.ComponentModel.IContainer Container { get; }
```

Gets the that contains the .

- *DesignMode*

```
protected bool DesignMode { get; }
```

Gets a value that indicates whether the is currently in design mode.

- *DesignTimeVisible*

```
public bool DesignTimeVisible { get; set; }
```

Gets or sets a value indicating whether the command object should be visible in a Windows Forms Designer control.

- *Events*

```
protected System.ComponentModel.EventHandlerList Events {  
get; }
```

Gets the list of event handlers that are attached to this .

- *Parameters*

```
public FirebirdSql.Data.Firebird.FbParameterCollection  
Parameters { get; }
```

Gets the FbParameterCollection.

- *Site*

```
public System.ComponentModel.ISite Site { get; set; }
```

Gets or sets the of the .

- *System.Data.IDbCommand.Connection*

```
private System.Data.IDbConnection  
System.Data.IDbCommand.Connection { get; set; }
```

- *System.Data.IDbCommand.Parameters*

```
private System.Data.IDataParameterCollection  
System.Data.IDbCommand.Parameters { get; }
```

- *System.Data.IDbCommand.Transaction*

```
private System.Data.IDbTransaction
System.Data.IDbCommand.Transaction { get; set; }
```

- *Transaction*

```
public FirebirdSql.Data.Firebird.FbTransaction Transaction {
get; set; }
```

Gets or sets the FbTransaction within which the FbCommand executes.

– **Usage**

- * You cannot set the Transaction property if it is already set to a specific value, and the command is in the process of executing. If you set the transaction property to an FbTransaction object that is not connected to the same FbConnection as the FbCommand object, an exception will be thrown the next time you attempt to execute a statement.

- *UpdatedRowSource*

```
public System.Data.UpdateRowSource UpdatedRowSource {
get; set; }
```

Gets or sets a value that specifies how the Update method should apply command results to the DataRow.

– **Usage**

- * The default UpdateRowSource value is Both unless the command is automatically generated (as in the case of the FbCommandBuilder), in which case the default is None.

CONSTRUCTORS

- *.ctor*

```
public FbCommand( )
```

Initializes a new instance of the FbCommand class.

- *.ctor*

```
public FbCommand( )
```

Initializes a new instance of the FbCommand class with the text of the query.

– **Parameters**

* `cmdText` -

- *.ctor*

```
public FbCommand( )
```

Initializes a new instance of the FbCommand class with the text of the query and an FbConnection object.

– **Parameters**

* `cmdText` -

* `connection` -

- *.ctor*

```
public FbCommand( )
```

Initializes a new instance of the FbCommand class with the text of the query, an FbConnection object and the Transaction.

– **Parameters**

* `cmdText` -

* `connection` -

* `transaction` -

METHODS

- *Cancel*

```
public void Cancel( )
```

Attempts to cancel the execution of an FbCommand.

– **Usage**

* Not currently supported.

- *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

- **Parameters**

- * `requestedType` -

- *CreateParameter*

```
public FirebirdSql.Data.Firebird.FbParameter CreateParameter( )
```

Creates a new instance of an FbParameter object.

- *Dispose*

```
public void Dispose( )
```

Releases all resources used by the .

- *Dispose*

```
protected void Dispose( )
```

Releases the unmanaged and, optionally, the managed resources used by the FbCommand object.

- **Usage**

- * This method is called by the public Dispose method and the Finalize method. Dispose() invokes the protected Dispose(Boolean) method with the disposing parameter set to true. Finalize invokes Dispose with disposing set to false. When the disposing parameter is true, the method releases all resources held by any managed objects that this FbCommand references.

It does this by invoking the Dispose() method of each referenced object. Notes: Dispose can be called multiple times by other objects. When overriding Dispose(Boolean), be careful not to reference objects that have been previously disposed of in an earlier call to Dispose. For more information about how to implement Dispose(Boolean), see "Implementing a Dispose Method" in the Microsoft .NET Framework SDK documentation.

Calling Dispose on a FbConnection object is different from calling Close. For example, Dispose clears the connection string while

Close does not. For more information about Dispose and Finalize, see "Cleaning Up Unmanaged Resources," and "Overriding the Finalize Method," in the .NET Framework SDK documentation.

– **Parameters**

* disposing -

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

- *ExecuteNonQuery*

```
public int ExecuteNonQuery( )
```

Executes an SQL statement against the Connection and returns the number of rows affected.

– **Usage**

* You can use ExecuteNonQuery to perform catalog operations (for example, querying the structure of a database or creating database objects such as tables); or to change the data in a database, without using a DataSet, by executing UPDATE, INSERT, or DELETE statements.

Although ExecuteNonQuery does not return any rows, any output parameters or return values mapped to parameters are populated with data.

- *ExecuteReader*

```
public FirebirdSql.Data.Firebird.FbDataReader ExecuteReader( )
```

Sends the CommandText to the Connection and builds an FbDataReader.

- *ExecuteReader*

```
public FirebirdSql.Data.Firebird.FbDataReader ExecuteReader( )
```

Sends the CommandText to the Connection, and builds an FbDataReader using one of the CommandBehavior values.

– **Parameters**

* **behavior** -

- *ExecuteScalar*

public object ExecuteScalar()

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

- *Finalize*

protected void Finalize()

Releases unmanaged resources and performs other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

public int GetHashCode()

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetLifetimeService*

public object GetLifetimeService()

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetService*

protected object GetService()

Returns an object that represents a service provided by the or by its .

– **Parameters**

* **service** -

- *GetType*

public System.Type GetType()

Gets the of the current instance.

- *ICloneable.Clone*

```
private object ICloneable.Clone( )
```

- *IDbCommand.CreateParameter*

```
private System.Data.IDbDataParameter
IDbCommand.CreateParameter( )
```

- *IDbCommand.ExecuteReader*

```
private System.Data.IDataReader IDbCommand.ExecuteReader(
)
```

– **Parameters**

* behavior -

- *IDbCommand.ExecuteReader*

```
private System.Data.IDataReader IDbCommand.ExecuteReader(
)
```

- *InitializeLifetimeService*

```
public object InitializeLifetimeService( )
```

Obtains a lifetime service object to control the lifetime policy for this instance.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *Prepare*

```
public void Prepare( )
```

Creates a prepared (or compiled) version of the command at the data source.

– **Usage**

*

- *ToString*

```
public string ToString( )
```

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.2 CLASS FbCommandBuilder

Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated data source. This class cannot be inherited. The FbDataAdapter does not automatically generate the SQL statements required to reconcile changes made to a DataSet associated with the data source. However, you can create an FbCommandBuilder object that generates SQL statements for single-table updates by setting the SelectCommand property of the FbDataAdapter. Then, the FbCommandBuilder generates any additional SQL statements that you do not set. To generate INSERT, UPDATE, or DELETE statements, the FbCommandBuilder uses the SelectCommand property to retrieve a required set of metadata. If you change the value of SelectCommand after the metadata has been retrieved (for example, after the first update), you then should call the RefreshSchema method to update the metadata.

DECLARATION

```
public class FbCommandBuilder
    : Component
```

PROPERTIES

- *Container*

```
public System.ComponentModel.IContainer Container { get; }
```

Gets the that contains the .

- *DataAdapter*

```
public FirebirdSql.Data.Firebird.FbDataAdapter DataAdapter {
    get; set; }
```

Gets or sets an FbDataAdapter object for which this FbCommandBuilder object will generate SQL statements.

- **Usage**

- * The FbCommandBuilder registers itself as a listener for RowUpdating events that are generated by the FbDataAdapter specified in this property.

- *DesignMode*

```
protected bool DesignMode { get; }
```

Gets a value that indicates whether the is currently in design mode.

- *Events*

```
protected System.ComponentModel.EventHandlerList Events {
    get; }
```

Gets the list of event handlers that are attached to this .

- *QuotePrefix*

```
public string QuotePrefix { get; set; }
```

Gets or sets the beginning character or characters to use when working with database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

- **Usage**

- * Some data sources may contain objects whose names include characters such as spaces, commas, and semicolons. To accommodate this, use the QuotePrefix and QuoteSuffix properties to specify delimiters, such as a left and right bracket, that will encapsulate the object name. Note: Although you cannot change the QuotePrefix or QuoteSuffix properties after an insert, update, or delete command has been generated, you can change their settings after calling the Update method of an FbDataAdapter.

- *QuoteSuffix*

```
public string QuoteSuffix { get; set; }
```

Gets or sets the ending character or characters to use when working with database objects, (for example, tables or columns), whose names contain characters such as spaces or reserved tokens.

- **Usage**

- * Some data sources may contain objects whose names include characters such as spaces, commas, and semicolons. To accommodate this, use the QuotePrefix and QuoteSuffix properties to specify delimiters, such as a left and right bracket, that will encapsulate the object name. Note: Although you cannot change the QuotePrefix or QuoteSuffix properties after an insert, update, or delete command has been generated, you can change their settings after calling the Update method of an FbDataAdapter.

- *Site*

```
public System.ComponentModel.ISite Site { get; set; }
```

Gets or sets the of the .

CONSTRUCTORS

- *.ctor*

```
public FbCommandBuilder( )
```

Initializes a new instance of the FbCommandBuilder class.

- *.ctor*

```
public FbCommandBuilder( )
```

– **Parameters**

* adapter -

METHODS

- *BuildDeleteCommand*

```
public FirebirdSql.Data.Firebird.FbCommand  
BuildDeleteCommand( )
```

Builds delete command.

– **Parameters**

* row -

* tableMapping -

- *BuildUpdateCommand*

```
public FirebirdSql.Data.Firebird.FbCommand  
BuildUpdateCommand( )
```

Builds update command.

– **Parameters**

* row -

* tableMapping -

- *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

- **Parameters**

- * `requestedType` -

- *DeriveParameters*

```
public void DeriveParameters( )
```

Populates the specified FbCommand object's Parameters collection with parameter information for a stored procedure specified in the FbCommand.

- **Usage**

- * DeriveParameters overwrites any existing parameter information for the FbCommand. DeriveParameters requires an extra call to the data server to obtain the information. If the parameter information is known in advance, it is more efficient to populate the parameters collection by setting the information explicitly. You can only use DeriveParameters with stored procedures. You cannot use DeriveParameters to populate the FbParameterCollection with arbitrary DSQL statements, such as a parameterized SELECT statement.

- **Parameters**

- * `command` -

- *Dispose*

```
public void Dispose( )
```

Releases all resources used by the .

- *Dispose*

```
protected void Dispose( )
```

Releases the unmanaged and, optionally, the managed resources used by the FbCommand object.

- **Usage**

* This method is called by the public Dispose method and the Finalize method. Dispose() invokes the protected Dispose(Boolean) method with the disposing parameter set to true. Finalize invokes Dispose with disposing set to false. When the disposing parameter is true, the method releases all resources held by any managed objects that this FbCommand references.

It does this by invoking the Dispose() method of each referenced object. Notes: Dispose can be called multiple times by other objects. When overriding Dispose(Boolean), be careful not to reference objects that have been previously disposed of in an earlier call to Dispose. For more information about how to implement Dispose(Boolean), see "Implementing a Dispose Method" in the Microsoft .NET Framework SDK documentation.

Calling Dispose on a FbConnection object is different from calling Close. For example, Dispose clears the connection string while Close does not. For more information about Dispose and Finalize, see "Cleaning Up Unmanaged Resources," and "Overriding the Finalize Method," in the .NET Framework SDK documentation.

– **Parameters**

* disposing -

• *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

• *Finalize*

```
protected void Finalize( )
```

• *GetDeleteCommand*

```
public FirebirdSql.Data.Firebird.FbCommand  
GetDeleteCommand( )
```

Gets the automatically generated FbCommand object required to perform deletions at the data source.

– **Usage**

- * You can use the `GetDeleteCommand` method for informational or troubleshooting purposes because it returns the `FbCommand` object to be executed. You can also use `GetDeleteCommand` as the basis of a modified command. For example, you might call `GetDeleteCommand` and modify the `CommandTimeout` value, and then explicitly set that on the `FbDataAdapter`. After the SQL statement is first generated, you must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetDeleteCommand` still will be using information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either `Update` or `GetDeleteCommand`.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetInsertCommand*

```
public FirebirdSql.Data.Firebird.FbCommand
GetInsertCommand( )
```

Gets the automatically generated `FbCommand` object required to perform insertions at the data source.

– **Usage**

- * You can use the `GetInsertCommand` method for informational or troubleshooting purposes because it returns the `FbCommand` object to be executed. You can also use `GetInsertCommand` as the basis of a modified command. For example, you might call `GetInsertCommand` and modify the `CommandTimeout` value, and then explicitly set that on the `FbDataAdapter`. After the SQL statement is first generated, you must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetInsertCommand` still will be using information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either `Update` or `GetInsertCommand`.

- *GetLifetimeService*

public object GetLifetimeService()

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetService*

protected object GetService()

Returns an object that represents a service provided by the or by its .

– **Parameters**

* *service* -

- *GetType*

public System.Type GetType()

Gets the of the current instance.

- *GetUpdateCommand*

**public FirebirdSql.Data.Firebird.FbCommand
GetUpdateCommand()**

Gets the automatically generated FbCommand object required to perform updates at the data source.

– **Usage**

- * You can use the GetUpdateCommand method for informational or troubleshooting purposes because it returns the FbCommand object to be executed. You can also use GetUpdateCommand as the basis of a modified command.

For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that on the FbDataAdapter.

After the SQL statement is first generated, you must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetUpdateCommand still will be using information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either Update or GetUpdateCommand.

- *InitializeLifetimeService*

public object InitializeLifetimeService()

Obtains a lifetime service object to control the lifetime policy for this instance.

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *RefreshSchema*

public void RefreshSchema()

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

– **Usage**

- * You should call RefreshSchema whenever the SelectCommand value of the FbDataAdapter changes.

- *ToString*

public string ToString()

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.3 CLASS FbConnection

Represents an open connection to a Firebird database. This class cannot be inherited. A FbConnection object represents a unique connection to Firebird server. FbConnection is used in conjunction with FbDataAdapter and FbCommand to increase performance when connecting to Firebird database.

If the FbConnection goes out of scope, it is not closed. Therefore, you must explicitly close the connection by calling Close or Dispose.

DECLARATION

```
public class FbConnection
    : Component
```

PROPERTIES

- *ConnectionString*

```
public string ConnectionString { get; set; }
```

Gets or sets the string used to open a FirebirdSQL database.

– Usage

- * The following table lists the valid names for keyword values within the ConnectionString:NameDescriptionDefaultDatabaseDatabase path to establish the connectionUserFirebird User account for loginPasswordPassword for the Firebird user accountDialectDatabase dialect3HostServer name for establish the connectionLocalhostPortPort number in the server for establish the connection3050CharsetDatabase Character SetNONERoleUser RolePacket SizeSize (in bytes) of network packets used to communicate with an instance of Firebird SQL Server.8192Connection LifetimeWhen a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by connection lifetime. 0Pooling When true, the FbConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are true, false, yes, and no. true The following table lists the valid names for keyword Charset of the ConnectionString: Firebird CharsetDescriptionASCIIAmerican Standard Code for Information Interchange.BIG_5Big5, Traditional Chinese.DOS437MS-DOS United States, Australia, New Zealand, South Africa.DOS850MS-DOS Latin-1.DOS860MS-DOS

Portugues.DOS861MS-DOS Icelandic.DOS863MS-DOS Canadian
 French.DOS865MS-DOS Nordic.EUCJ.0208JIS X 0201, 0208, 0212,
 EUC encoding, Japanese.GB.2312GB2312, EUC encoding,
 Simplified Chinese.ISO8859_1ISO 8859-1, Latin alphabet No.
 1.ISO8859_2ISO 8859-2, Latin alphabet No. 2.KSC.5601Windows
 Korean.ISO2022-JPWindows Japanese.SJIS.0208Japanese
 (Shift-JIS)UNICODE_FSSEight-bit Unicode Transformation
 Format.WIN1250Windows Eastern European.WIN1251Windows
 Cyrillic.WIN1252Windows Latin-1.WIN1253Windows
 Greek.WIN1254Windows Turkish.WIN1254Windows
 Hebrew.ArabicWindows Turkish.WIN1257Windows Baltic.

- *ConnectionTimeout*

```
public int ConnectionTimeout { get; }
```

Gets the time to wait while trying to establish a connection before terminating the attempt and generating an error.

- **Usage**

- * A value of 0 indicates no limit, and should be avoided in a ConnectionString because an attempt to connect will wait indefinitely. Not currently supported.

- *Container*

```
public System.ComponentModel.IContainer Container { get; }
```

Gets the that contains the .

- *Database*

```
public string Database { get; }
```

Gets the name of the current database or the database to be used once a connection is open.

- *DataSource*

```
public string DataSource { get; }
```

Gets the name of the instance of FirebirdSQL to which to connect.

- *DesignMode*

```
protected bool DesignMode { get; }
```

Gets a value that indicates whether the is currently in design mode.

- *Events*

```
protected System.ComponentModel.EventHandlerList Events {
get; }
```

Gets the list of event handlers that are attached to this .

- *PacketSize*

```
public int PacketSize { get; }
```

Gets the size (in bytes) of network packets used to communicate with an instance of FirebirdSQL.

- **Usage**

- * If an application performs bulk copy operations, or sends or receives large amounts of text or image data, a packet size larger than the default may improve efficiency because it results in fewer network read and write operations. If an application sends and receives small amounts of information, you can set the packet size to 512 bytes (using the Packet Size value in the ConnectionString), which is sufficient for most data transfer operations. For most applications, the default packet size is best. PacketSize may be a value in the range of 512 and 32767 bytes. An exception is generated if the value is outside of this range.

- *ServerVersion*

```
public string ServerVersion { get; }
```

Gets a string containing the version of the instance of Firebird Server to which the client is connected.

- *Site*

```
public System.ComponentModel.ISite Site { get; set; }
```

Gets or sets the of the .

- *State*

```
public System.Data.ConnectionState State { get; }
```

Gets the current state of the connection.

– **Usage**

- * The allowed state changes are: From Closed to Open, using the Open method of the connection object.
From Open to Closed, using either the Close method or the Dispose method of the connection object.

CONSTRUCTORS

- *.ctor*

```
public FbConnection( )
```

Initializes a new instance of the FbConnection class.

– **Usage**

- * When a new instance of FbConnection is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the ConnectionString property.

- *.ctor*

```
public FbConnection( )
```

Initializes a new instance of the FbConnection class when given a string containing the connection string.

– **Usage**

- * When a new instance of FbConnection is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the ConnectionString property.

– **Parameters**

- * `connString` -

METHODS

- *BeginTransaction*

```
public FirebirdSql.Data.Firebird.FbTransaction  
BeginTransaction( )
```

Begins a database transaction.

– **Usage**

- * To commit or rollback the transaction, you must explicitly use the Commit or Rollback methods.

– **Example**

The following example creates an FbConnection and an FbTransaction. It also demonstrates how to use the BeginTransaction, Commit, and Rollback methods.

```
public void RunFirebirdTransaction(string myConnString) {
    FbConnection myConnection = new FbConnection(myConnString);
    myConnection.Open();
    FbCommand myCommand = new FbCommand(); FbTransaction
    myTrans;
    // Start a local transaction myTrans =
    myConnection.BeginTransaction(); // Assign transaction
    object for a pending local transaction myCommand.Connection
    = myConnection; myCommand.Transaction = myTrans;
    try { myCommand.CommandText = "Insert into Region (RegionID,
    RegionDescription) VALUES (100, 'Description')";
    myCommand.ExecuteNonQuery(); myCommand.CommandText = "Insert
    into Region (RegionID, RegionDescription) VALUES (101,
    'Description')"; myCommand.ExecuteNonQuery();
    myTrans.Commit(); Console.WriteLine("Both records are
    written to database."); } catch(Exception e) {
    myTrans.Rollback(); Console.WriteLine(e.ToString());
    Console.WriteLine("Neither record was written to
    database."); } finally { myConnection.Close(); } }
```

- *BeginTransaction*

```
public FirebirdSql.Data.Firebird.FbTransaction
BeginTransaction( )
```

Begins a database transaction with the specified transaction name.

– **Usage**

- * You must explicitly commit or roll back the transaction using the Commit or Rollback method.

– **Parameters**

- * transactionName -

– **Example**

The following example creates an FbConnection and an FbTransaction. It also demonstrates how to use the BeginTransaction, Commit, and Rollback methods.

```

FbConnection myConnection = new
FbConnection(connectionString); myConnection.Open();
FbCommand myCommand = new FbCommand(); FbTransaction
myTrans;
// Start a local transaction myTrans =
myConnection.BeginTransaction(); // Assign transaction
object for a pending local transaction myCommand.Connection
= myConnection; myCommand.Transaction = myTrans;
try { myCommand.CommandText = "INSERT INTO PROJECT(proj_id,
proj_name, product) Values('FBNP', '.Net Provider', 'N/A')";
myCommand.ExecuteNonQuery();
myTrans.Save("SampleTransaction"); myCommand.CommandText =
"INSERT INTO PROJECT(proj_id, proj_name, product)
Values('FBN1', '.Net Provider1.', 'N/A')";
myCommand.ExecuteNonQuery(); myTrans.Commit();
Console.WriteLine("Both records are written to database.");
} catch (Exception e) { try {
myTrans.Rollback("SampleTransaction"); } catch (FbException
ex) { if (myTrans.Connection != null) {
Console.WriteLine("An exception of type " + ex.GetType() + "
was encountered while attempting to roll back the
transaction."); } }
Console.WriteLine("An exception of type " + e.GetType() + "
was encountered while inserting the data.");
Console.WriteLine("Neither record was written to
database."); } finally { myConnection.Close(); }

```

- *BeginTransaction*

```

public FirebirdSql.Data.Firebird.FbTransaction
BeginTransaction( )

```

Begins a transaction at the database with the specified IsolationLevel value.

– Usage

- * You must explicitly commit or roll back the transaction using the Commit or Rollback method.

```

public void RunFirebirdTransaction(string myConnString) {
FbConnection myConnection = new
FbConnection(myConnString); myConnection.Open();
FbCommand myCommand = new FbCommand(); FbTransaction
myTrans;
// Start a local transaction myTrans =
myConnection.BeginTransaction(IsolationLevel.ReadCommitted);
// Assign transaction object for a pending local transaction

```

```

myCommand.Connection = myConnection;
myCommand.Transaction = myTrans;
try { myCommand.CommandText = "Insert into Region
(RegionID, RegionDescription) VALUES (100, 'Description')";
myCommand.ExecuteNonQuery(); myCommand.CommandText =
"Insert into Region (RegionID, RegionDescription) VALUES (101,
'Description')"; myCommand.ExecuteNonQuery();
myTrans.Commit(); Console.WriteLine("Both records are written
to database."); } catch(Exception e) { myTrans.Rollback();
Console.WriteLine(e.ToString()); Console.WriteLine("Neither
record was written to database."); } finally {
myConnection.Close(); } }

```

– **Parameters**

* *level* -

• *BeginTransaction*

```

public FirebirdSql.Data.Firebird.FbTransaction
BeginTransaction( )

```

Begins a database transaction with the specified isolation level and transaction name.

– **Usage**

```

* You must explicitly commit or roll back the transaction using the
Commit or Rollback method.
FbConnection myConnection = new
FbConnection(connectionString); myConnection.Open();
FbCommand myCommand = new FbCommand(); FbTransaction
myTrans;
// Start a local transaction myTrans =
myConnection.BeginTransaction(); // Assign transaction object for
a pending local transaction myCommand.Connection =
myConnection; myCommand.Transaction = myTrans;
try { myCommand.CommandText = "INSERT INTO
PROJECT(proj_id, proj_name, product) Values('FBNP', '.Net
Provider', 'N/A')"; myCommand.ExecuteNonQuery();
myTrans.Save("SampleTransaction"); myCommand.CommandText
= "INSERT INTO PROJECT(proj_id, proj_name, product)
Values('FBN1', '.Net Provider1.', 'N/A')";
myCommand.ExecuteNonQuery(); myTrans.Commit();
Console.WriteLine("Both records are written to database."); }
catch(Exception e) { try {
myTrans.Rollback("SampleTransaction"); } catch (FbException ex)

```

```

    { if (myTrans.Connection != null) { Console.WriteLine(" An
    exception of type " + ex.GetType() + " was encountered while
    attempting to roll back the transaction."); } }
    Console.WriteLine("An exception of type " + e.GetType() + " was
    encountered while inserting the data.");
    Console.WriteLine("Neither record was written to database."); }
    finally { myConnection.Close(); }

```

– **Parameters**

- * `level` -
- * `transactionName` -

• *ChangeDatabase*

```
public void ChangeDatabase( )
```

Changes the current database for an open FbConnection.

– **Usage**

- * Not currently supported.

– **Parameters**

- * `db` -

• *Close*

```
public void Close( )
```

Closes the connection to the database. This is the preferred method of closing any open connection.

– **Usage**

- * The Close method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled. If Close is called while handling a StateChange event, no additional StateChange events are fired.

An application can call Close more than one time without generating an exception.

• *CreateCommand*

```
public FirebirdSql.Data.Firebird.FbCommand CreateCommand(
)
```

Creates and returns a FbCommand object associated with the FbConnection.

- *CreateDatabase*

```
public void CreateDatabase( )
```

Creates a new database.

– **Parameters**

- * `dataSource` -
- * `port` -
- * `database` -
- * `user` -
- * `password` -
- * `dialect` -
- * `forceWrite` -
- * `pageSize` -
- * `charset` -

- *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

– **Parameters**

- * `requestedType` -

- *Dispose*

```
public void Dispose( )
```

Releases all resources used by the .

- *Dispose*

```
protected void Dispose( )
```

Releases the unmanaged and, optionally, the managed resources used by the FbConnection object.

– **Usage**

- * This method is called by the public `Dispose()` method and the `Finalize` method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the disposing parameter set to true. `Finalize` invokes `Dispose` with disposing set to false. When the disposing parameter is true, the method releases all resources held by any managed objects that this FbConnection

references. It does this by invoking the Dispose() method of each referenced object.

– **Parameters**

* disposing -

- *Equals*

public bool Equals()

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

- *Finalize*

protected void Finalize()

Releases unmanaged resources and performs other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

public int GetHashCode()

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetLifetimeService*

public object GetLifetimeService()

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetService*

protected object GetService()

Returns an object that represents a service provided by the or by its .

– **Parameters**

* service -

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *ICloneable.Clone*

```
private object ICloneable.Clone( )
```

– **Usage**

* Not currently supported.

- *IDbConnection.BeginTransaction*

```
private System.Data.IDbTransaction  
IDbConnection.BeginTransaction( )
```

Begins a transaction at the database with the specified IsolationLevel value.

– **Usage**

* You must explicitly commit or roll back the transaction using the Commit or Rollback method.

```
public void RunFirebirdTransaction(string myConnString) {  
    FbConnection myConnection = new  
    FbConnection(myConnString); myConnection.Open();  
    FbCommand myCommand = new FbCommand(); FbTransaction  
    myTrans;  
    // Start a local transaction myTrans =  
    myConnection.BeginTransaction(IsolationLevel.ReadCommitted);  
    // Assign transaction object for a pending local transaction  
    myCommand.Connection = myConnection;  
    myCommand.Transaction = myTrans;  
    try { myCommand.CommandText = "Insert into Region  
    (RegionID, RegionDescription) VALUES (100, 'Description')";  
    myCommand.ExecuteNonQuery(); myCommand.CommandText =  
    "Insert into Region (RegionID, RegionDescription) VALUES (101,  
    'Description')"; myCommand.ExecuteNonQuery();  
    myTrans.Commit(); Console.WriteLine("Both records are written  
    to database."); } catch(Exception e) { myTrans.Rollback();  
    Console.WriteLine(e.ToString()); Console.WriteLine("Neither  
    record was written to database."); } finally {  
    myConnection.Close(); } }
```

– **Parameters**

* level -

- *IDbConnection.BeginTransaction*

```
private System.Data.IDbTransaction
IDbConnection.BeginTransaction( )
```

Begins a database transaction.

– **Usage**

* To commit or rollback the transaction, you must explicitly use the Commit or Rollback methods.

– **Example**

The following example creates an FbConnection and an FbTransaction. It also demonstrates how to use the BeginTransaction, Commit, and Rollback methods.

```
public void RunFirebirdTransaction(string myConnString) {
    FbConnection myConnection = new FbConnection(myConnString);
    myConnection.Open();
    FbCommand myCommand = new FbCommand(); FbTransaction
    myTrans;
    // Start a local transaction myTrans =
    myConnection.BeginTransaction(); // Assign transaction
    object for a pending local transaction myCommand.Connection
    = myConnection; myCommand.Transaction = myTrans;
    try { myCommand.CommandText = "Insert into Region (RegionID,
    RegionDescription) VALUES (100, 'Description')";
    myCommand.ExecuteNonQuery(); myCommand.CommandText = "Insert
    into Region (RegionID, RegionDescription) VALUES (101,
    'Description')"; myCommand.ExecuteNonQuery();
    myTrans.Commit(); Console.WriteLine("Both records are
    written to database."); } catch(Exception e) {
    myTrans.Rollback(); Console.WriteLine(e.ToString());
    Console.WriteLine("Neither record was written to
    database."); } finally { myConnection.Close(); } }
```

- *IDbConnection.CreateCommand*

```
private System.Data.IDbCommand
IDbConnection.CreateCommand( )
```

Creates and returns a FbCommand object associated with the FbConnection.

- *InitializeLifetimeService*

public object InitializeLifetimeService()

Obtains a lifetime service object to control the lifetime policy for this instance.

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *Open*

public void Open()

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

– **Usage**

- * The FbConnection draws an open connection from the connection pool if one is available. Otherwise, it establishes a new connection to the database. Connection pooling is not currently supported. Note: If the FbConnection goes out of scope, the connection it represents does not close automatically. Therefore, you must explicitly close the connection by calling Close or Dispose.

- *ToString*

public string ToString()

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.4 CLASS FbDataAdapter

Represents a set of data commands and a connection to a data source that are used to fill the DataSet and update the data source. This class cannot be inherited. The FbDataAdapter, serves as a bridge between a DataSet and FirebirdSQL for retrieving and saving data. The FbDataAdapter provides this bridge by mapping Fill, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet, using the appropriate DSQL statements against the data source. FbDataAdapter is used in conjunction with FbConnection and FbCommand to increase performance when connecting to a FirebirdSQL Server database.

The FbDataAdapter also includes the SelectCommand, InsertCommand, DeleteCommand, UpdateCommand, and TableMappings properties to facilitate the loading and updating of data.

When an instance of FbDataAdapter is created, the read/write properties are set to initial values.

DECLARATION

```
public class FbDataAdapter
    : DbDataAdapter
```

PROPERTIES

- *AcceptChangesDuringFill*

```
public bool AcceptChangesDuringFill { get; set; }
```

Gets or sets a value indicating whether is called on a after it is added to the during any of the operations.

- *Container*

```
public System.ComponentModel.IContainer Container { get; }
```

Gets the that contains the .

- *ContinueUpdateOnError*

```
public bool ContinueUpdateOnError { get; set; }
```

Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.

- *DeleteCommand*

```
public FirebirdSql.Data.Firebird.FbCommand DeleteCommand {
    get; set; }
```

Gets or sets an SQL statement or stored procedure used to delete records in the data source.

- **Usage**

- * During Update, if this property is not set and primary key information is present in the DataSet, the DeleteCommand can be generated automatically if you set the SelectCommand property and use the FbCommandBuilder. Then, any additional commands that you do not set are generated by the FbCommandBuilder. This generation logic requires key column information to be present in the DataSet.

When DeleteCommand is assigned to a previously created FbCommand, the FbCommand is not cloned. The DeleteCommand maintains a reference to the previously created FbCommand object.

- *DesignMode*

```
protected bool DesignMode { get; }
```

Gets a value that indicates whether the is currently in design mode.

- *Events*

```
protected System.ComponentModel.EventHandlerList Events {
    get; }
```

Gets the list of event handlers that are attached to this .

- *InsertCommand*

```
public FirebirdSql.Data.Firebird.FbCommand InsertCommand {
    get; set; }
```

Gets or sets an SQL statement or stored procedure used to insert new records into the data source.

- **Usage**

- * During Update, if this property is not set and primary key information is present in the DataSet, the InsertCommand can be generated automatically if you set the SelectCommand property and use the FbCommandBuilder. Then, any additional commands

that you do not set are generated by the FbCommandBuilder. This generation logic requires key column information to be present in the DataSet.

When InsertCommand is assigned to a previously created FbCommand, the FbCommand is not cloned. The InsertCommand maintains a reference to the previously created FbCommand object.

- *MissingMappingAction*

```
public System.Data.MissingMappingAction
MissingMappingAction { get; set; }
```

Determines the action to take when incoming data does not have a matching table or column.

- *MissingSchemaAction*

```
public System.Data.MissingSchemaAction MissingSchemaAction
{ get; set; }
```

Determines the action to take when existing schema does not match incoming data.

- *SelectCommand*

```
public FirebirdSql.Data.Firebird.FbCommand SelectCommand {
get; set; }
```

Gets or sets an SQL statement or stored procedure used to select records in the data source.

- **Usage**

- * When SelectCommand is assigned to a previously created FbCommand, the FbCommand is not cloned. The SelectCommand maintains a reference to the previously created FbCommand object. If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

- *Site*

```
public System.ComponentModel.ISite Site { get; set; }
```

Gets or sets the of the .

- *System.Data.IDataAdapter.TableMappings*

```
private System.Data.ITableMappingCollection
System.Data.IDbDataAdapter.TableMappings { get; }
```

- *System.Data.IDbDataAdapter.DeleteCommand*

```
private System.Data.IDbCommand
System.Data.IDbDataAdapter.DeleteCommand { get; set; }
```

- *System.Data.IDbDataAdapter.InsertCommand*

```
private System.Data.IDbCommand
System.Data.IDbDataAdapter.InsertCommand { get; set; }
```

- *System.Data.IDbDataAdapter.SelectCommand*

```
private System.Data.IDbCommand
System.Data.IDbDataAdapter.SelectCommand { get; set; }
```

- *System.Data.IDbDataAdapter.UpdateCommand*

```
private System.Data.IDbCommand
System.Data.IDbDataAdapter.UpdateCommand { get; set; }
```

- *TableMappings*

```
public System.Data.Common.DataTableMappingCollection
TableMappings { get; }
```

Gets a collection that provides the master mapping between a source table and a .

- *UpdateCommand*

```
public FirebirdSql.Data.Firebird.FbCommand UpdateCommand
{ get; set; }
```

Gets or sets an SQL statement or stored procedure used to update records in the data source.

– Usage

- * During Update, if this property is not set and primary key information is present in the DataSet, the UpdateCommand can be generated automatically if you set the SelectCommand property and use the FbCommandBuilder. Then, any additional commands that you do not set are generated by the FbCommandBuilder. This

generation logic requires key column information to be present in the DataSet.

When UpdateCommand is assigned to a previously created FbCommand, the FbCommand is not cloned. The UpdateCommand maintains a reference to the previously created FbCommand object.

CONSTRUCTORS

- *.ctor*

```
public FbDataAdapter( )
```

Initializes a new instance of the FbDataAdapter class.

– **Usage**

- * When you create an instance of FbDataAdapter, the following read/write properties are set to their default values, as shown in the table. PropertiesDefault ValueMissingMappingActionMissingMappingAction.PassthroughMissingSchemaActionMissingSchemaAction.Add

- *.ctor*

```
public FbDataAdapter( )
```

Initializes a new instance of the FbDataAdapter class with the specified SQL SELECT statement.

– **Usage**

- * When you create an instance of FbDataAdapter, the following read/write properties are set to their default values, as shown in the table. PropertiesDefault ValueMissingMappingActionMissingMappingAction.PassthroughMissingSchemaActionMissingSchemaAction.Add

– **Parameters**

- * `selectCommand` -

- *.ctor*

```
public FbDataAdapter( )
```

Initializes a new instance of the FbDataAdapter class with an SQL SELECT statement and an FbConnection.

– **Usage**

- * When you create an instance of FbDataAdapter, the following read/write properties are set to their default values, as shown in the table. PropertiesDefault ValueMissingMappingActionMissingMappingAction.PassthroughMissingSchemaActionMissingSchemaAction.Add

– **Parameters**

- * `selectCommandText` -
- * `selectConnection` -

• *.ctor*

```
public FbDataAdapter( )
```

Initializes a new instance of the FbDataAdapter class with an SQL SELECT statement and a connection string.

– **Usage**

- * When you create an instance of FbDataAdapter, the following read/write properties are set to their default values, as shown in the table. PropertiesDefault ValueMissingMappingActionMissingMappingAction.PassthroughMissingSchemaActionMissingSchemaAction.Add

– **Parameters**

- * `selectCommandText` -
- * `selectConnectionString` -

METHODS

• *CloneInternals*

```
protected System.Data.Common.DataAdapter CloneInternals( )
```

Creates a copy of this instance of .

• *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

– **Parameters**

* requestedType -

- *CreateRowUpdatedEvent*

protected **System.Data.Common.RowUpdatedEventArgs**
CreateRowUpdatedEvent()

Initializes a new instance of the RowUpdatedEventArgs class, regardless of whether the update is successful.

– **Parameters**

* dataRow -
 * command -
 * statementType -
 * tableMapping -

- *CreateRowUpdatingEvent*

protected **System.Data.Common.RowUpdatingEventArgs**
CreateRowUpdatingEvent()

Initializes a new instance of the RowUpdatingEventArgs class.

– **Parameters**

* dataRow -
 * command -
 * statementType -
 * tableMapping -

- *CreateTableMappings*

protected **System.Data.Common.DataTableMappingCollection**
CreateTableMappings()

Creates a new .

- *Dispose*

public void Dispose()

Releases all resources used by the .

- *Dispose*

protected void Dispose()

Releases the unmanaged and, optionally, the managed resources used by the FbCommand object.

– **Usage**

- * This method is called by the public Dispose method and the Finalize method. Dispose() invokes the protected Dispose(Boolean) method with the disposing parameter set to true. Finalize invokes Dispose with disposing set to false. When the disposing parameter is true, the method releases all resources held by any managed objects that this FbCommand references. It does this by invoking the Dispose() method of each referenced object. Notes: Dispose can be called multiple times by other objects. When overriding Dispose(Boolean), be careful not to reference objects that have been previously disposed of in an earlier call to Dispose. For more information about how to implement Dispose(Boolean), see "Implementing a Dispose Method" in the Microsoft .NET Framework SDK documentation. Calling Dispose on a FbConnection object is different from calling Close. For example, Dispose clears the connection string while Close does not. For more information about Dispose and Finalize, see "Cleaning Up Unmanaged Resources," and "Overriding the Finalize Method," in the .NET Framework SDK documentation.

– **Parameters**

- * disposing -

• *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

- * obj -

• *Fill*

```
protected int Fill( )
```

Adds or refreshes rows in a to match those in the data source using the specified and names.

– **Parameters**

- * dataTable -
- * dataReader -

• *Fill*

protected int Fill()

Adds or refreshes rows in a specified range in the to match those in the data source using the , , and names.

– **Parameters**

- * **dataSet** -
- * **srcTable** -
- * **dataReader** -
- * **startRecord** -
- * **maxRecords** -

• *Fill*

protected int Fill()

Adds or refreshes rows in a to match those in the data source using the name, the specified SQL SELECT statement, and .

– **Parameters**

- * **dataTable** -
- * **command** -
- * **behavior** -

• *Fill*

protected int Fill()

Adds or refreshes rows in a specified range in the to match those in the data source using the and source table names, command string, and command behavior.

– **Parameters**

- * **dataSet** -
- * **startRecord** -
- * **maxRecords** -
- * **srcTable** -
- * **command** -
- * **behavior** -

• *Fill*

public int Fill()

Adds or refreshes rows in the to match those in the data source using the name, and creates a named "Table".

– **Parameters**

* dataSet -

- *Fill*

public int Fill()

Adds or refreshes rows in a to match those in the data source using the name.

– **Parameters**

* dataTable -

- *Fill*

public int Fill()

Adds or refreshes rows in the to match those in the data source using the and names.

– **Parameters**

* dataSet -

* srcTable -

- *Fill*

public int Fill()

Adds or refreshes rows in a specified range in the to match those in the data source using the and names.

– **Parameters**

* dataSet -

* startRecord -

* maxRecords -

* srcTable -

- *FillSchema*

protected System.Data.DataTable FillSchema()

Configures the schema of the specified based on the specified , command string, and values.

– **Parameters**

* dataTable -

* schemaType -

* command -

* behavior -

- *FillSchema*

```
protected System.Data.DataTable[] FillSchema( )
```

Adds a to the specified and configures the schema to match that in the data source based on the specified .

– **Parameters**

- * dataSet -
- * schemaType -
- * command -
- * srcTable -
- * behavior -

- *FillSchema*

```
public System.Data.DataTable[] FillSchema( )
```

Adds a named "Table" to the specified and configures the schema to match that in the data source based on the specified .

– **Parameters**

- * dataSet -
- * schemaType -

- *FillSchema*

```
public System.Data.DataTable FillSchema( )
```

Configures the schema of the specified based on the specified .

– **Parameters**

- * dataTable -
- * schemaType -

- *FillSchema*

```
public System.Data.DataTable[] FillSchema( )
```

Adds a to the specified and configures the schema to match that in the data source based upon the specified and .

– **Parameters**

- * dataSet -
- * schemaType -
- * srcTable -

- *Finalize*

protected void Finalize()

Releases unmanaged resources and performs other cleanup operations before the is reclaimed by garbage collection.

- *GetFillParameters*

public System.Data.IDataParameter[] GetFillParameters()

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

- *GetHashCode*

public int GetHashCode()

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetLifetimeService*

public object GetLifetimeService()

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetService*

protected object GetService()

Returns an object that represents a service provided by the or by its .

– **Parameters**

* **service** -

- *GetType*

public System.Type GetType()

Gets the of the current instance.

- *ICloneable.Clone*

private object ICloneable.Clone()

- *InitializeLifetimeService*

public object InitializeLifetimeService()

Obtains a lifetime service object to control the lifetime policy for this instance.

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *OnFillError*

protected void OnFillError()

Raises the event.

– **Parameters**

* **value** -

- *OnRowUpdated*

protected void OnRowUpdated()

Raises the RowUpdated event using a RowUpdatedEventArgs object.

– **Parameters**

* **value** -

- *OnRowUpdating*

protected void OnRowUpdating()

Raises the RowUpdating event using a RowUpdatingEventArgs object, whether or not the update operation is successful.

– **Parameters**

* **value** -

- *ShouldSerializeTableMappings*

protected bool ShouldSerializeTableMappings()

Determines whether one or more objects exist and they should be persisted.

- *Tostring*

```
public string ToString( )
```

- *Update*

```
protected int Update( )
```

Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of objects.

– **Parameters**

- * `dataRows` -
- * `tableMapping` -

- *Update*

```
public int Update( )
```

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

– **Parameters**

- * `dataSet` -

- *Update*

```
public int Update( )
```

Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of objects.

– **Parameters**

- * `dataRows` -

- *Update*

```
public int Update( )
```

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

– **Parameters**

- * `dataTable` -

- *Update*

```
public int Update( )
```

Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the with the specified name.

– **Parameters**

- * `dataSet` -
- * `srcTable` -

EXTENDED INFORMATION

- Assembly: `FirebirdSql.Data.Firebird`

1.2.5 CLASS FbDataReader

Provides a means of reading a forward-only stream of rows from a Firebird database. This class cannot be inherited. To create an FbDataReader, you must call the ExecuteReader method of the FbCommand object, rather than directly using a constructor.

While the FbDataReader is in use, the associated FbConnection is busy serving the FbDataReader, and no other operations can be performed on the FbConnection other than closing it. This is the case until the Close method of the FbDataReader is called. For example, you cannot retrieve output parameters until after you call Close. IsClosed and RecordsAffected are the only properties that you can call after the FbDataReader is closed. In some cases, you must call Close before you can call RecordsAffected.

DECLARATION

```
public class FbDataReader
    : MarshalByRefObject
```

PROPERTIES

- *Depth*

```
public int Depth { get; }
```

Gets a value indicating the depth of nesting for the current row.

- **Usage**

- * The outermost table has a depth of zero. The Firebird .NET Data Provider does not support nesting and always returns zero.

- *FieldCount*

```
public int FieldCount { get; }
```

Gets the number of columns in the current row.

- **Usage**

- * After executing a query that does not return rows, FieldCount returns 0.

- *IsClosed*

```
public bool IsClosed { get; }
```

Gets a value indicating whether the data reader is closed.

– **Usage**

- * IsClosed and RecordsAffected are the only properties that you can call after the FbDataReader is closed.

- *Item*

```
public object Item { get; }
```

Gets the value of the specified column in its native format given the column ordinal.

– **Parameters**

- * *i* -

- *Item*

```
public object Item { get; }
```

Gets the value of the specified column in its native format given the column name.

– **Parameters**

- * *name* -

- *RecordsAffected*

```
public int RecordsAffected { get; }
```

Gets the number of rows changed, inserted, or deleted by execution of the DSQL statement.

– **Usage**

- * The RecordsAffected property is not set until all rows are read and you close the FbDataReader.
The value of this property is cumulative. For example, if two records are inserted in batch mode, the value of RecordsAffected will be two. IsClosed and RecordsAffected are the only properties that you can call after the FbDataReader is closed.

- *Close*

```
public void Close( )
```

Closes the FbDataReader object.

- **Usage**

- * You must explicitly call the Close method when you are through using the the FbDataReader to use the associated FbConnection for any other purpose.

The Close method fills in the values for output parameters, return values and RecordsAffected, increasing the amount of time it takes to close a FbDataReader that was used to process a large or complicated query. In cases where the return values and the number of records affected by a query are not significant, the amount of time it takes to close the FbDataReader can be reduced by calling the Cancel method of the associated FbCommand object before calling the Close method.

- *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

- **Parameters**

- * requestedType -

- *Dispose*

```
public void Dispose( )
```

Releases the unmanaged resources used by the FbDataReader and optionally releases the managed resources.

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

- **Parameters**

- * obj -

- *Finalize*

```
protected void Finalize( )
```

- *GetBoolean*

```
public bool GetBoolean( )
```

Gets the value of the specified column as a Boolean.

- **Usage**

- * Call IsDBNull to check for null values before calling this method.

- **Parameters**

- * *i* -

- *GetByte*

```
public byte GetByte( )
```

Gets the value of the specified column as a byte.

- **Parameters**

- * *i* -

- *GetBytes*

```
public long GetBytes( )
```

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

- **Usage**

- * GetBytes returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetBytes has already been used to obtain bytes from the field. This may be the case, for example, if the FbDataReader is reading a large data structure into a buffer. For more information, see the SequentialAccess setting for CommandBehavior.

- If you pass a buffer that is a null reference, GetBytes returns the length of the field in bytes.

- No conversions are performed, therefore the data retrieved must already be a byte array.

- **Parameters**

```

* i -
* dataIndex -
* buffer -
* bufferSize -
* length -

```

- *GetChar*

```
public char GetChar( )
```

Gets the value of the specified column as a character.

- **Parameters**

```
* i -
```

- *GetChars*

```
public long GetChars( )
```

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

- **Usage**

- * GetChars returns the number of available characters in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetChars has already been used to obtain characters from the field. This may be the case, for example, if the FbDataReader is reading a large data structure into a buffer. For more information, see the SequentialAccess setting for CommandBehavior.

If you pass a buffer that is a null reference, GetBytes returns the length of the field in characters.

No conversions are performed, therefore the data retrieved must already be a character array.

- **Parameters**

```

* i -
* dataIndex -
* buffer -
* bufferSize -
* length -

```

- *GetData*

```
public System.Data.IDataReader GetData( )
```

Not currently supported.

– **Parameters**

* i -

• *GetDataTypeName*

```
public string GetDataTypeName( )
```

Gets the name of the source data type.

– **Parameters**

* i -

• *GetDateTime*

```
public System.DateTime GetDateTime( )
```

Gets the value of the specified column as a DateTime object.

– **Parameters**

* i -

• *GetDecimal*

```
public decimal GetDecimal( )
```

Gets the value of the specified column as a Decimal object.

– **Parameters**

* i -

• *GetDouble*

```
public double GetDouble( )
```

Gets the value of the specified column as a double-precision floating point number.

– **Parameters**

* i -

• *GetFieldType*

```
public System.Type GetFieldType( )
```

Gets the Type that is the data type of the object.

- **Parameters**

- * *i* -

- *GetFloat*

```
public float GetFloat( )
```

Gets the value of the specified column as a single-precision floating-point number.

- **Parameters**

- * *i* -

- *GetGuid*

```
public System.Guid GetGuid( )
```

Gets the value of the specified column as a globally-unique identifier (GUID).

- **Parameters**

- * *i* -

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetInt16*

```
public System.Int16 GetInt16( )
```

Gets the value of the specified column as a 16-bit signed integer.

- **Parameters**

- * *i* -

- *GetInt32*

```
public int GetInt32( )
```

Gets the value of the specified column as a 32-bit signed integer.

- **Parameters**

* i -

- *GetInt64*

```
public long GetInt64( )
```

Gets the value of the specified column as a 64-bit signed integer.

– **Parameters**

* i -

- *GetLifetimeService*

```
public object GetLifetimeService( )
```

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetName*

```
public string GetName( )
```

Gets the name of the specified column.

– **Parameters**

* i -

- *GetOrdinal*

```
public int GetOrdinal( )
```

Gets the column ordinal, given the name of the column.

– **Parameters**

* name -

- *GetSchemaTable*

```
public System.Data.DataTable GetSchemaTable( )
```

Returns a DataTable that describes the column metadata of the FbDataReader.

– **Usage**

- * DataReader ColumnDescriptionColumnName The name of the column; this might not be unique. If this cannot be determined, a null value is returned. This name always reflects the most recent renaming of the column in the current view or command text.
- ColumnOrdinal The ordinal of the column. This is zero for the bookmark column of the row, if any. Other columns are numbered starting with one. This column cannot contain a null value.
- ColumnSize The maximum possible length of a value in the column. For columns that use a fixed-length data type, this is the size of the data type.
- NumericPrecision If ProviderType is a numeric data type, this is the maximum precision of the column. The precision depends on the definition of the column. If ProviderType is not a numeric data type, this is a null value.
- NumericScale If ProviderType is DECIMAL or NUMERIC, the number of digits to the right of the decimal point. Otherwise, this is a null value.
- DataTypeMaps to the .Net Framework type of the column.ProviderType The indicator of the column's data type. If the data type of the column varies from row to row, this must be Object. This column cannot contain a null value.
- IsLong Set if the column contains a Binary Long Object (BLOB) that contains very long data. The definition of very long data is provider-specific.
- AllowDBNull Set if the consumer can set the column to a null value or if the provider cannot determine whether or not the consumer can set the column to a null value. Otherwise, not set. A column may contain null values, even if it cannot be set to a null value.
- IsReadOnlytrue if the column can be modified; otherwise false.
- IsRowVersion Set if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identity the row.
- IsUniquetrue: No two rows in the base table-the table returned in BaseTableName-can have the same value in this column. IsUnique is guaranteed to be true if the column constitutes a key by itself or if there is a constraint of type UNIQUE that applies only to this column. false: The column can contain duplicate values in the base table.The default of this column is false.
- IsKeytrue: The column is one of a set of columns in the rowset that, taken together, uniquely identify the row. The set of columns with IsKey set to true must uniquely identify a row in the rowset. There is no requirement that this set of columns is a minimal set of columns. This set of columns may be generated from a base table primary key, a unique constraint or a unique index. false: The column is not required to uniquely identify the row.
- IsAutoIncrementtrue: The column assigns values to new rows in fixed increments. false: The column does not assign values to new rows in fixed increments.The default of this column is false.

IsAliased true if the column name is an alias; otherwise false.

IsExpression true if the column is an expression; otherwise false.

BaseSchemaName The name of the schema in the data store that contains the column. A null value if the base schema name cannot be determined. The default of this column is a null value.

BaseCatalogName The name of the catalog in the data store that contains the column. NULL if the base catalog name cannot be determined. The default of this column is a null value.

BaseTableName The name of the table or view in the data store that contains the column. A null value if the base table name cannot be determined. The default of this column is a null value.

BaseColumnName The name of the column in the data store. This might be different than the column name returned in the ColumnName column if an alias was used. A null value if the base column name cannot be determined or if the rowset column is derived, but not identical to, a column in the data store. The default of this column is a null value.

- *GetString*

```
public string GetString( )
```

Gets the value of the specified column as a string.

- **Parameters**

- * *i* -

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *GetValue*

```
public object GetValue( )
```

Gets the value of the column at the specified ordinal in its native format.

- **Usage**

- * This method returns DBNull for null database columns.

- **Parameters**

- * *i* -

- *GetValues*

```
public int GetValues( )
```

Gets all the attribute columns in the current row.

- **Parameters**

- * values -

- *IEnumerable.GetEnumerator*

```
private System.Collections.IEnumerator  
IEnumerable.GetEnumerator( )
```

- *InitializeLifetimeService*

```
public object InitializeLifetimeService( )
```

Obtains a lifetime service object to control the lifetime policy for this instance.

- *IsDBNull*

```
public bool IsDBNull( )
```

Gets a value indicating whether the column contains non-existent or missing values.

- **Parameters**

- * i -

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *NextResult*

```
public bool NextResult( )
```

Advances the data reader to the next result, when reading the results of batch DSQL statements.

- **Usage**

- * Used to process multiple results, which can be generated by executing batch DSQL statements.
By default, the data reader is positioned on the first result.

- *Read*

```
public bool Read( )
```

Advances the FbDataReader to the next record.

- **Usage**

- * The default position of the FbDataReader is prior to the first record. Therefore, you must call Read to begin accessing any data. Only one FbDataReader per associated FbConnection may be open at a time, and any attempt to open another will fail until the first one is closed. Similarly, while the FbDataReader is in use, the associated FbConnection is busy serving it until you call Close.

- *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.6 CLASS FbError

Collects information relevant to a warning or error returned by Firebird. This class cannot be inherited. This class is created by the Firebird .NET Data Provider when an error occurs. An instance of FbError is created and managed by the FbException class.

DECLARATION

```
public class FbError
    : Object
```

PROPERTIES

- *Class*

```
public byte Class { get; }
```

Gets the severity level of the error returned from Firebird.

- *LineNumber*

```
public int LineNumber { get; }
```

Bets the line number within the DSQL command batch or stored procedure that contains the error.

– **Usage**

- * Line numbering starts at 1. If the value is 0, the line number is not applicable.

- *Message*

```
public string Message { get; }
```

Gets the text describing the error.

- *Number*

```
public int Number { get; }
```

Gets a number that identifies the type of error.

METHODS

• *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

• *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

• *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

• *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

• *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

• *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.7 CLASS FbErrorCollection

Collects all errors generated by the Firebird .NET Data Provider. This class cannot be inherited. This class is created by FbException to collect instances of the FbError class. FbErrorCollection always contains at least one instance of the FbError class.

DECLARATION

```
public class FbErrorCollection
    : Object
```

PROPERTIES

- *Count*

```
public int Count { get; }
```

Gets the number of FbError objects in the collection.

- *Item*

```
public FirebirdSql.Data.Firebird.FbError Item { get; set; }
```

Gets or sets the FbError with the specified error Message.

– Parameters

* errorMessage -

- *Item*

```
public FirebirdSql.Data.Firebird.FbError Item { get; set; }
```

– Parameters

* errorIndex -

- *System.Collections.ICollection.IsSynchronized*

```
private bool System.Collections.ICollection.IsSynchronized { get;
}
```

- *System.Collections.ICollection.SyncRoot*

```
private object System.Collections.ICollection.SyncRoot { get; }
```

CONSTRUCTORS

- *.ctor*

```
public FbErrorCollection( )
```

Initializes a new instance of the FbErrorCollection class.

METHODS

- *CopyTo*

```
public void CopyTo( )
```

Copies the elements of the FbErrorCollection collection into an Array, starting at the given index within the Array.

– **Parameters**

* **array** -

* **index** -

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* **obj** -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *IEnumerable.GetEnumerator*

```
private System.Collections.IEnumerator  
IEnumerable.GetEnumerator( )
```

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.8 CLASS FbException

The exception that is thrown when Firebird Server returns a warning or error. This class cannot be inherited. This class is created whenever the Firebird Server .NET Data Provider encounters an error generated from the server. (Client side errors are thrown as standard URT exceptions.) FbException always contains at least one instance of FbError.

DECLARATION

```
public class FbException
    : SystemException
```

PROPERTIES

- *ErrorCode*

```
public int ErrorCode { get; }
```

Gets a value representing the Firebird error code

- *Errors*

```
public FirebirdSql.Data.Firebird.FbErrorCollection Errors { get;
}
```

Gets a collection of one or more FbError objects that give detailed information about exceptions generated by the Firebird .NET Data Provider.

- *HelpLink*

```
public string HelpLink { get; set; }
```

Gets or sets a link to the help file associated with this exception.

- *HResult*

```
protected int HResult { get; set; }
```

Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

- *InnerException*

```
public System.Exception InnerException { get; }
```

Gets the instance that caused the current exception.

- *Message*

```
public string Message { get; }
```

Gets a message that describes the current exception.

- *Source*

```
public string Source { get; set; }
```

Gets or sets the name of the application or the object that causes the error.

- *StackTrace*

```
public string StackTrace { get; }
```

Gets a string representation of the frames on the call stack at the time the current exception was thrown.

- *TargetSite*

```
public System.Reflection.MethodBase TargetSite { get; }
```

Gets the method that throws the current exception.

METHODS

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetBaseException*

```
public System.Exception GetBaseException( )
```

When overridden in a derived class, returns the that is the root cause of one or more subsequent exceptions.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetObjectData*

```
public void GetObjectData( )
```

When overridden in a derived class, sets the with information about the exception.

– **Parameters**

- * *info* -
- * *context* -

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *ToString*

```
public string ToString( )
```

Creates and returns a string representation of the current exception.

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.9 CLASS FbInfoMessageEventArgs

Provides data for the InfoMessage event. This class cannot be inherited. The InfoMessage event contains an FbErrorCollection collection with warnings sent from the Firebird Server.

DECLARATION

```
public class FbInfoMessageEventArgs
    : EventArgs
```

PROPERTIES

- *Errors*

```
public FirebirdSql.Data.Firebird.FbErrorCollection Errors { get; }
}
```

Gets the collection of warnings sent from the Firebird Server.

- *Message*

```
public string Message { get; }
```

Gets a value representing the complete error message sent from the Firebird Server.

METHODS

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.10 CLASS FbParameter

Represents a parameter to a FbCommand, and optionally, its mapping to DataSet columns. This class cannot be inherited. Parameter names are not case sensitive.

DECLARATION

```
public class FbParameter
    : MarshalByRefObject
```

PROPERTIES

- *DbType*

```
public System.Data.DbType DbType { get; set; }
```

Gets or sets the DbType of the parameter.

- *Direction*

```
public System.Data.ParameterDirection Direction { get; set; }
```

Gets or sets the DbType of the parameter.

- *IsNullable*

```
public bool IsNullable { get; set; }
```

Gets or sets a value indicating whether the parameter accepts null values.

- *ParameterName*

```
public string ParameterName { get; set; }
```

Gets or sets the name of the FbParameter.

- *Precision*

```
public byte Precision { get; set; }
```

Gets or sets the maximum number of digits used to represent the Value property.

- *Scale*

```
public byte Scale { get; set; }
```

Gets or sets the number of decimal places to which Value is resolved.

- *Size*

```
public int Size { get; set; }
```

Gets or sets the maximum size, in bytes, of the data within the column.

- *SourceColumn*

```
public string SourceColumn { get; set; }
```

Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the Value.

- *SourceVersion*

```
public System.Data.DataRowVersion SourceVersion { get; set; }  
}
```

Gets or sets the DataRowVersion to use when loading Value.

- *Value*

```
public object Value { get; set; }
```

Gets or sets the value of the parameter.

CONSTRUCTORS

- *.ctor*

```
public FbParameter( )
```

Initializes a new instance of the FbParameter class.

- *.ctor*

```
public FbParameter( )
```

Initializes a new instance of the FbParameter class.

– Parameters

* *parameterName* -

* *fbType* -

- *.ctor*

```
public FbParameter( )
```

Initializes a new instance of the FbParameter class.

- **Parameters**

- * `parameterName` -
- * `paramValue` -

- *.ctor*

```
public FbParameter( )
```

Initializes a new instance of the FbParameter class.

- **Parameters**

- * `parameterName` -
- * `fbType` -
- * `sourceColumn` -

- *.ctor*

```
public FbParameter( )
```

Initializes a new instance of the FbParameter class.

- **Parameters**

- * `parameterName` -
- * `fbType` -
- * `size` -
- * `sourceColumn` -

METHODS

- *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

- **Parameters**

- * `requestedType` -

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetLifetimeService*

```
public object GetLifetimeService( )
```

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *ICloneable.Clone*

```
private object ICloneable.Clone( )
```

- *InitializeLifetimeService*

```
public object InitializeLifetimeService( )
```

Obtains a lifetime service object to control the lifetime policy for this instance.

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *ToString*

public string ToString()

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.11 CLASS FbParameterCollection

Collects all parameters relevant to a FbCommand as well as their respective mappings to DataSet columns. This class cannot be inherited. The number of the parameters in the collection must be equal to the number of parameter placeholders within the command text, or Firebird raises an error.

DECLARATION

```
public class FbParameterCollection
    : MarshalByRefObject
```

PROPERTIES

- *Count*

```
public int Count { get; }
```

Gets the number of FbParameter objects in the collection.

- *Item*

```
public FirebirdSql.Data.Firebird.FbParameter Item { get; set;
}
```

Gets or sets the FbParameter with the specified name.

– **Parameters**

* `parameterName` -

- *Item*

```
public FirebirdSql.Data.Firebird.FbParameter Item { get; set;
}
```

Gets or sets the FbParameter at the specified index.

– **Parameters**

* `parameterIndex` -

- *System.Collections.ICollection.IsSynchronized*

```
private bool System.Collections.ICollection.IsSynchronized { get;
}
```

- *System.Collections.ICollection.SyncRoot*

```
private object System.Collections.ICollection.SyncRoot { get; }
```

- *System.Collections.IList.IsFixedSize*

```
private bool System.Collections.IList.IsFixedSize { get; }
```

- *System.Collections.IList.IsReadOnly*

```
private bool System.Collections.IList.IsReadOnly { get; }
```

- *System.Collections.IList.Item*

```
private object System.Collections.IList.Item { get; set; }
```

– Parameters

* *parameterIndex* -

- *System.Data.IDataParameterCollection.Item*

```
private object System.Data.IDataParameterCollection.Item {
get; set; }
```

– Parameters

* *parameterName* -

CONSTRUCTORS

- *.ctor*

```
public FbParameterCollection( )
```

Initializes a new instance of the FbParameterCollection class.

METHODS

- *Add*

```
public int Add( )
```

Adds the specified FbParameter object to the collection.

– **Parameters**

* value -

• *Add*

```
public FirebirdSql.Data.Firebird.FbParameter Add( )
```

Adds the specified FbParameter object to the collection.

– **Parameters**

* param -

• *Add*

```
public FirebirdSql.Data.Firebird.FbParameter Add( )
```

Adds the specified FbParameter object to the collection.

– **Parameters**

* parameterName -

* type -

• *Add*

```
public FirebirdSql.Data.Firebird.FbParameter Add( )
```

Adds a FbParameter to the FbParameterCollection with the parameter name and the data type.

– **Parameters**

* parameterName -

* value -

• *Add*

```
public FirebirdSql.Data.Firebird.FbParameter Add( )
```

Adds a FbParameter to the FbParameterCollection with the parameter name the data type and the source column.

– **Parameters**

* parameterName -

* fbType -

* sourceColumn -

- *Add*

```
public FirebirdSql.Data.Firebird.FbParameter Add( )
```

Adds a FbParameter to the FbParameterCollection with the parameter name, the data type, the parameter size (column width), and the source column name.

- **Parameters**

- * `parameterName` -
 - * `fbType` -
 - * `size` -
 - * `sourceColumn` -

- *Clear*

```
public void Clear( )
```

Removes all items from the collection.

- *Contains*

```
public bool Contains( )
```

Indicates whether a FbParameter exists in the collection.

- **Parameters**

- * `value` -

- *Contains*

```
public bool Contains( )
```

Indicates whether a FbParameter exists in the collection.

- **Parameters**

- * `parameterName` -

- *CopyTo*

```
public void CopyTo( )
```

Copies FbParameterCollection to the specified array.

- **Parameters**

- * array -
- * index -

- *CreateObjRef*

public System.Runtime.Remoting.ObjRef CreateObjRef()

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

– **Parameters**

- * requestedType -

- *Equals*

public bool Equals()

Determines whether the specified is equal to the current .

– **Parameters**

- * obj -

- *Finalize*

protected void Finalize()

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetEnumerator*

public System.Collections.IEnumerator GetEnumerator()

This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.

- *GetHashCode*

public int GetHashCode()

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetLifetimeService*

public object GetLifetimeService()

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *IndexOf*

```
public int IndexOf( )
```

Gets the location of the FbParameter in the collection with a specific parameter name.

– **Parameters**

* value -

- *IndexOf*

```
public int IndexOf( )
```

Gets the location of the FbParameter in the collection with a specific parameter name.

– **Parameters**

* parameterName -

- *InitializeLifetimeService*

```
public object InitializeLifetimeService( )
```

Obtains a lifetime service object to control the lifetime policy for this instance.

- *Insert*

```
public void Insert( )
```

Gets the location of the FbParameter in the collection with a specific parameter name.

– **Parameters**

* index -

* value -

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *Remove*

public void Remove()

Removes the specified FbParameter from the collection.

– **Parameters**

* value -

- *RemoveAt*

public void RemoveAt()

Removes the specified FbParameter from the collection using a specific index.

– **Parameters**

* index -

- *RemoveAt*

public void RemoveAt()

Removes the specified FbParameter from the collection.

– **Parameters**

* parameterName -

- *ToString*

public string ToString()

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.12 CLASS FbPermission

Enables the Firebird .NET Data Provider to ensure that a user has a security level adequate to access an Firebird data source. This class cannot be inherited. The `IsUnrestricted` property takes precedence over the `AllowBlankPassword` property. Therefore, if you set `AllowBlankPassword` to false, you must also set `IsUnrestricted` to false to prevent a user from making a connection using a blank password.

DECLARATION

```
public class FbPermission
: DBDataPermission
```

PROPERTIES

- *AllowBlankPassword*

```
public bool AllowBlankPassword { get; set; }
```

Gets a value indicating whether a blank password is allowed.

CONSTRUCTORS

- *.ctor*

```
public FbPermission( )
```

Initializes a new instance of the FbPermission class.

- *.ctor*

```
public FbPermission( )
```

Initializes a new instance of the FbPermission class with one of the `PermissionState` values.

– **Parameters**

* `state` -

- *.ctor*

```
public FbPermission( )
```

Initializes a new instance of the FbPermission class.

– **Usage**

- * The PermissionState enumeration takes precedence over the AllowBlankPassword property. Therefore, if you set AllowBlankPassword to false, you must also set PermissionState to None to prevent a user from making a connection using a blank password.

– **Parameters**

- * `state` -
- * `allowBlankPassword` -

METHODS

- *Add*

```
public void Add( )
```

Adds access for the specified connection string to the existing state of the permission.

– **Parameters**

- * `connectionString` -
- * `restrictions` -
- * `behavior` -

- *Assert*

```
public void Assert( )
```

Declares that the calling code can access the resource protected by a permission demand through the code that calls this method, even if callers higher in the stack have not been granted permission to access the resource.

Using

can create security vulnerabilities.

- *Clear*

```
protected void Clear( )
```

Removes all permissions that were previous added using the method.

- *Copy*

```
public System.Security.IPermission Copy( )
```

Creates and returns an identical copy of the current permission object.

- *CreateInstance*

```
protected System.Data.Common.DBDataPermission
CreateInstance( )
```

Creates a new instance of a DataPermission class.

- *Demand*

```
public void Demand( )
```

Forces a at run time if all callers higher in the call stack have not been granted the permission specified by the current instance.

- *Deny*

```
public void Deny( )
```

Prevents callers higher in the call stack from using the code that calls this method to access the resource specified by the current instance.

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

* obj -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *FromXml*

```
public void FromXml( )
```

Reconstructs a security object with a specified state from an XML encoding.

– **Usage**

* Custom code that extends security objects needs to implement the ToXml and FromXml methods to make the objects security-encodable.

– **Parameters**

* securityElement -

• *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

• *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

• *Intersect*

```
public System.Security.IPermission Intersect( )
```

Returns a new permission object representing the intersection of the current permission object and the specified permission object.

– **Usage**

* The intersection of two permissions is a permission that describes the set of operations they both describe in common. Only a demand that passes both original permissions will pass the intersection.

– **Parameters**

* target -

• *IsSubsetOf*

```
public bool IsSubsetOf( )
```

Returns a value indicating whether the current permission object is a subset of the specified permission object.

– **Usage**

* The current permission object is a subset of the specified permission object if the current permission object specifies a set of operations that is wholly contained by the specified permission object. For example, a permission that represents access to C: example.txt is a subset of a permission that represents access to C: . If this method returns true, the current permission object represents no more access to the protected resource than does the specified permission object.

– **Parameters**

* **target** -

- *IsUnrestricted*

public bool IsUnrestricted()

Returns a value indicating whether the permission can be represented as unrestricted without any knowledge of the permission semantics.

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *PermitOnly*

public void PermitOnly()

Prevents callers higher in the call stack from using the code that calls this method to access all resources except for the resource specified by the current instance.

- *ToString*

public string ToString()

Creates and returns a string representation of the current permission object.

– **Usage**

* This method is useful in debugging when you need to display the permission as a string.

- *ToXml*

public System.Security.SecurityElement ToXml()

Creates an XML encoding of the security object and its current state.

– **Usage**

* Custom code that extends security objects needs to implement the ToXml and FromXml methods to make the objects security-encodable.

- *Union*

public System.Security.IPermission Union()

Returns a new permission object that is the union of the current and specified permission objects.

– **Usage**

- * The result of a call to Union is a permission that represents all the operations represented by both the current permission object and the specified permission object. Any demand that passes either permission passes their union.

– **Parameters**

- * **target** -

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.13 CLASS FbPermissionAttribute

Associates a security action with a custom security attribute.

DECLARATION

```
public class FbPermissionAttribute
: DBDataPermissionAttribute
```

PROPERTIES

- *Action*

```
public System.Security.Permissions.SecurityAction Action { get;
set; }
```

Gets or sets a security action.

- *AllowBlankPassword*

```
public bool AllowBlankPassword { get; set; }
```

Gets or sets a value indicating whether a blank password is allowed.

- *ConnectionString*

```
public string ConnectionString { get; set; }
```

Gets or sets a permitted connection string.

- *KeyRestrictionBehavior*

```
public System.Data.KeyRestrictionBehavior
KeyRestrictionBehavior { get; set; }
```

Identifies whether the list of connection string parameters identified by the property are the only additional connection string parameters allowed.

- *KeyRestrictions*

```
public string KeyRestrictions { get; set; }
```

Gets or sets connection string parameters that are allowed or disallowed.

- *TypeId*

```
public object TypeId { get; }
```

When implemented in a derived class, gets a unique identifier for this .

- *Unrestricted*

```
public bool Unrestricted { get; set; }
```

Gets or sets a value indicating whether full (unrestricted) permission to the resource protected by the attribute is declared.

CONSTRUCTORS

- *.ctor*

```
public FbPermissionAttribute( )
```

Initializes a new instance of the FbPermissionAttribute class with one of the SecurityAction values.

– **Parameters**

* *action* -

METHODS

- *CreatePermission*

```
public System.Security.IPermission CreatePermission( )
```

Returns an FbPermission object that is configured according to the attribute properties.

- *Equals*

```
public bool Equals( )
```

– **Parameters**

* *obj* -

- *Finalize*

protected void Finalize()

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

public int GetHashCode()

Returns the hash code for this instance.

- *GetType*

public System.Type GetType()

Gets the of the current instance.

- *IsDefaultAttribute*

public bool IsDefaultAttribute()

When overridden in a derived class, returns an indication whether the value of this instance is the default value for the derived class.

- *Match*

public bool Match()

When overridden in a derived class, returns a value indicating whether this instance equals a specified object.

- **Parameters**

- * obj -

- *MemberwiseClone*

protected object MemberwiseClone()

Creates a shallow copy of the current .

- *ToString*

public string ToString()

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: `FirebirdSql.Data.Firebird`

1.2.14 CLASS FbRowUpdatedEventArgs

Provides data for the RowUpdated event. This class cannot be inherited.

DECLARATION

```
public class FbRowUpdatedEventArgs
    : RowUpdatedEventArgs
```

PROPERTIES

- *Command*

```
public FirebirdSql.Data.Firebird.FbCommand Command { get; }
```

Gets the FbCommand executed when Update is called.

- *Errors*

```
public System.Exception Errors { get; set; }
```

Gets any errors generated by the .NET Framework data provider when the was executed.

- *RecordsAffected*

```
public int RecordsAffected { get; }
```

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

- *Row*

```
public System.Data.DataRow Row { get; }
```

Gets the sent through an .

- *StatementType*

```
public System.Data.StatementType StatementType { get; }
```

Gets the type of SQL statement executed.

- *Status*

```
public System.Data.UpdateStatus Status { get; set; }
```

Gets the of the property.

- *TableMapping*

```
public System.Data.Common.DataTableMapping TableMapping {  
get; }
```

Gets the sent through an .

CONSTRUCTORS

- *.ctor*

```
public FbRowUpdatedEventArgs( )
```

Initializes a new instance of the FbRowUpdatedEventArgs class.

– **Parameters**

- * row -
- * command -
- * statementType -
- * tableMapping -

METHODS

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

- * obj -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.15 CLASS FbRowUpdatingEventArgs

Provides data for the RowUpdating event. This class cannot be inherited.

DECLARATION

```
public class FbRowUpdatingEventArgs
    : RowUpdatingEventArgs
```

PROPERTIES

- *Command*

```
public FirebirdSql.Data.Firebird.FbCommand Command { get;
set; }
```

Gets or sets the FbCommand to execute when Update is called.

- *Errors*

```
public System.Exception Errors { get; set; }
```

Gets any errors generated by the .NET Framework data provider when the executes.

- *Row*

```
public System.Data.DataRow Row { get; }
```

Gets the to send through an .

- *StatementType*

```
public System.Data.StatementType StatementType { get; }
```

Gets the type of SQL statement to execute.

- *Status*

```
public System.Data.UpdateStatus Status { get; set; }
```

Gets the of the property.

- *TableMapping*

```
public System.Data.Common.DataTableMapping TableMapping {  
    get; }  
}
```

Gets the to send through the .

CONSTRUCTORS

- *.ctor*

```
public FbRowUpdatingEventArgs( )
```

Initializes a new instance of the FbRowUpdatingEventArgs class.

– **Parameters**

- * *row* -
- * *command* -
- * *statementType* -
- * *tableMapping* -

METHODS

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

– **Parameters**

- * *obj* -

- *Finalize*

```
protected void Finalize( )
```

Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: FirebirdSql.Data.Firebird

1.2.16 CLASS FbTransaction

Represents a Firebird transaction to be made in a Firebird database. This class cannot be inherited.

DECLARATION

```
public class FbTransaction
    : MarshalByRefObject
```

PROPERTIES

- *Connection*

```
public FirebirdSql.Data.Firebird.FbConnection Connection {
    get; set; }
```

Gets the FbConnection object associated with the transaction, or a null reference if the transaction is no longer valid.

– Usage

- * A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by BeginTransaction.

- *IsolationLevel*

```
public System.Data.IsolationLevel IsolationLevel { get; set; }
```

Specifies the IsolationLevel for this transaction.

- *System.Data.IDbTransaction.Connection*

```
private System.Data.IDbConnection
    System.Data.IDbTransaction.Connection { get; }
```

METHODS

- *Commit*

```
public void Commit( )
```

Commits the database transaction.

- *CommitRetaining*

```
public void CommitRetaining( )
```

Commits the database transaction and retains the transaction context after a commit.

- *CreateObjRef*

```
public System.Runtime.Remoting.ObjRef CreateObjRef( )
```

Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

- **Parameters**

- * requestedType -

- *Dispose*

```
public void Dispose( )
```

Releases the unmanaged resources used by the FbTransaction and optionally releases the managed resources.

- *Equals*

```
public bool Equals( )
```

Determines whether the specified is equal to the current .

- **Parameters**

- * obj -

- *Finalize*

```
protected void Finalize( )
```

- *GetHashCode*

```
public int GetHashCode( )
```

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

- *GetLifetimeService*

```
public object GetLifetimeService( )
```

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

- *GetType*

```
public System.Type GetType( )
```

Gets the of the current instance.

- *InitializeLifetimeService*

```
public object InitializeLifetimeService( )
```

Obtains a lifetime service object to control the lifetime policy for this instance.

- *MemberwiseClone*

```
protected object MemberwiseClone( )
```

Creates a shallow copy of the current .

- *Rollback*

```
public void Rollback( )
```

Rolls back a transaction from a pending state.

- *Rollback*

```
public void Rollback( )
```

Rolls back a transaction from a pending state, and specifies the transaction or savepoint name.

– **Parameters**

* savePointName -

– **Example**

```
FbConnection myConnection = new  
FbConnection(connectionString); myConnection.Open();
```

```

FbCommand myCommand = new FbCommand(); FbTransaction
myTrans;
// Start a local transaction myTrans =
myConnection.BeginTransaction(); // Assign transaction
object for a pending local transaction myCommand.Connection
= myConnection; myCommand.Transaction = myTrans;
try { myCommand.CommandText = "INSERT INTO PROJECT(proj_id,
proj_name, product) Values('FBNP', '.Net Provider', 'N/A')";
myCommand.ExecuteNonQuery();
myTrans.Save("SampleTransaction"); myCommand.CommandText =
"INSERT INTO PROJECT(proj_id, proj_name, product)
Values('FBN1', '.Net Provider1.', 'N/A')";
myCommand.ExecuteNonQuery(); myTrans.Commit();
Console.WriteLine("Both records are written to database.");
} catch (Exception e) { try {
myTrans.Rollback("SampleTransaction"); } catch (FbException
ex) { if (myTrans.Connection != null) {
Console.WriteLine("An exception of type " + ex.GetType() + "
was encountered while attempting to roll back the
transaction."); } }
Console.WriteLine("An exception of type " + e.GetType() + "
was encountered while inserting the data.");
Console.WriteLine("Neither record was written to
database."); } finally { myConnection.Close(); }

```

- *RollbackRetaining*

```
public void RollbackRetaining( )
```

Rolls back a transaction from a pending state and retains the transaction context after a commit.

- *Save*

```
public void Save( )
```

Rolls back a transaction from a pending state, and specifies the transaction or savepoint name.

– **Usage**

- * Savepoints offer a mechanism to roll back portions of transactions. You create a savepoint using the Save method, and then later call the Rollback method to roll back to the savepoint instead of rolling back to the start of the transaction.

Savepoints are useful in situations where errors are unlikely to occur. The use of a savepoint to roll back part of a transaction in the case of an infrequent error can be more efficient than having each transaction test to see if an update is valid before making the update. Updates and rollbacks are expensive operations, so savepoints are effective only if the probability of encountering the error is low and the cost of checking the validity of an update beforehand is relatively high.

– **Parameters**

* savePointName -

– **Example**

```
FbConnection myConnection = new
FbConnection(connectionString); myConnection.Open();
FbCommand myCommand = new FbCommand(); FbTransaction
myTrans;
// Start a local transaction myTrans =
myConnection.BeginTransaction(); // Assign transaction
object for a pending local transaction myCommand.Connection
= myConnection; myCommand.Transaction = myTrans;
try { myCommand.CommandText = "INSERT INTO PROJECT(proj_id,
proj_name, product) Values('FBNP', '.Net Provider', 'N/A')";
myCommand.ExecuteNonQuery();
myTrans.Save("SampleTransaction"); myCommand.CommandText =
"INSERT INTO PROJECT(proj_id, proj_name, product)
Values('FBN1', '.Net Provider1.', 'N/A')";
myCommand.ExecuteNonQuery(); myTrans.Commit();
Console.WriteLine("Both records are written to database.");
} catch(Exception e) { try {
myTrans.Rollback("SampleTransaction"); } catch (FbException
ex) { if (myTrans.Connection != null) {
Console.WriteLine("An exception of type " + ex.GetType() + "
was encountered while attempting to roll back the
transaction."); } }
Console.WriteLine("An exception of type " + e.GetType() + "
was encountered while inserting the data.");
Console.WriteLine("Neither record was written to
database."); } finally { myConnection.Close(); }
```

- *ToString*

```
public string ToString( )
```

Returns a that represents the current .

EXTENDED INFORMATION

- Assembly: `FirebirdSql.Data.Firebird`