ADO Driver documentation

Contents

[Component Overview 1](#_Toc128505678)

[Connection Strings 5](#_Toc128505679)

[Working with Change Views 7](#_Toc128505680)

[Events 12](#_Toc128505681)

[Entity FrameworkCore 14](#_Toc128505682)

[Entity UDF 17](#_Toc128505683)

[DDEX 23](#_Toc128505684)

# Component Overview

The core driver name is InterBaseSQL.Data.InterBaseClient.dll. This needs to be added to your project in the references section. If for some reason the client is not registered using GACUTIL (<https://docs.microsoft.com/en-us/dotnet/framework/app-domains/install-assembly-into-gac>) then you can manually locate it with Add Reference | Browse.

The ADO.NET driver does require that the InterBase client be installed on the client machine.

**Common Components – Non-inherited properties and procedures documented. Inherited use the MicroSoft documentation from the parent class.**

**InterBaseSql.Data.Client.Native**

IBClientFactory – This gets an instance to the client library if you need direct access to the API.

GetGDSLibrary – This is a static public method to get access to an instance of either the server or embedded InterBase client. Platform/bitness is determined for you. It returns an interface of IIBClient defined an IIBClient.cs. The methods are documented in the InterBaseAPI pdf.

Use – var ibclient = IBClientFactory.GetGDSLibrary(IBServerType.Default);

**InterBaseSql.Data.InterBaseClient**

IBCommand – This is the DBCommand descendant. This represents an SQL statement or stored procedure to execute against a data source. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbcommand?view=net-5.0>

Constructors – Multiple constructors allowing for optionally setting the command text, connection or transaction.

1. No params - IBCommand()
2. 1 string param - IBCommand(string cmdText)
3. 2 Params, String and IBConnection - IBCommand(string cmdText, IBConnection connection)
4. 3 Params, String, IBConnection, IBTransaction - IBCommand(string cmdText, IBConnection connection, IBTransaction transaction)

CommandPlan – Returns the plan for the currently Prepared SQL. Read only.

Connection – Sets or Gets the IBConnection this command will execute against. Setting is disallowed if there is an active DataReader against this command.

Parameters – Returns the IBParameterCollection for this command.

Transaction – Sets or Gets the IBTransaction this command is to use. If using an implicit transaction the Getter returns NULL. Setting is disallowed if there is an active DataReader .

FetchSize – Sets or Gets the Integer value for how many records to fetch on opening. Setting disallowed if there is an Active DataReader.

Create Parameter – Returns a new IBParameter instance.

ExecuteReader – This accepts 0 or 1 parameter. The optional parameter is a CommandBehavior type. No parameters uses the Default. This executes the command and returns a new IBDataReader against the results.

IBCommandBuilder – DbCommandBuilder descendant. This automatically generates single-table commands used to reconcile changes made to a [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=net-5.0) with the associated database. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbcommandbuilder?view=net-5.0>

Constructor – Accepts 0 or 1 parameters, the optional parameter is an IBDataAdapter to use.

DeriveParameters – This is a static method for determining the parameters of a Stored Procedure only. Accepts the IBCommand as input and when done the IBCommand’s Parameters will reflect the SP’s parameters.

DataAdapter – Gets or Sets the IBDataAdapter to use.

**All of the GetXxxxCommand come in 2 forms. 0 Parameters or an optional Boolean when set true uses the column for parameter names.**

GetInsertCommand – Returns the Insert SQL in the form of an IBCommand.

GetUpdateCommand – Returns the Update SQL in the form of an IBCommand.

GetDeleteCommand – Returns the Delete SQL in the form of an IBCommand.

IBConnection – DbConnection descendant. Defines the core behavior of database connections. Works from a connection string (see below). <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbconnection?view=net-5.0>

Constructors - 0 or 1 parameters. The single parameter version takes a connection string and associates it with the newly created IBConnection.

ClearAllPools – Clears all the pools in the IBConnectionManager.

ClearPool – Accepts a single IBConnection and clears all instances using that IBConnection’s connection string.

ClearPool – Accepts a single string and clears all instances that use that connection string.

Static CreateDatabase – Accepts a connection string and an overwrite Boolean. Creates a new database based on the connection string. If overwrite is false fails if the DB already exists.

Static CreateDatabase – Accepts up to 4 inputs to create a new database

1. A connection string
2. A page size (default 4096)
3. Whether Forced writes (Sync mode) is on (default true)
4. Whether to overwrite an existing DB file (default false)

Static DropDatabase – Drops (deletes) the database identified by the passed connection string.

PacketSize – returns the packet size used in communicating with the server. Read Only.

All Transaction methods have an optional transaction name that can be passed at the end of the parameter list whenusing save points.

BeginTransaction – Starts a Transaction and returns the IBTransaction. Several flavors

1. 0 Parameters – starts a ReadCommitted transaction
2. 1 IsolationLevel param – Starts a transaction based on the isolation level passed.
3. 1 IBTransactionOptions – starts a transaction based on the options passed.

CreateCommand – Returns a new IBCommand to use with this connection.

ChangeDatabase – Accepts a new connection string and switches this connection to use the new information. If an exception is raised in connecting the old string is restored.

IBConnectionStringBuilder – DbConnectionStringBuilder decendant. This can be used to create a connection string from the individual parts. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbconnectionstringbuilder?view=net-5.0>

Details on connection strings properties [below](#ConnectionStrings).

Constructor – Two forms, no parameters or a connection string that will get parsed into its pieces.

IBDataAdapter – DbDataAdapter descendant. This implements a set of functions to provide strong typing, but inherit most of the functionality needed to fully implement a DataAdapter. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbdataadapter?view=net-5.0>

IBDatabaseInfo – Class that given a connection allows for retrieving information on that database.

Properties

IBConnection – The connection component to use to query against.

IscVersion

PageSize

AllocationPages

BaseLevel

DbId

Implementation

NoReserve

OdsVersion

OdsMinorVersion

MaxMemory

CurrentMemory

ForcedWrites

NumBuffers

SweepInterval

ReadOnly

Fetches

Marks

Reads

Writes

BackoutCount

DeleteCount

ExpungeCount

InsertCount

PurgeCount

ReadIdxCount

ReadSeqCount

UpdateCount

DatabaseSizeInPages

ActiveUsers

IBDataReader - DbDataReader descendant. This is a forward only stream of rows from a data source. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbdatareader?view=net-5.0>

IBEnlistmentNotification - IEnlistmentNotification implementation. <https://docs.microsoft.com/en-us/dotnet/api/system.transactions.ienlistmentnotification?view=net-5.0>

IBTransaction – DbTransaction descendant. Transaction component for transaction control. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbtransaction?view=net-5.0>

InterBaseClientFactory – DbProviderFactory descendant. This represents a set of methods for creating instances of a provider's implementation of the data source classes. <https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbproviderfactory?view=net-5.0>

# Connection Strings

The following describes the keys and synonyms for the Connection strings to attach to an InterBase database. Note the first column represents the property in the IBConnectionString class (so won’t have spaces). The second column is the key name (default in bold and alternatives) that you would use when writing the connection string by hand.

|  |  |  |
| --- | --- | --- |
| IBConnectionBuilder Property | Connection string key names (default in bold) | Description |
| Datasource | **Data Source**, server, host, datasource | This is the IP address or the Host name for the location of the server. |
| Port | **Port Number**, port | The port the IB server is listening on. Default 3050. |
| Database | **Initial Catalog**, database | Path to the database or the database alias. |
| UserID | UserID, User, uid, user name, username, **user id** | The user to log in as. |
| Password | **Password**, User password, userpassword | Password to use logging in. |
| Dialect | **dialect** | Dialect to force the connection to be in. Default 3. |
| Role | **role** | The role to log in as. |
| Charset | **Character Set**, Charset | The character set to establish the connection with. Default is none (ANSII) |
| PacketSize | **Packet Size**, PacketSize | Packet size to use when communicating with the server. Default 8192 |
| truncate\_char | **truncate\_char** | Default is false. When true Char datatypes with be right trimmed (iow act more like a VarChar with trailing spaces not important) |
| Pooling | **Pooling** | When true the connection is grabbed from a pool or, if necessary, created and added to the appropriate pool. Default true. |
| ConnectionLifeTime | **Connection Lifetime**, ConnectionLifeTime | When 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. Default 0 |
| ConnectionTimeout | **connection timeout**, timeout, ConnectionTimeout | The time (in seconds) to wait for a connection to open. Default 15. |
| MinPoolSize | **Min pool size**, MinPoolSize | The minimum number of connections allowed in the pool. Default 0 |
| MaxPoolSize | **Max pool size**, MaxPoolSize | The maximum number of connections allowed in the pool. Default 100 |
| FestchSize | **Fetch size**, FestchSize | The maximum number of rows to be fetched in a single call to read into the internal row buffer. Default 200 |
| ServerType | **server type**,  ServerType | The type of server used. Default IBServerType.Default (Server). Other option is IBServerType.Embedded |
| Isolationlevel | **Isolation level**, Isolationlevel | The default Isolation Level for implicit transactions. Default ReadCommitted |
| ReturnRecordsAffected | **Records Affected** | Get the number of rows affected by a command when true. Default true. Note Select statements cannot give an accurate number until all records fetched. |
| Enlist | **Enlist** | If true, enlists the connections in the current transaction. Default true |
| embedded | **clientEmbedded**, embedded | If true indicates to act as if the Server Type is Embedded. Default false |
| DBCachepages | Pagebuffers, page buffers, **cache pages**,  Cachepages | How many cache buffers to use for this session. Default 0 (server side determined) |
| EUAEnabled | **EUAEnabled** | Set true when connecting to an End User Authentication enabled DB |
| InstanceName | **Instance Name** | The string alias version of the IB Instance |
| SEPPassword | **SEP password** | The password for SEP enabled DBs |
| SSL | **SSL** | Boolean if to use SSL connection |
| ServerPublicFile | **ServerPublicFile** | The Server Public File to use |
| ServerPublicPath | **ServerPublicPath** | The Server Public Path |
| ClientCertFile | **ClientCertFile** | The Client Certification File |
| ClientPassPhraseFile | **ClientPassPhraseFile** | The Client Pass Phrase File |
| ClientPassPhrase | **ClientPassPhrase** | The Client Pass Phrase |

For more information on the SSL items read <http://docwiki.embarcadero.com/InterBase/2020/en/Setting_up_OTW_Encryption>

Simple example connecting to the employee database on the local server –

server=localhost;database=c:\embarcadero\InterBase64\Examples\Database\employee.gdb;user=sysdba;password=masterkey

# Working with Change Views

Change views are a powerful feature of InterBase. You can get an overview of change views at <https://docwiki.embarcadero.com/InterBase/2020/en/Change_Views>.

First there is an enum located in the InterBaseSql.Data.Common namespace called IBChangeState. It has the following values –

public enum IBChangeState

{

csSame,

csInsert,

csUpdate,

csDelete,

csTruncate,

csUnknown

}

Accessing change view information from an IBDataReader. The DbValue class now has a read only property called ChangeState that returns an IBChangeState. Instead you access the change state for an element in the row similarly to how you would access the Data itself. There is a GetChangeState(int index) and one that accepts the column name method to get the change state for the current row and index into that row.

Here is just a quick little method used in the ChangeView demo that activates a subscription for a location

private void ActivateSubscription(string Name, string Dest)

{

using (var cmd = Connection.CreateCommand())

{

\_snaptransaction = Connection.BeginTransaction(IsolationLevel.Snapshot);

cmd.Transaction = \_snaptransaction;

cmd.CommandText = "SET SUBSCRIPTION " + Name + " AT '" + Dest + "' ACTIVE";

cmd.ExecuteNonQuery();

}

}

Remember that subscriptions should be in a snapshot transaction.

ActivateSubscription("EMPLOYEE\_ALL", "TESTBED");

using (var cmd = Connection.CreateCommand())

{

cmd.Transaction = \_snaptransaction;

cmd.CommandText = "select \* from employee";

var reader = cmd.ExecuteReader();

IBChangeState[] states = { IBChangeState.csInsert, IBChangeState.csDelete,

IBChangeState.csUpdate, IBChangeState.csTruncate };

while (reader.Read())

{

for (int i = 0; i < reader.FieldCount - 1, i++)

{

if (states.Contains(reader.GetChangeState(i)))

Console.WriteLine("Field " + reader.GetName(i) + " has value " +

reader.GetString(i) + " and a change state of " +

reader.GetChangeState(i).ToString());

}

Console.WriteLine("");

}

}

So all this little snippet does is walks all the returned rows from the query and if it is a change writes out the field name, the value and finally the change state. You can also access it just with the field name like

var cs = reader.GetChangeState("last\_name");

Finally another way to get at the data from an IBDataReader would be to type cast the result from indexer directly from a DbValue (the concrete return type of the indexer)

var d = ((DbValue) reader["last\_name"]).ChangeState;

The above would not be a normal way to access it, there are less verbose ways seen further up to get at the change state for the column.

What about DataTables? DataTable support was a little problematic because all access to the underlying DbValue is gone by the time Fill is finished filling out the data. To get around that there are 5 new procedures for IBDataAdapter that basically mirror the Fill routines. I will go into detail for IBDataTable and IBDataRow in a bit.

public int FillWithChangeState(IBDataTable dataTable)

This just takes an IBDataTable and fills it out with its change states.

public int FillWithChangeState(IBDataTable dataTable, int startRecord, int maxRecords)

This just takes an IBDataTable and fills it out with its change states starting on row startRecord and ending at maxRecords. Like the Fill procedure passing 0 for maxRecords means all. Since InterBase does not support multiple result sets in a single query the DataTable[] parameter from the Fill routine was just turned into an IBDataTable.

public int FillWithChangeState(DataSet dataSet)

This routine will accept a DataSet and it will add a m IBDataTable to it with the name being the return from DbDataAdapter.DefaultSourceTableName.

public int FillWithChangeState(DataSet dataSet, string srcTable)

This routine accepts a DataSet and the name of the table you want to fill. If the table is not an IBDataTable and exception will be raised (in this case you want to call Fill directly like you normally would).

public int FillWithChangeState(DataSet dataSet, int startRecord, int maxRecords, string srcTable)

This routine accepts a DataSet, stating and ending record, and the name of the table you want to fill. If the table is not an IBDataTable and exception will be raised (in this case you want to call Fill directly like you normally would).

**IBDataTable** is a descendant of DataTable but instead of holding rows of DataRow it holds rows of IBDataRow which includes the change state information for each column.

It has an indexer into the row that returns the IBDataRow for that row. So if I want to access row 3 you can access it like

IBDataRow dr = Dt[3];

Or you can write the more verbose

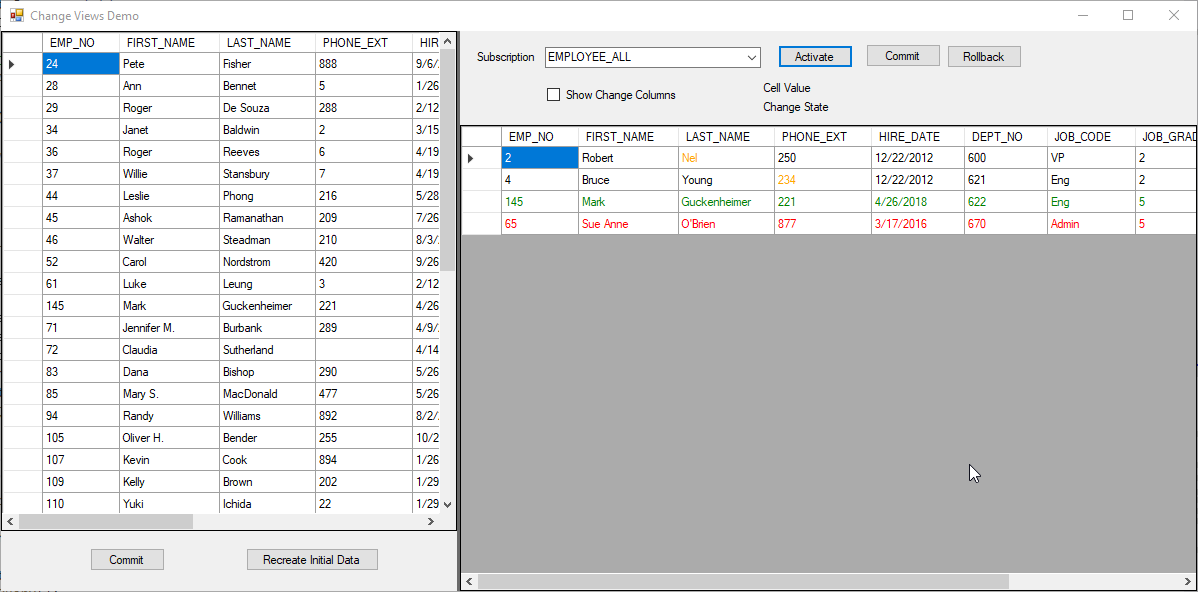
IBDataRow dr = (IBDataRow) dt.Rows[3];

One difference between IBDataTable and DataTable is how you get a new row. With DataTable you call NewRow, With IBDataTable you will call GetNewRow. This will almost never be something you use, but the new rows returned will all have change states of csUnknown as the row did not come from the server.

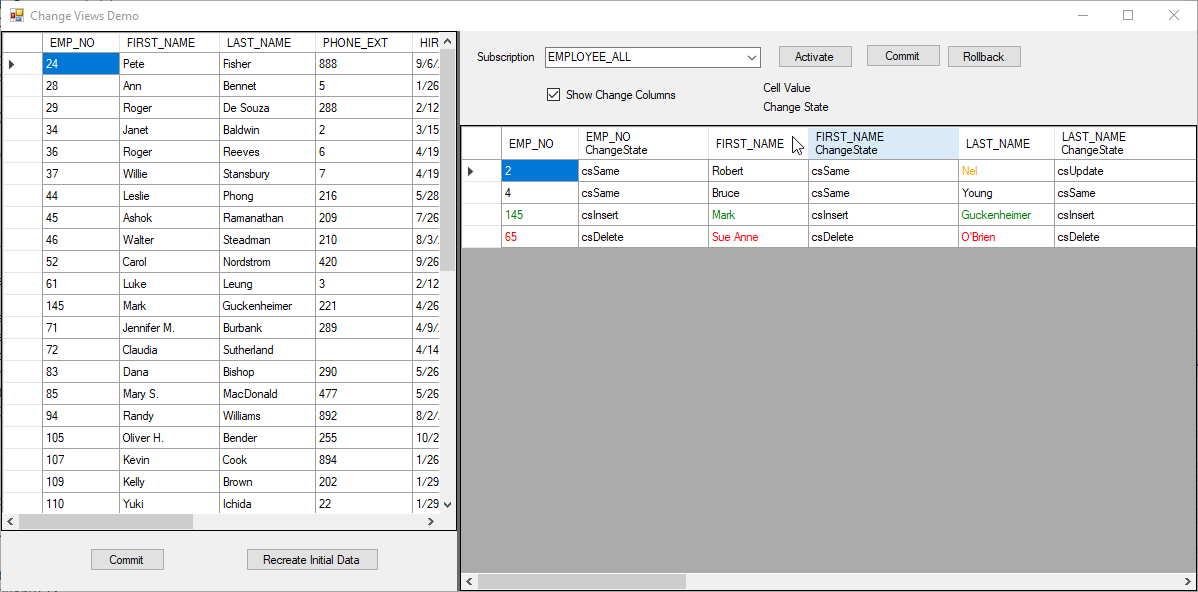
public bool ChangeStateColumns

This property was added that when true it will create 1 read only column name “<orig column name> ChangeState”. When false it removes those columns. If you try to set it to try and there are no columns it will remain false. The new read only columns containing the change state info will be ordinally next to the column it is a change state for.

Normal viewing with the property false



Setting it true



**IBDataRow** is the new DataRow that has the additional information for the ChangeState for a column.

It has three overloaded functions that return an IBChangeState accepting the column index or the column name or the DataColumn itself.

Let’s look at some uses for all this with regards to DataTables (these are snippets from the ChangeViews demo installed when you install the ADO.Driver itself from the installer).

This first is simply filling out the IBDataTable and hooking it up to a grid.

private void btnActivate\_Click(object sender, EventArgs e)

{

ActivateSubscription(comboBox1.SelectedItem.ToString(), "TESTBED");

using (var cmd = \_subConnection.CreateCommand())

{

cmd.Transaction = \_snaptransaction;

cmd.CommandText = "select \* from employee order by emp\_no";

var dataAdapter = new IBDataAdapter(cmd);

IBDataTable table = new IBDataTable();

dataAdapter.FillWithChangeState(table);

bsSubscription.DataSource = table;

dataGridView2.DataSource = bsSubscription;

dataGridView2.AutoGenerateColumns = true;

// Resize the DataGridView columns to fit the newly loaded content.

dataGridView2.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);

}

}

This is almost identical to how you would normally do it, except that we create an instance of the IBDataTable class instead of DataTable and call FillWithChangeState vs just fill. Beyond that is it normal operations.

To show the change columns in a grid like the screen shot from above it looks like

private void chkChangeColumns\_CheckedChanged(object sender, EventArgs e)

{

dataGridView2.DataSource = null;

((IBDataTable)bsSubscription.DataSource).ChangeStateColumns = chkChangeColumns.Checked;

dataGridView2.DataSource = bsSubscription;

dataGridView2.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);

}

Note that the DataSource = null part. Without it the DataGrid will just add the new columns to the end. By setting it null first it will put the change states next to the column they are the change state for.

The demo grabs the change state for a clicked cell and displays information in the labels.

private void dataGridView2\_CellClick(object sender, DataGridViewCellEventArgs e)

{

var table = (IBDataTable) bsSubscription.DataSource;

lblValue.Text = "Cell Value = " + table[e.RowIndex][e.ColumnIndex];

lblState.Text = "Change State = " + table[e.RowIndex].ChangeState(e.ColumnIndex).ToString();

}

table[e.RowIndex].ChangeState(e.ColumnIndex).ToString(); is the important line here. By using the IBDataTables indexer which returns a typed IBDataRow you can call ChangeState directly with no casts. If you call Rows it would be more verbose like

((IBDataRow) table.Rows[e.RowIndex]).ChangeState(e.ColumnIndex).ToString()

Finally I added a little color to the grid based on the columns’ change state in the Demo.

private void dataGridView2\_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)

{

var table = (IBDataTable)bsSubscription.DataSource;

switch (table[e.RowIndex].ChangeState(e.ColumnIndex))

{

case IBChangeState.csUpdate :

dataGridView2.Rows[e.RowIndex].Cells[e.ColumnIndex].Style =

new DataGridViewCellStyle { ForeColor = Color.Orange,

BackColor = Color.White };

break;

case IBChangeState.csDelete:

dataGridView2.Rows[e.RowIndex].Cells[e.ColumnIndex].Style =

new DataGridViewCellStyle { ForeColor = Color.Red,

BackColor = Color.White };

break;

case IBChangeState.csInsert:

dataGridView2.Rows[e.RowIndex].Cells[e.ColumnIndex].Style =

new DataGridViewCellStyle { ForeColor = Color.Green,

BackColor = Color.White };

break;

default:

dataGridView2.Rows[e.RowIndex].Cells[e.ColumnIndex].Style =

dataGridView2.DefaultCellStyle;

break;

}

}

This is again pretty straight forward. Just remember you need to typecast the Bindingsource’s DataSource (which returns just an Object) back to an IBDataTable. From there accessing the ChangeState is the same as before.

Main thing to remember with this is really just when you want to access the change state information from your IBDataTable to use the IBDataTable’s indexer instead of going through .Rows.

# Events

Event support was introduced in 7.12.0. While the original Firebird code had events stubbed out the work was not completed and actually using any of the three Event procedures will just raise an unsupported exception.

At this time, we have decided to just leave those stubbed out and go with an individual class to handle events – IBEvents. The reasoning is that you can have groupings of events and turn on or off each individual group versus the connection only way which would have been monolithic and all on or all off with rebuilding each time.

Due to a strange occurrence with the InterBase client library we have not yet resolved, events are limited to 15.

IBEvents class (nomenclature note – Registered always means that it is listening for the events being fired from the server)

Properties

1. Connection – accepts an IBConnection and is the connection to be used to listen to events
2. Events – a List<string> of all the events this IBEvents object will listen to. Note that while .NET strings are Unicode, the InterBase API is ASCII strings so will be converted before being registered.
3. Registered – bool. True means the events are registered false not registered. This is an alternative way to calling RegisterEvents() or UnregisterEvents() when setting. When getting returns if it is registered or not.

Events

1. EventAlert – accepts a method of the signature void EventAlertHandler(object sender, EventAlertArgs e)

Methods

1. RegisterEvents – Registers the events in the Events property and starts listening for them to fire. Will raise exceptions if the connection is not assigned, connection not open or it is already listening.
2. UnregisterEvents – Stops all the listening threads and unregisters with the server listening to those events.

Example snippets –

Creation – After creating an instance you need to assign the connection property to the connection to be used and then an event handler for when events are fired.

The event fired signature looks like



Finally, to register Events you call RegisterEvents passing a List<String> of the strings you want to watch. No Unicode characters allowed. The list you pass will be converted to ASCII character strings before being registered. You cannot call RegisterEvents if there are currently Registered events. You must first Unregister the events and then call RegisterEvents. It is safe to call UnregisterEvents when nothing is registered.



# Entity FrameworkCore

Some of the implementation of the EntityFrameworkCode, especially when using LINQ, requires the EntityUDF. The script for this UDF is included as well as the dlls for the 32/64 bit platforms. UDF is documented further below. The binaries for the UDF library and the script are included in the main ADO.NET driver install that can be found <https://github.com/Embarcadero/IB.NETDataProvider/releases>.

Also important is the version of the MicroSoft EntityFrameworkCode assemblies. The IB EntityFrameworkCore is based against the Microsoft 6.0.11 so when you use NuGet to pull the MS code, you should pull the 6.0.11. The is especially true when doing scaffolding. Scaffolding for some reason is very sensitive to differences in EntityFrameworkCore versions and will result in errors when running db\_scaffold if you try to mix and match versions.

**Note – The original Firebird code is hard wired to quote identify column, table and alias names. There is not a clean way to switch back to dialect 1 quoting at the level things are happening. To get around this IBSQLGeneratorHelper has a public static variable called Dialect that defaults to 3. If you are working with a Dialect 1 database before you start using the DbContext you need to first make this call –**

            InterBaseSql.EntityFrameworkCore.InterBase.Storage.Internal.IBSqlGenerationHelper.Dialect = 1;

**If you add**

using InterBaseSql.EntityFrameworkCore.InterBase.Storage.Internal;

**is simplifies down to**

    IBSqlGenerationHelper.Dialect = 1;

**This only needs to be called once so in your startup code is a good place.**

Walk through for setting up and using the Entity Framework core.

**Creating New Project**

1. Open Visual Studio
2. On the **File** menu point to **New** and then click **Project**. The New Project dialog box will open.
3. On the left side of the dialog box select **Installed -> Visual C# -> .NET Core**.
4. On the right side of the dialog box select **Console App**.
5. Enter the project name and location if necessary.
6. Click OK. A new project will be created.

**Installing NuGet Packages**

After we created a new project, we need to add the necessary NuGet packages to it.

Now let's install the necessary packages:

1. On the **Tools** menu point to **NuGet Package Manager** and then click **PManage NuGet Packages for Solution**.
2. For Entity Framework Core 6.0, search on Microsoft.EntityFrameworkCore.Tools and for the version select 6.0.11. Do this again for Microsoft.EntityFrameworkCore.Design.
3. Add the InterBase package - InterBaseSQL.EntityFrameworkCore.InterBase

#### Creating a Model From the Database

For Entity Framework Core, creating a model from the database is as easy as entering the [Scaffold-DbContext](https://docs.efproject.net/en/latest/miscellaneous/cli/powershell.html#scaffold-dbcontext) command with a connection string and a provider as parameters. For example, you can run the following command in the Package Manager Console:

Scaffold-DbContext "data source=localhost;initial catalog=Employee;user id=sysdba;password=masterkey" InterBaseSQL.EntityFrameworkCore.InterBase

This assumes an Alias pointing to the Employee database. If you have other tables in your database, you may use additional parameters - -Schemas and -Tables - to filter the list of schemas and/or tables that are added to the model. For example, you can use the following command:

Scaffold-DbContext "data source=localhost;initial catalog=Employee;user id=sysdba;password=masterkey" InterBaseSQL.EntityFrameworkCore.InterBase -Tables employee,department,customer –o Model

This will setup the basic framework. In the EmployeeContext.cs the DBContext is Framework is setup. It will also setup a class for each table (see the EntityCore demo for examples of the end results of each).

Because InterBase does not have functionality to return values modified by triggers it is necessary to get the generated value before inserting a new record. Let’s add an easy method to get at any generator value. This can be done just by extending the EmployeeContext in it’s own C# file like this

using Microsoft.EntityFrameworkCore;

namespace EFCore101

{

partial class EmployeeContext : DbContext

{

public long GetNextSequenceValue(string genName)

{

using (var cmd = Database.GetDbConnection().CreateCommand())

{

Database.GetDbConnection().Open();

cmd.CommandText = "SELECT gen\_id(" + genName + ", 1) from rdb$database";

var obj = cmd.ExecuteScalar();

return (long)obj;

}

}

}

}

Now the Framework is ready to be used for the Employee table. Go back to the main program and we can insert a new record then list all the records like this -

using System;

using ConsoleApp3.Model;

namespace ConsoleApp3

{

class Program

{

static void Main(string[] args)

{

using (var db = new EmployeeContext())

{

// Creating a new Employee and saving it to the database

var newEmp = new Employee()

{

Emp\_No = (short) db.GetNextSequenceValue("EMP\_NO\_GEN"),

First\_Name = "John", Last\_Name = "Doe", Phone\_Ext = "000",

Hire\_Date = DateTime.Now, Dept\_No = "900", Job\_Code = "Sales",

Job\_Grade = 3, Job\_Country = "USA", Salary = 45000

};

db.Employee.Add(newEmp);

var count = db.SaveChanges();

Console.WriteLine("{0} records saved to database", count);

// Retrieving and displaying data

Console.WriteLine();

Console.WriteLine("All Employees in the database:");

foreach (var emp in db.Employee)

{

Console.WriteLine("{0} | {1} | {2}", emp.Full\_Name, emp.Salary, emp.Dept\_No);

}

}

}

}

}

For the most part everything should work just like any other Entity framework example. Current drawback is retrieving changes made on the DB side on Inserts and Updates. To see these you will need to requery the Record. In the above example to get the Full\_Name computed field you would relook it up like this.

# Entity UDF

This is a UDF designed to add some missing internal functions. Since not all were in the shipping IB\_UDF library it was decided to have a self contained library. These can be used just fine outside the EntityFramework Core too. Place the correct bit (32 or 64) dll into the <InterBase>\UDF directory then run the entityUDF.sql script against your Database to use.

Math functions

EF\_Abs –

Input Double

Output Double

Returns the absolute value of the input.

EF\_Ceiling –

Input Double

Output Numeric(18, 0)

Rounds up towards positive infinity.

EF\_Floor –

Input Double

Output Numeric(18, 0)

Rounds down towards negative infinity.

EF\_Round –

Input num Double

scale integer

Output Double

Rounds a number to the nearest integer. If the fractional part is exactly 0.5, rounding is upward for positive numbers and downward for negative numbers. With the optional scale argument, the number can be rounded to powers-of-ten multiples (tens, hundreds, tenths, hundredths, etc.) instead of just integers.

Examples

* EF\_ROUND(123.654, 1) returns 123.7
* EF\_ROUND(8341.7, -3) returns 8000.0
* EF\_ROUND(45.1212, 0) returns 45.0

EF\_Power –

Input x Double

y Double

Ouput Double.

Returns x to the y'th power.

EF\_Truncate –

Input num Double

scale Double

Output Double.

Returns the integer part of a number. With the optional scale argument, the number can be truncated to powers-of-ten multiples (tens, hundreds, tenths, hundredths, etc.) instead of just integers. Examples

* TRUNC(789.2225, 2) returns 789.22
* TRUNC(345.4, -2) returns 300.0
* TRUNC(-163.41, 0) returns -163.0

EF\_BitAnd –

Input x Numeric(18,0)

y Numeric(18, 0)

Ouput Numeric(18, 0).

Returns the result of the bitwise AND operation on the arguments.

EF\_BitOr –

Input x Numeric(18,0)

y Numeric(18, 0)

Ouput Numeric(18, 0).

Returns the result of the bitwise OR operation on the arguments.

EF\_BitXor –

Input x Numeric(18,0)

y Numeric(18, 0)

Ouput Numeric(18, 0).

Returns the result of the bitwise Xor operation on the arguments.

EF\_BitNot –

Input x Numeric(18,0)

Ouput Numeric(18, 0).

Returns the result of the bitwise Not operation on the argument.

String Functions – All string functions have in their SQL definition sizes of 2048. If you need larger or smaller you can always change the definitions up to 32K before running the script.

Also note that because UDF string types are CString (basically a CHAR datatype) they will always return right padded with spaces, but it you cast the result to a VarChar(2048) you will get the unpadded results.

Example:

Select EF\_Reverse(‘ abc ’) from RDB$DATABASE

Returns ‘ cba ‘ (padded out to 2048)

Select Cast(EF\_Reverse(‘ abc ’) as VarChar(2048)) from RDB$DATABASE

Returns ‘ cba ‘

EF\_Reverse –

Input CSTRING(2048)

Output CSTRING(2048).

Returns a string backwards.

Examples

1. EF\_Reverse ('spoonful') -- returns 'lufnoops'
2. EF\_Reverse ('Was it a cat I saw?') -- returns '?was I tac a ti saW'

EF\_Position –

Input substr CSTRING(2048)

str CSTRING(2048)

startIdx integer

Output Integer.

Returns the (1-based) position of the first occurrence of a substring in a host string. With the optional third argument, the search starts at a given offset, disregarding any matches that may occur earlier in the string. If no match is found, the result is 0.

Examples

1. EF\_position ('be', 'To be or not to be', 1) -- returns 4
2. EF\_position ('be', 'To be or not to be', 4) -- returns 4
3. EF\_position ('be', 'To be or not to be', 8) -- returns 17
4. EF\_position ('be', 'To be or not to be', 18) -- returns 0

EF\_Length –

Input CSTRING(2048)

output Integer.

Returns the length of the character string.

EF\_Lower -

Input cstring(2048)

returns cstring(2048).

Returns the ANSI lower case for the string. It should respect things like Ä and return ä. Note the internal Upper function does not do this.

EF\_Trim –

Input type CSTRING(8)

str CSTRING(2048)

Output CString(2048).

Input Types can be ‘BOTH’ or ‘LEADING’ or ‘TRAILING’ (case insensitive). Trims spaces from either before, after or both of the inputed string. Note that the after side will only work if you cast the result to a VarChar as it will get padded by the Server even though properly trimmed.

EF\_Left –

Input Str CSTRING(2048)

len Integer

Output CSTRING(2048).

Returns the leftmost part of the argument string. The number of characters is given in the second argument.

EF\_Right –

Input Str CSTRING(2048)

len Integer

Output CSTRING(2048).

Returns the rightmost part of the argument string. The number of characters is given in the second argument.

EF\_Replace –

Input str CSTRING(2048)

Find CSTRING(2048)

Repl CSTRING(2048)

Output CSTRING(2048)

Replaces all occurrences of a substring in a string.

Example

* EF\_replace ('Billy Wilder', 'il', 'oog') -- returns 'Boogly Woogder'
* EF\_replace ('Billy Wilder', 'il', '') -- returns 'Bly Wder'
* EF\_replace ('Billy Wilder', 'xyz', 'abc') -- returns 'Billy Wilder'
* EF\_replace ('Billy Wilder', '', 'abc') -- returns 'Billy Wilder'

EF\_SubStr –

Input str cstring(2048),

startIdx integer,

length integer

Ouput cstring(2048)

Returns the substring starting at the StartIdx and going to length # of characters.

NewGuid

Ouput CSTRING(16) character set OCTETS

Returns a 16 byte GUID Array. Should go into an OCTET variable

Time Functions

EF\_UTCCurrentTime

Ouput timestamp

Returns the current UTC time.

EF\_DateAdd

Input unit cstring(7)

amount numeric(18, 0)

toDate timestamp

output timestamp

Returns the date added by the type passed. Valid units are - 'YEAR' , 'MONTH' , 'WEEK', 'DAY', 'HOUR', 'MINUTE', 'SECOND'.

Examples

* EF\_dateadd (‘day’, 28, current\_date)
* EF\_dateadd (‘hour’, -6, current\_time)
* EF\_dateadd (‘month’, 9, DateOfConception)
* EF\_dateadd (‘minute’, 90, 'now')

EF\_DateDiff –

Input units cstring(12)

from timestamp

to timestamp

Output numeric(18,0)

Returns the number of years, months, days, hours, minutes, seconds elapsed between two date/time values. Valid units are - 'YEAR' , 'MONTH' , 'WEEK', 'DAY', 'HOUR', 'MINUTE', 'SECOND'

# DDEX

The DDEX driver should be installed by the installer. After that install the ADO.NET driver on your machine (this should make all the machine.config changes necessary). One note, the machine.config must have only one entry for the InterBaseSQL.Data.InterBaseClient.dll –

"InterBaseSql.Data.InterBaseClient" description=".NET Framework Data Provider for InterBase" type="InterBaseSql.Data.InterBaseClient.InterBaseClientFactory, InterBaseSql.Data.InterBaseClient, Version=7.10.2.0, Culture=neutral, PublicKeyToken=73f45bff97b4c31b"/>

The version # should match your installed version.

As of VS2017 the registry for Visual Studio was moved into a private registry. This make the installation a manual process.

Before VS2017, Visual Studio stored its configuration in registry in HKLM and HKCU. Not anymore. The configuration is now stored in a private registry. It’s the privateregistry.bin file in %LOCALAPPDATA% \Microsoft\VisualStudio\15.0\_<something> or 16.0\_<something>. The rest is the same as previous versions.

The installer will install 2 .reg files - one for VS2017 (InterBaseDDEXProviderVS2017.reg) and one for VS2019 (InterBaseDDEXProviderVS2019.reg). You will need to modify these files slightly before they can be imported into the private registry. First is going to be the hive name you will give when loading it into regedit. The files are delivered with names equal to their \_config folder in the private registry. For VS2017 this is 15.0\_7cf12c00 and for VS2019 it is 16.0\_d30585a1. If you change the hive name you will need to update the .reg file to point to that hive. IOW change in the file all instances of [HKEY\_USERS\16.0\_d30585a1 to [HKEY\_USERS\<my hive name>.

The other thing you will need to do is set the CodeBase path. So this

"CodeBase"="%PATH%\\InterBaseSql.VisualStudio.DataTools.dll"

Needs the %PATH% replaced with where you installed the DDEX driver. If you are using the defaults for the installer it will be

"CodeBase"="C:\\Program Files (x86)[\\InterBaseDDEX\\InterBaseSql.VisualStudio.DataTools.dll](file:///\\InterBaseDDEX\\InterBaseSql.VisualStudio.DataTools.dll)"

Save the changes and launch Regedit and load the Visual Studio privateregistry.bin file (for VS 2019 it usually will be here C:\Users\<user>\AppData\Local\Microsoft\VisualStudio\16.0\_d30585a1) using the File > Load Hive... menu and load it into HKEY\_USERS under the name the reg file uses (either the default or your name you went with). Note the VS cannot be running when loading it. So for VS 2019 using the default hive name and default DDEX dll location the reg file will look like

Windows Registry Editor Version 5.00

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataSources\{108473C1-273A-41BF-AE1E-CA515C423B9F}]

@="InterBase Data Source"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataSources\{108473C1-273A-41BF-AE1E-CA515C423B9F}\SupportingProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}]

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}]

@=".NET Framework Data Provider for InterBase"

"DisplayName"="Provider\_DisplayName, InterBaseSql.VisualStudio.DataTools.Properties.Resources"

"ShortDisplayName"="Provider\_ShortDisplayName, InterBaseSql.VisualStudio.DataTools.Properties.Resources"

"Description"="Provider\_Description, InterBaseSql.VisualStudio.DataTools.Properties.Resources"

"CodeBase"="C:\\Program Files (x86)\\InterBaseDDEX\\InterBaseSql.VisualStudio.DataTools.dll"

"InvariantName"="InterBaseSql.Data.InterBaseClient"

"Technology"="{77AB9A9D-78B9-4ba7-91AC-873F5338F1D2}"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects]

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects\DataConnectionSupport]

@="InterBaseSql.VisualStudio.DataTools.IBDataConnectionSupport"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects\DataConnectionProperties]

@="InterBaseSql.VisualStudio.DataTools.IBDataConnectionProperties"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects\DataConnectionUIControl]

@="InterBaseSql.VisualStudio.DataTools.IBDataConnectionUIControl"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects\DataSourceInformation]

@="InterBaseSql.VisualStudio.DataTools.IBDataSourceInformation"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects\DataObjectSupport]

@="InterBaseSql.VisualStudio.DataTools.IBDataObjectSupport"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\DataProviders\{ACE0EC8B-0EC7-4C0C-B99A-39E6AE7B5419}\SupportedObjects\DataViewSupport]

@="InterBaseSql.VisualStudio.DataTools.IBDataViewSupport"

[HKEY\_USERS\16.0\_d30585a1\Software\Microsoft\VisualStudio\16.0\_d30585a1\_Config\Services\{EADC6C1E-17D2-42FF-9816-19850428BCF1}]

@="{0095710D-F7DC-4FA1-8FEB-C8153AA5DF75}"

"Name"="InterBase Provider Object Factory"

Once the reg file is merged in you must unload the hive from RegEdit before you launch Visual Studio.