



## DataTier.Net Users Guide



<https://github.com/DataJuggler/DataTier.Net>

Version 4.3.2  
Last Updated: 8.6.2023

**Prerequisites:**

You should read the DataTier.Net Quick Start to setup DataTier.Net before reading this document.

## Database & Fields Tutorial

**Database Requirements:**

Before creating a new Data Tier.Net project you should select or create a Microsoft SQL Server database. The database you create or select may have an unlimited number of tables and views \*

**Table Requirements**

A table may have an unlimited number of fields \*

\* Limited to the SQL Server limits of course

**Primary Key (Strongly Recommended) Updated 5.15.2016**

Data Tier.Net now handles primary keys that are non-integers and not identity insert columns. The Delete, Update and Find methods will only be created for tables that contain an identity insert primary key.

## Supported Field Data Types:

The method below is used to determine if a field is supported:

```
public static bool IsSupported(DataManager.DataTypeEnum dataType)
{
    // initial value
    bool IsSupported = false;

    switch(dataType)
    {
        case DataManager.DataTypeEnum.Autonumber:
        case DataManager.DataTypeEnum.Currency:
        case DataManager.DataTypeEnum.Double:
        case DataManager.DataTypeEnum.String:
        case DataManager.DataTypeEnum.Integer:
        case DataManager.DataTypeEnum.DateTime:
        case DataManager.DataTypeEnum.Boolean:
        case DataManager.DataTypeEnum.Guid:

            // these fields are supported
            IsSupported = true;

            // required
            break;
    }

    // return value
    return IsSupported;
}
#endregion
```

This list is using the DataManager.DataTypes enumeration which is set in the SQLDatabaseConnector.ParseDatabaseType method.

## Parse Data Type Method

```
// Determine Database Type
switch(dataType.ToLower())
{
    case "system.int16":
    case "system.int32":
    case "system.byte":
    case "int":
    case "smallint":

        // Integer
        return DataManager.DataTypeEnum.Integer;

    case "system.datetime":
    case "datetime":

        // DateTime
        return DataManager.DataTypeEnum.DateTime;

    case "system.string":
    case "varchar":
    case "nvarchar":
    case "nchar":
    case "char":

        // String
        return DataManager.DataTypeEnum.String;

    case "system.guid":
    case "uniqueidentifier":

        // GUID
        return DataManager.DataTypeEnum.Guid;

    case "system.decimal":
    case "decimal":
    case "numeric":
    case "system.single":
    case "system.double":
    case "money":
    case "float":

        // Double
        return DataManager.DataTypeEnum.Double;

    case "system.byte[]":

        // Bytes Are Not Supported Yet
        return DataManager.DataTypeEnum.Binary;

    case "system.boolean":
    case "bit":

        // Boolean
        return DataManager.DataTypeEnum.Boolean;

    default:

        // Not Supported
        return DataManager.DataTypeEnum.NotSupported;
}
```

## Using the Gateway Class

The gateway class is the bridge between your application and the Data Access Component.

The following code snippet creates a new instance of a Gateway class and loads a collection of 'FieldSet' objects for the selected DTNTable.

Note: For .NET 5 / .NET 6 and .NET 7, you must set the Connection name, which refers to an environment variable connection string.

```
// Create a new instance of a 'Gateway' object.  
Gateway gateway = new Gateway();  
  
// reload the FieldsSets  
this.SelectedTable.FieldSets = gateway.LoadFieldSetsForTable(this.SelectedTable.TableId);
```

(the screen shot above is a Custom Method, but it works the same for standard methods).

## Gateway Deep Dive

After you build your project with DataTier.Net, the standard Gateway class will contain up to 4 methods per table: Delete, Find, FetchAll, and Save.

The Update, Delete and Find methods will only be created for tables that contain a primary key, specifically an Identity Insert (auto number) primary key.

## Example of a Custom Method (Find Single Item)

Below is the Gateway class used in DataTier.Net to find Custom Methods for a table:

```
public Method FindMethod(int methodId, Method tempMethod = null)
{
    // initial value
    Method method = null;

    // if the AppController exists
    if (this.HasAppController)
    {
        // if the tempMethod does not exist
        if (tempMethod == null)
        {
            // create a temp Method
            tempMethod = new Method();
        }

        // if the methodId is set
        if (methodId > 0)
        {
            // set the primary key
            tempMethod.UpdateIdentity(methodId);
        }

        // perform the find
        method = this.AppController.ControllerManager.MethodController.Find(tempMethod);
    }

    // return value
    return method;
}
```

The 'TempMethod' object passed in is an optional parameter that is passed into the Data Access Component, where the Data Writer converts those values into SQL Parameters.

When you create a custom 'Method' using DataTier.Net, the custom method will set the values for this object instance, and then pass that onto the standard method in the Gateway to perform the data operation (Create, Read, Update or Delete, aka CRUD).

## Example of a Custom Method (Load Many Items)

```
public List<Method> LoadMethodsForTable(int tableId)
{
    // initial value
    List<Method> methods = null;

    // if the AppController exists
    if (this.HasAppController)
    {
        // Create a new instance of a 'Method' object.
        Method tempMethod = new Method();

        // Set the value for the property 'FetchAllForTable' to true
        tempMethod.FetchAllForTable = true;

        // Set the value for the property 'TableId'
        tempMethod.TableId = tableId;

        // Load the methods
        methods = LoadMethods(tempMethod);
    }

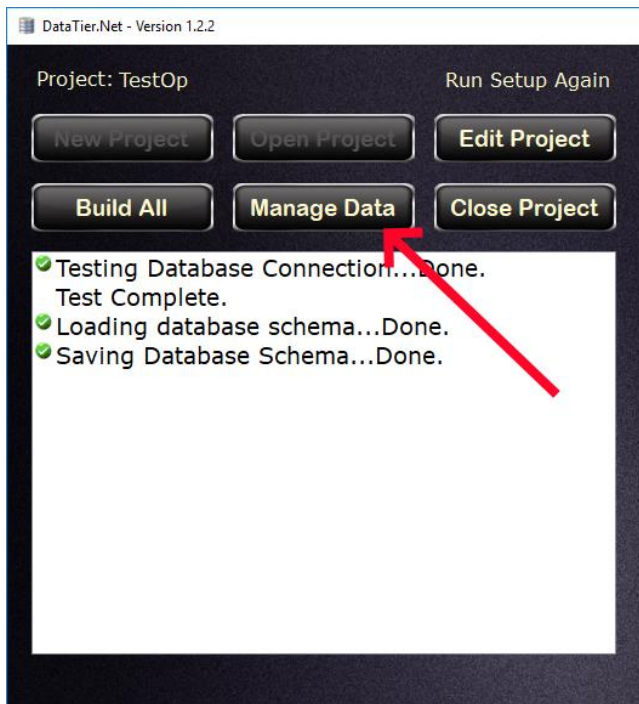
    // return value
    return methods;
}
```

The method above is code generated via DataTier.Net. Hopefully you will agree this is more elegant than most code generated code; most human written code as well.

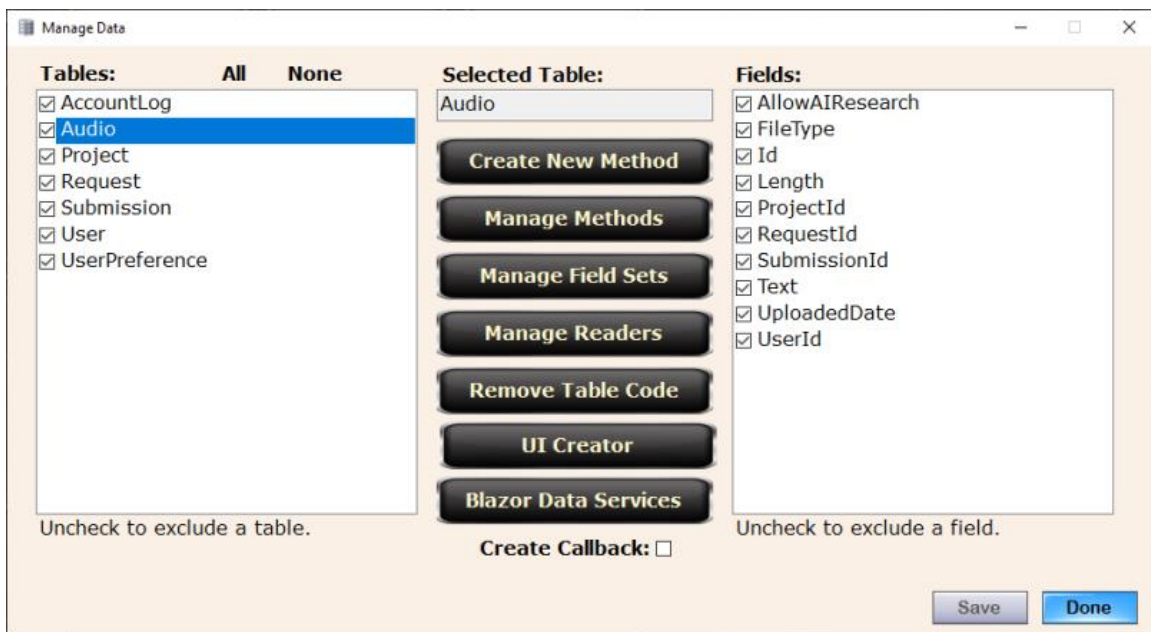
When I decided to rename the program DataTier.Net from RAD Studio Code Generation Toolkit, I created a control called the Data Editor Control, which I originally called the Table Editor Control. The Data Editor provides a way to designate specific tables, views or fields to exclude. Once I added the ability to create custom methods, custom field sets, custom readers and other things I realized Table Editor was not an appropriate name.

You can launch the Data Editor after you build by clicking the Manage Data button, which will become visible after your build completes.

## Data Editor



Screen shot of the Data Editor:





## **Data Juggler.Net**

This is the Nuget package that powers DataTier.Net, and DB Compare, SQL Snapshot and XML Mirror as well.

The project DataJuggler.Net has been tested and worked on for about 20 years. I am not stating this code is bug free by any measure, just much more tested compared to the new code in the Data Editor Control.

## **Database Compatibility**

DataTier.Net works well on any database I create. When I work on databases that were created by someone else, I have had to adapt the code a few times. A few years ago, I had to modify DataTier.Net to support tables that do not have an Identity Insert Primary key. My client's database contains 5 tables to handle OWIN login support:

AspNetRoles, AspNetUserClaims, AspNetUserLogins, AspNetUserRoles, AspNetUsers.

The tables above use a varchar primary key, and now DataTier.Net supports this feature. My point is, there will probably be new features needed to support more SQL Server database types.

I tried using the North Wind database once, but this database has a different database owner with tables like Sales.Reports (or something like that), and the database owner not being dbo messed up DataTier.Net. I spent a long time trying to fix this once, and just gave up as I never create database owner objects like that.

## **Why the Custom Method Editor Is Needed?**

The first 10 years or so I would create custom methods by hand. This wasn't that difficult, but I would have to go to SQL Server, alter an existing stored procedure to create one that accepts some parameter(s), and modify about 3 places manually in the code to execute a custom stored proc.

I never found this whole process too difficult, but at my day job where I inherited an Entity Framework project with years of code attached, I couldn't just switch data tiers overnight; even if I wanted to.

## **Custom Stored Procedures Are Now Updated When You Build**

Another Epic new feature in this version that accompanies Custom Methods, is that now when you rebuild your project DataTier.Net, the file stored procedures.sql will update any custom procs with 'Update On Build' set to true.

Maintenance was very difficult on custom procedures before this release. When new fields were added, you would have to manually update any custom procs.

## **A Word About Entity Framework**

After a few years of using EF, it is hard to deny that lambda expressions can be a time saver. My biggest reason for continuing development on this project even though 'The World' uses EF, is the amount of SQL that an EF app executes in a production environment, compared to DataTier.Net. In DataTier.Net schema reading is only performed on a dev machine.

At my current day job, I maintain a DataTier.Net working data tier in addition to our EF one. DataTier.Net has saved me on numerous occasions because in the end, I am calling code that executes a stored procedure with DataTier.Net.

Entity Framework determines on your behalf what to save. If it gets it right, all is good. When EF gets it wrong, I found it difficult to debug on a few occasions.

## The Gateway vs Data Context

The Gateway was not in the first version of Data Class Builder.Net, although I was manually creating one in all projects because it was easier to call this

```
// find the Method
Method method = gateway.FindMethod(this.MethodInfo.MethodId);
```

Rather than call all the code that I showed here:

```
public Method FindMethod(int methodId, Method tempMethod = null)
{
    // initial value
    Method method = null;

    // if the AppController exists
    if (this.HasAppController)
    {
        // if the tempMethod does not exist
        if (tempMethod == null)
        {
            // create a temp Method
            tempMethod = new Method();
        }

        // if the methodId is set
        if (methodId > 0)
        {
            // set the primary key
            tempMethod.UpdateIdentity(methodId);
        }

        // perform the find
        method = this.AppController.ControllerManager.MethodController.Find(tempMethod);
    }

    // return value
    return method;
}
```

## Getting the Last Exception

```
// Save the DTNField
tempSaved = gateway.SaveDTNField(ref clone);

// if not saved
if (!tempSaved)
{
    // set to false
    saved = false;

    // get the error, for debugging only
    error = gateway.GetLastException();
}
```

## **The Gateway is More Portable**

Another advantage that I find useful is the fact that you can create an object with one instance of a Gateway and save it with another.

The disadvantage to this (to some people), is that DataTier.Net does not keep track of change tracking like EF. I toyed around with an object Serializer, and I sometimes create my own change trackers, so this is up to your personal taste.

**XML Mirror** can be used to create an 'Original' state of your object, and then you can create a 'Current' state, and if the two strings are different, your object has changed. This is demonstrated in the Custom Reader Editor in DataTier.Net.

## **Entity Framework Is More Tested**

Entity Framework has been tested internally by Microsoft by tens of thousands of engineers, testers and researchers, and hundreds of thousands if not millions of developers outside of Microsoft.

As smart I think I am some days, the next day I find out something that makes me look so stupid, I have learned to buy good tasting socks for these foot in mouth situations. Hopefully soon I will not be the only person developing on DataTier.Net and it can keep improving.

A note to anyone like me that needs to keep pesky day jobs. You may be at a different stage in your career than I am, and being a conformist is probably in your own best interest. My boss is under the camp if I can solve something in a shorter amount of time, the method is less important than solving the problem. My boss is happy once their clients are happy.

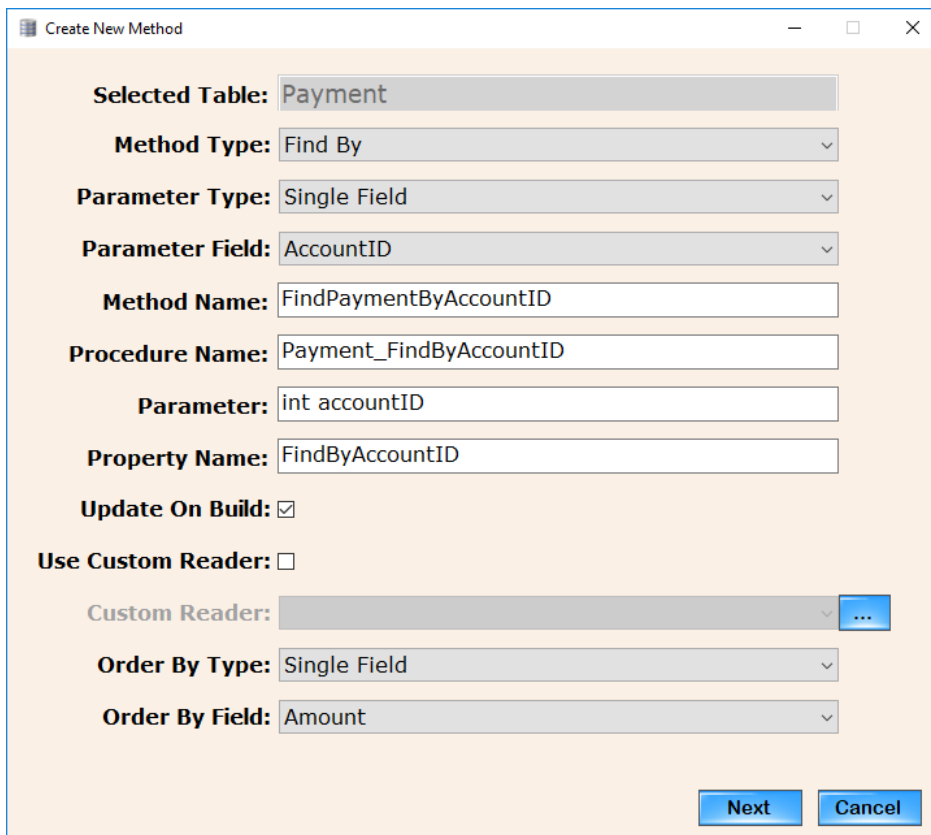
*My broker is EF Nuttin, and when he gives advice, no one listens.*

## DataTier.Net Is Not An Industry Standard (yet)

As of the time of this writing, only a handful of people have heard of DataTier.Net. If your boss or client says this project needs to use EF, then EF is what you should learn to love, or at least act like you do just like I do.

Being EF is so much easier than the old way I was using RAD Studio Code Generation Toolkit, instead of just porting my old code from Code Plex over, I knew I had to step the game for DataTier.Net up a few levels to compete. Now more than ever, I prefer DataTier.Net over EF.

## Creating Custom Methods



The screenshot shows the 'Create New Method' dialog box with the following fields and values:

- Selected Table:** Payment
- Method Type:** Find By
- Parameter Type:** Single Field
- Parameter Field:** AccountID
- Method Name:** FindPaymentByAccountID
- Procedure Name:** Payment\_FindByAccountID
- Parameter:** int accountID
- Property Name:** FindByAccountID
- Update On Build:** ☒
- Use Custom Reader:** ☐
- Custom Reader:** (empty)
- Order By Type:** Single Field
- Order By Field:** Amount

Buttons: Next, Cancel

As you select the values for Method Type, Parameter Type and either a Parameter Field or Parameter Field Set, the Method Name, Procedure Name, parameter data type and the Property Name fields are completed for you.

I will admit this is not an unbreakable editor. As you select each field, if you decide you want to change anything previously selected, start over if it doesn't behave properly. The events on this control append text onto the existing textboxes, rather than check for the previous existence and attempting to modify.

Note: I added the Parameter Type of 'No Parameters'. I can think of few scenarios like 'Top Selling Products', that do not require any parameters, and this mostly work, however you must write your own stored procedure.

A more detailed explanation of this control might help you understand how DataTier.Net works under the hood (for when something goes wrong).

**Method Type** - Find, Delete or Load.

**Method Name** – This is the Method Name in the Gateway. DataTier.Net fills this in for you, but you can change it

**Procedure Name** – This is the Stored Procedure Name and you may change it

**Parameter(s)** – The parameter (or parameters if you are using a Parameter Type of Field Set) contains the data type and field(s) for the gateway method. The Procedure Writer object converts the Parameters to SQL Data Types.

**Property Name** – The property name is the name of the property in the business object for your table. This is used to indicate to the Data Writer, the stored procedure name must be changed, and if there are any parameters, the DataWriter must convert them to SQL Parameters when it executes.

```
// if LoadByCategoryID is true
if (boatView.LoadByCategoryID)
{
    // Change the procedure name
    fetchAllBoatViewsStoredProcedure.ProcedureName = "BoatView_FetchAllByCategoryID";

    // Create the @CategoryID parameter
    fetchAllBoatViewsStoredProcedure.Parameters = SqlParameterHelper.CreateSqlParameters("@CategoryID", boatView.CategoryID);

    // set UseCustomReader to true
    fetchAllBoatViewsStoredProcedure.UseCustomReader = true;

    // set the CustomReaderName
    fetchAllBoatViewsStoredProcedure.CustomReaderName = "SpeedBoatViewReader";
}
```

The class SqlParameterHelper allows you to create up to 4 SQL Parameters.

I will update the documentation more for the Custom Field Sets, Custom Readers and Order by as I have time.

They all work basically the same, and then you end up on the 'Confirm Changes Control', which requires your authorization to modify your project.

### FieldSets

A Fieldset is a collection of fields and there are several uses for fieldsets.

1. As a parameter list for methods
2. Used to order by two or more fields \*
3. With custom readers where you only want to return a select set of fields

\* You could use a field set to order by one field, but a single field OrderByType is recommended for this use case.

## Using the Field Set Editor

The screenshot shows the 'Field Set Editor' window with the title bar 'Field Set Editor'. It has two tabs: 'Parameter Mode' (active) and 'Edit Mode'. In 'Parameter Mode', there is a 'Field Sets:' section with a list containing '<New Field Set>' and buttons 'Add', 'Edit', and 'Delete'. Below this is a 'Field Set Name:' text box containing '<New Field Set>'. There are three checkboxes: 'Parameter Mode:' (checked), 'Order By Mode:' (unchecked), and 'Reader Mode:' (unchecked). At the bottom are 'Save' and 'Cancel' buttons. On the right, the 'Selected Table:' is 'Stocks'. Below it, the 'Fields:' list contains 17 items with checkboxes: AverageVolume, Confirmed, Dividend, EPS, Exchange, ForeignStock, HasDividend, Id, Industry, IPOYear, LastDividendDate, LastPriceLookup, LastSale, MarketCap, and Name. The 'Fields Selected:' count is 0. A 'Done' button is at the bottom right.

Click the Add button and give the new field set a name. Next select two or more fields. The reason two fields is required is the same functionality can be accomplished without a field set if you want to use only one field.

### Parameter Mode

Parameter mode is used to create procedures with multiple parameters.

### Order By Mode

Order By Mode is used to order your return results by multiple fields.



## Custom Reader Mode

Custom Reader Mode is used to designate the fields returned in method using a custom reader.

## Order By Fields List Box

### Update 10.10.2019: Not All My Ideas Are Good Ones.

I threw out the first order by control because it didn't work, and now I added another list box to allow you to arrange the order of the order by fields:

The screenshot shows the 'Field Set Editor' dialog box with the 'Order By Mode' tab selected. The 'Field Sets' list on the left contains 'SectorAndIndustry' and 'IndustryAndScore', with 'IndustryAndScore' selected. Below this list are 'Add', 'Edit', and 'Delete' buttons. The 'Field Set Name' field contains 'IndustryAndScore'. The 'Parameter Mode' checkbox is unchecked, 'Order By Mode' is checked, and 'Reader Mode' is unchecked. The 'Selected Table' is 'IndustryHistory'. The 'Fields' list in the center includes 'Date', 'Down', 'Id', 'Industry' (checked), 'Score' (checked), 'Sector', 'Streak', 'Total', and 'Up'. On the right, the 'Order By Fields' list contains 'Industry' and 'Score desc', with 'Score desc' selected. Above this list is the 'Descending' checkbox, which is checked. Blue and red arrows are visible to the right of the 'Order By Fields' list, indicating a swap function. At the bottom right is a 'Done' button. A red 'Edit Mode' button is in the top right corner.

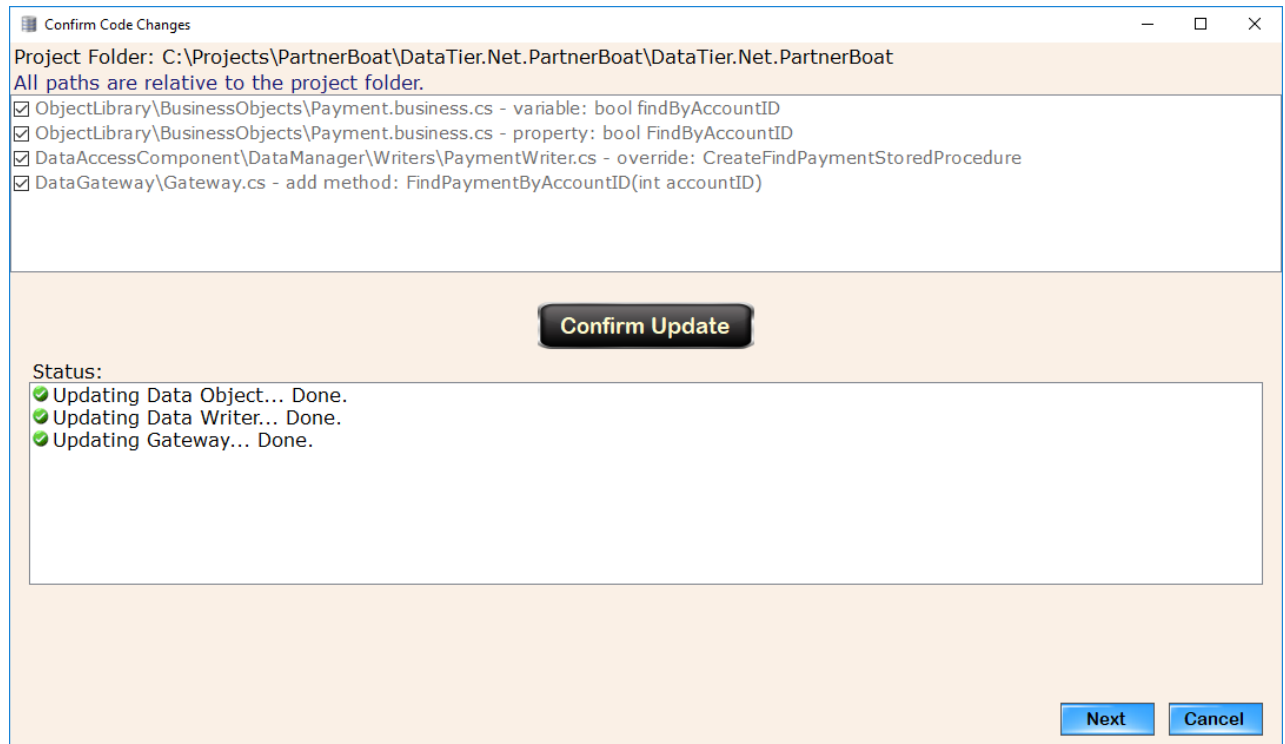
Click the name of the field to select it, and arrows appear to allow you to swap the order with a field adjacent to the selected field.

### New 9.4.2019: Descending Order Now Available

Picture shown has the Score field currently set as Descending; check or uncheck the box to switch from ascending to descending.

The Order By Fields becomes enabled if Order By Mode is selected.

## Confirm Changes Control



The messages shown above could be slightly different if any of the objects have already been updated, DataTier.Net will inform you of this fact. This was mostly for me during testing, but it is still useful, so I left it in.

Next you will be taken to the New Stored Procedure Editor. It is called new because there was already a Procedure Editor Control in the Project Wizard Control, but these two controls are very different.

## New Stored Procedure Editor

**Create Stored Procedure**

**Selected Table:**  ☒ I will update my database manually **Copy Procedure**

**Procedure Name:**  ☐ Insert Stored Procedure In Database

```
set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON
Go
-- =====
-- Procure Name: Payment_FindByAccountID
-- Author:      Data Juggler - Data Tier.Net Procedure Generator
-- Create Date: 4/5/2019
-- Description: Find an existing Payment for the AccountID given.
-- =====

-- Check if the procedure already exists
IF EXISTS (select * from syscomments where id = object_id ('Payment_FindByAccountID'))

-- Procedure Does Exist, Drop First
BEGIN

-- Execute Drop
Drop Procedure Payment_FindByAccountID

-- Test if procedure was dropped
IF OBJECT_ID('dbo.Payment_FindByAccountID') IS NOT NULL
```

**Done**

By default, you must copy the stored procedure text to your clipboard and go to SQL Server Management Studio and install it into your project's database.

If the connection string you used to create your DataTier.Net project has permission to create and execute a stored procedure, you may choose to insert the stored procedure.

I found out while writing the C# code to create a stored procedure on SQL, you must execute the If Exists, Drop Procedure Query separately from the create, so the code executes two SQL statements here, parsed by the Go keyword.

Once your stored procedure has been installed into your database, you should be able to execute the new code in your project.

## Update Proc Permissions (optional – if security problems exist)

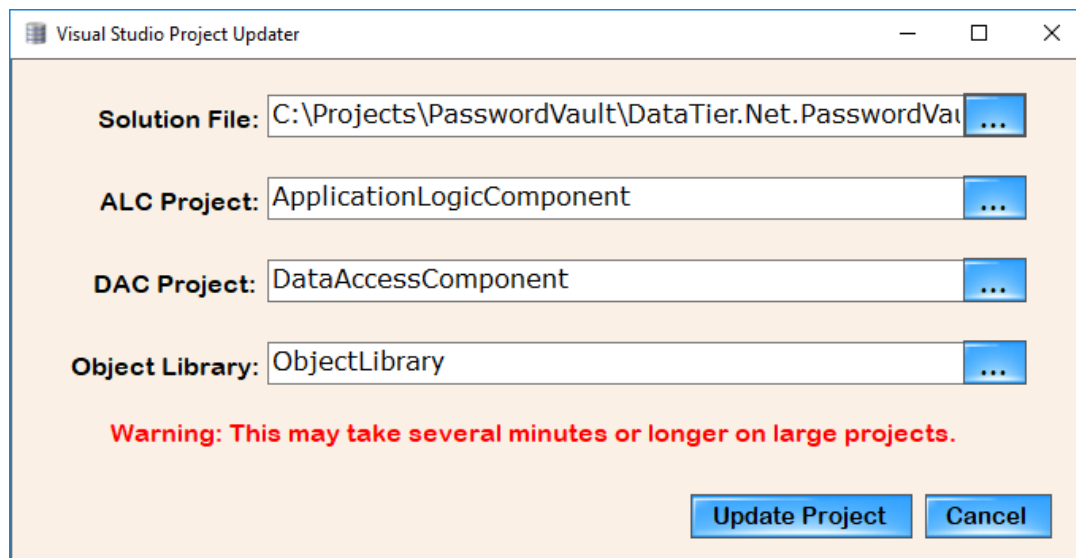
As explained in the DataTier.Net Quick Start Guide, there is an optional stored procedure in the DataTier.Net.Database called UpdateProcPermissions.

I wish I had saved the original author of this post to give them credit, but I forgot where I found this stored procedure. If you are experiencing any any problems executing stored procedures, install the UpdateProcPermissions stored procedure and execute it as shown here:

Exec UpdateProcPermissions '[User Name]'

## Visual Studio Project Updater

If new files were created during your build, the Visual Studio Project Updater will launch.



## Removing Table Code

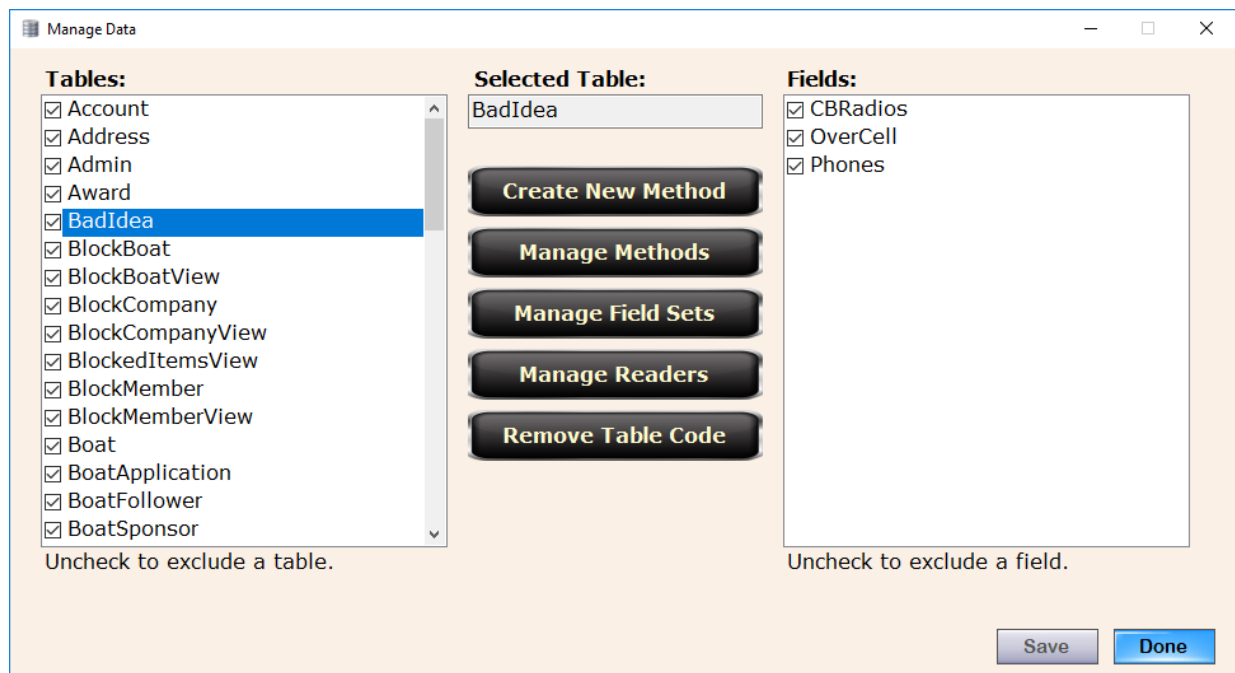
If you are anything like me, I have a lot of ideas. Some ideas turn out to be good ones, like DataTier.Net. Others are not so good. Sooner or later, you will end up needing to delete or rename a table. Removing a table was something I used to do manually, so table '[Good Idea That Fizzed]' would stick around much longer than I wished.

Now that it is a button click, it is a lot simpler, but it is still is kind of tricky because there a couple of final steps that must be taken to finalize the removal after the button click event executes. While writing the documentation and filming the Quick Start video I messed this up, so I modified the app to launch a message box indicating the final steps that are still needed.

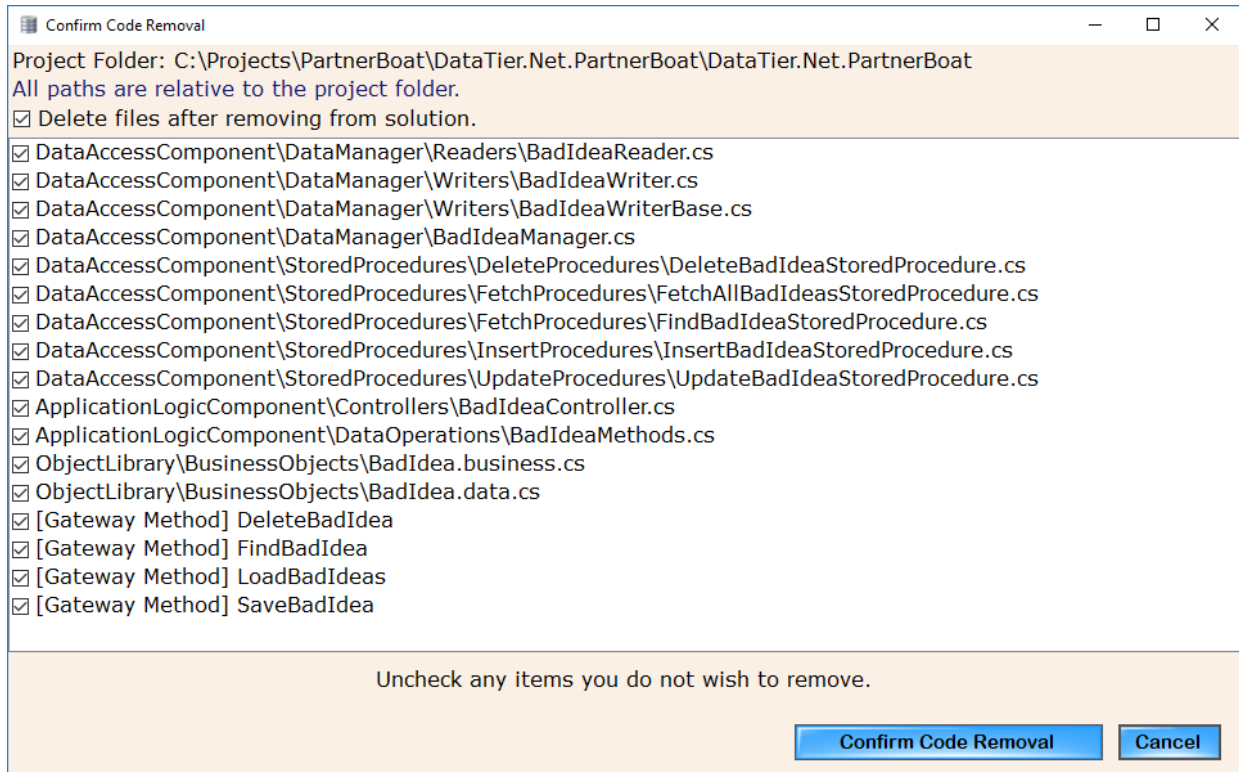
Step 1: Drop or rename a table in SQL Server Management Studio

Step 2. Select the Table in the Data Editor and click the Remove Table Code button.

Note: Click the Remove Table Code button while the table is still active (checked)



## Confirm Removal Control



Confirm Code Removal

Project Folder: C:\Projects\PartnerBoat\DataTier.Net.PartnerBoat\DataTier.Net.PartnerBoat  
All paths are relative to the project folder.

☒ Delete files after removing from solution.

- ☒ DataAccessComponent\DataManager\Readers\BadIdeaReader.cs
- ☒ DataAccessComponent\DataManager\Writers\BadIdeaWriter.cs
- ☒ DataAccessComponent\DataManager\Writers\BadIdeaWriterBase.cs
- ☒ DataAccessComponent\DataManager\BadIdeaManager.cs
- ☒ DataAccessComponent\StoredProcedures\DeleteProcedures\DeleteBadIdeaStoredProcedure.cs
- ☒ DataAccessComponent\StoredProcedures\FetchProcedures\FetchAllBadIdeasStoredProcedure.cs
- ☒ DataAccessComponent\StoredProcedures\FetchProcedures\FindBadIdeaStoredProcedure.cs
- ☒ DataAccessComponent\StoredProcedures\InsertProcedures\InsertBadIdeaStoredProcedure.cs
- ☒ DataAccessComponent\StoredProcedures\UpdateProcedures\UpdateBadIdeaStoredProcedure.cs
- ☒ ApplicationLogicComponent\Controllers\BadIdeaController.cs
- ☒ ApplicationLogicComponent\DataOperations\BadIdeaMethods.cs
- ☒ ObjectLibrary\BusinessObjects\BadIdea.business.cs
- ☒ ObjectLibrary\BusinessObjects\BadIdea.data.cs
- ☒ [Gateway Method] DeleteBadIdea
- ☒ [Gateway Method] FindBadIdea
- ☒ [Gateway Method] LoadBadIdeas
- ☒ [Gateway Method] SaveBadIdea

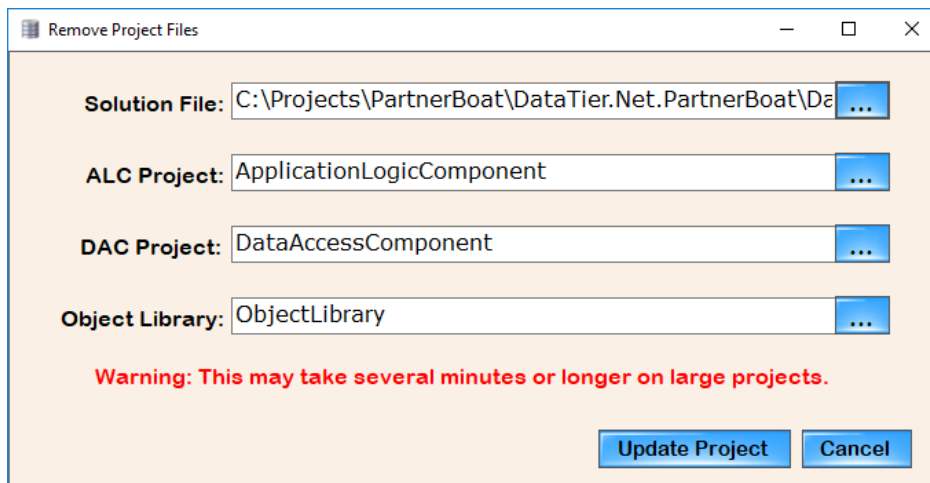
Uncheck any items you do not wish to remove.

**Confirm Code Removal** **Cancel**

Click the Confirm Code Removal button to proceed to the next step.

This will launch the Visual Studio project updater

Step 3: Select the solution file. and each of the project names will be populated like it does when you include files after building.



Remove Project Files

**Solution File:** C:\Projects\PartnerBoat\DataTier.Net.PartnerBoat\DataTier.Net.PartnerBoat\PartnerBoat.sln ...

**ALC Project:** ApplicationLogicComponent ...

**DAC Project:** DataAccessComponent ...

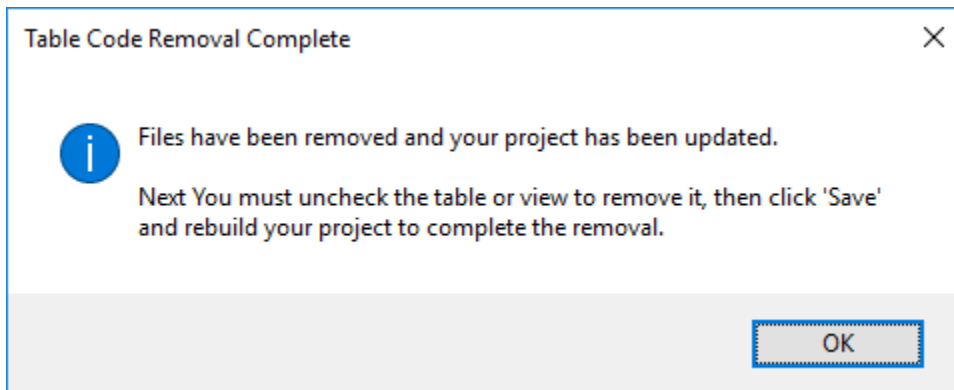
**Object Library:** ObjectLibrary ...

**Warning:** This may take several minutes or longer on large projects.

**Update Project** **Cancel**

Although the warning label that states this may take several minutes is still visible in remove files mode, this usually completes in about 5 seconds for me.

After it completes you will be shown a series of message boxes, that explain the final instructions on the second message box:



\* The two message boxes are because I was in a hurry and reused already built controls without spending a few minutes to fix it. I wasn't 100% sure if the same code that interacts with COM and performs the file inclusion would work for removal purposes. It turned out this was simple because I already had a list of files available in the Project File Manager, so all I had to do was pass in a list of files and change .add file to .remove files while in File Removal Mode.

## DataTier.Net Project Files

This screen shot a couple of pages back illustrates the project structure of a DataTier.Net project. When I get some "magical" free time, I will build some updated documents or graphical models to better illustrate the project structure.

If I were rebuilding DataTier.Net from scratch today, I would have created this project structure with three projects instead of four. There is nothing inherently wrong with the Application Logic Component; I just wish I had written the code in the Controllers and Data Operations folders to be part of the Data Access Component.

It could still be done if other people feel that the ALC is excess baggage. Removing it “shouldn’t” be too difficult (what could go wrong?).

The more features that get added like Custom Methods and Removing Tables, the more complicated future maintenance becomes, so now is the time to sound off, or forever hold your piece.

### **Blazor Data Services**

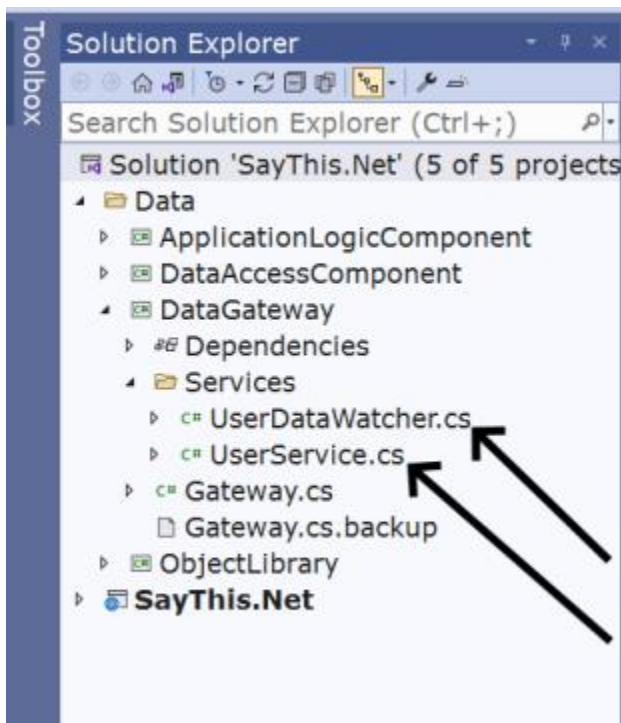
In the data editor, click ‘Create Blazor Services’ for a table, and two classes are created.

[TableName]DataServices.cs

[TableName]DataWatcher.cs

### **Blazor Data Services Files**

Data Services are created under the Services folder of the Data Gateway project.





## Example of Data Service Class:

```
namespace DataGateway.Services
{
    #region class UserService
    /// <summary>
    /// This is the Service class for managing User objects.
    /// </summary>
    1 reference
    public class UserService
    {
        #region Methods

        FindUser(int id)

        GetUserList()

        RemoveUser(User user)

        SaveUser(ref User user)

        #endregion
    }
    #endregion
}
```

This example updates the FilesDownloaded count for a user,

## Calling A Service Method

```
#region Download()
/// <summary>
/// This method Downloads the current file
/// </summary>
1 reference
public async void Download()
{
    // Set the ImagePath
    DownloadLink = ImagePath;

    // if the value for HasLoggedInUser is true
    if (HasLoggedInUser)
    {
        // Increment the value for LoggedInUser
        LoggedInUser.FilesDownloaded++;

        // save the FilesDownloaded count to the database
        await UserService.SaveUser(ref loggedInUser);
    }
}
#endregion
```

## Data Watchers (To Be Removed In .NET 8)

A data watcher is used to keep track of changes from your UI. If you want to save values or know there are changes to prompt the user, the data watcher adds a call back property to each data field in your table.

```
#region ItemChanged(object itemChanged, ListChangeTypeEnum listChangeType)
/// <summary>
/// This method Item Changed
/// </2 mmary>
1 reference
public async void ItemChanged(object itemChanged, ChangeTypeEnum listChangeType)
{
    // cast the item as a ToDo object
    User user = itemChanged as User;

    // If the user object exists
    if (NullHelper.Exists(user))
    {
        // perform the saved
        bool saved = await UserService.SaveUser(ref user);
    }
}
#endregion
```

The purpose of the Data Watcher was to allow you to save changes as the user makes data entry changes. I no longer think this is needed.

## Troubleshooting

One more advantage I find in DataTier.Net over Entity Framework is in the end you are writing code that calls a stored procedure. If there is a problem with the code, you can go and fix it, instead of being constrained to what you can do from outside of a Data Context.

If anything goes wrong with the building and including or removal of files, Visual Studio will notify you of the file(s) missing.

Clicking the 'Show All Files' button in Solution Explorer will show you if the file was created or not. You can then decide if it should or should not be included in the project.

One easy way to fix a build if something goes wrong with a table is to remove the table, build and then add it back again. Make sure to build again. This should fix most issues unless something is really messed up.

I will be posting some new videos covering a deep dive and some new tutorials and some sample projects to cover a lot of the new features as soon as possible. I didn't want to delay the release another week.



### **Bugs** (not the handsome wascally one on the left)

DataTier.Net 7 is currently stable. I know the changes for .NET 8 will introduce new bugs, so please be patient and report any issue at [GitHub.com/DataJuggler/DataTier.Net/Issues](https://github.com/DataJuggler/DataTier.Net/Issues)

DataTier.Net 8 is a big undertaking. Today I have taken the first steps.



## Hire Me 8.6.2023

I am not currently working, so if you need a part time / remote C# / SQL Server / Blazor developer, please contact me.



Please subscribe to my YouTube channel, as I publish new videos often:

<https://www.youtube.com/DataJuggler>

## Frequently Asked Questions (a work in progress)

### Question:

Will Data Tier.Net work if a table does not have a primary key?

### Answer:

Yes, but the methods for Update, Delete and Find will not be generated.

Data Tier.Net has been tested much more thoroughly on tables that do have a primary key.

**Question:** Does Data Tier.Net support composite primary keys (a primary key consisting of two or more fields)?

**Answer:**

Currently the answer is no. Data Tier.Net does have code to detect if a field has a composite primary key, but the data operation methods (find, update and delete) only work for a single primary key currently. Future updates may address this issue.

**Question:**

Does Data Tier.Net support using enumerations for integer fields?

**Answer:**

Yes. After you create a project, edit the project and click the enumerations button. Add the field name and enumeration name where they should be substituted, and when you build an integer datatype field will be created in your data tier as the enumeration name.

**Question:**

Does Data Tier.Net support Image or binary files in a database?

**Answer:**

Not at this time. At one time I had image blobs working but I had too much trouble code generating the stored procedures I decided to put this off because I do not use images in a database unless a client or employer has a requirement for this that I can't talk them out of it. My personal preferred method is to store the filename or Id in the database and retrieve the image from the file system or an Image Manager of some type. I forget the exact problem I had but it had something to do with the fact that the image field had to be last field in the select statement I think (it has been a while).

To handle this solution correctly I would code generate the table as if the image field did not exist, and then create an Image Helper to load the image or binary data. I never made this a priority because I never had the demand.



## **Ways You Can Help Support This Project**

I lack the social gene that most people have, so if you are a member of Twitter, Facebook, etc. Please help me spread the word about the DataTier.Net.

Thanks for using DataTier.Net

Corby / Data Juggler  
<https://datajuggler.com>

If you have a kind word, a suggestion or a question, please contact me at:

corby@datajuggler.com