TABLE OF CONTENTS

PREPARATION

Database Design Tips (MS SQL Server) TOC

1. For each table, I use a single identity column as the primary key. You can use GUIDs or multi-column primary keys too.
2. For each table, I add a column named "RowVersion" with a datatype of *timestamp*. (dOOdads will use this column to handle concurrency.)
3. Since dOOdads doesn't handle joins, I prepare the joins ahead of time by creating views.
4. Wherever possible, I design my database and application to minimize problems with nulls. When I do run into problems with nulls, I use *String Properties*.

Install MyGeneration and add the dOOdads project to your solution TOC

1. Download from http://www.mygenerationsoftware.com and install.
2. The installer puts the application in Program Files.
3. Add the dOOdads project to your Visual Studio solution:
    a. Right-click on the Solution item and choose "Add Existing Project".
    b. Navigate to the dOOdads project (`C:\Program Files\MyGeneration\Architectures\dOOdads\CSharp\MyGeneration.dOOdads\dOOdads.csproj`) and select. The dOOdads project will now appear in your solution.
    c. In the DbAdapters folder, open the "Entity" corresponding to your database (e.g. "SqlClientEntity"). Find "Build Action" in "Properties" and set to "Compile". Repeat for the "DynamicQuery" adapter.
4. Build the dOOdads project.

Get your .Net project ready for dOOdads

1. Add a reference to the dOOdads assembly to your project:
    a. Right-click on "References" and choose "Add Reference".
    b. On the Projects tab, double-click on the dOOdads project and click "OK".
2. Add 2 folders to the project:
    a. DAL (data access layer) - to hold the abstract classes created by MyGeneration
    b. BLL (business logic layer) - to hold the concrete classes that inherit the abstract classes

With MyGeneration:

Create the CRUD stored procedures TOC

Start MyGeneration.

1. Edit | DefaultSettings and set the database.
2. Select the template that will generate stored procedures that will work with dOOdads (e.g., "SQL Stored Procedures").
3. Run the template, select the tables you want (usually all of them), and click OK.
4. The generated DDL will be copied to the Clipboard and displayed in the Output page.
5. Go to the query tool for your database, paste the DDL code, and execute to create the stored procedures.

Create the dOOdads for tables TOC

1. Return to MyGeneration and select the template that will generate the dOOdads code for tables (eg, "dOOdads Business Entity").
2. Run the template, change the output path to the DAL folder in your project, select the tables you want (usually all), and execute to create the abstract classes.
3. The class name for each of these dOOdads will start with an underscore.

Create the dOOdads for views TOC

1. Select the template that will generate the dOOdads code for views (e.g., "dOOdads Business View").
2. Run the template, change the output path to the BLL folder in your project, select the views you want (usually all), and execute to create the concrete classes.
3. The class name for each of these dOOdads will NOT start with an underscore because they are *already* concrete classes in the BLL folder.

Write the concrete classes corresponding to the abstract classes TOC

1. Run the Concrete Classes template, saving the code in the BLL folder of your project
2. or write the classes yourself as follows:

```
public class Employees : _Employees
{
	public Employees()
	{

	}
}
```

With Visual Studio:

Include the dOOdads in your project

1. Right-click on the DAL folder and select "Include in project".
2. Repeat for the BLL folder.

Enter the connection string in the AppSettings section of the config file TOC

1. Put the connection string in the *web.config* (for web apps) or *app.config* (for WinForms apps) file and name it "**dbConnection**".
2. If the connection string is not explicitly assigned in the dOOdad, the dOOdad will look in the *config* file for "**dbConnection**".

Enhance your concrete classes TOC

If you wish to add any custom properties and methods to your business entities, you can do so in the concrete classes at any time. If your data structure changes and you need to recreate your dOOdads, the code you add here will not be overwritten.

For processing related to the business entity that normal dOOdads methods won't handle, consider using the special LoadFromSql* methods (see SPECIAL FUNCTIONS) here in the concrete class.

COMMON TASKS

Prep

```
Employees emps = new Employees();
int empID;
```

Retrieve all rows TOC

```
emps.LoadAll();
```

Retrieve one row by primary key TOC

```
emps.LoadByPrimaryKey(empID);
```

Retrieve one or more rows based on other criteria TOC

See *DYNAMIC QUERY*

Insert row TOC

```
emps.AddNew();
emps.LastName = "Smith";
emps.HireDate = DateTime.Now;
emps.Save();
empID = emps.EmployeeID; //emps returns new key value
```

Delete row (also see *SET THE CURRENT ROW*) TOC

```
//already have the desired row set as the current row, then...
emps.MarkAsDeleted();
emps.Save();
```

Update row (also see *SET THE CURRENT ROW*) TOC

```
//already have the desired row set as the current row, then...
emps.LastName = "Jones";
emps.Save();
```

Get number of rows TOC

```
emps.RowCount;
```

Iterate through rows TOC

```
if(emps.RowCount > 0)
{
	emps.Rewind(); //move to first record
	do
	{
		//do something with the current row
	} while(emps.MoveNext());
}
```

Set the sort expression TOC

```
emps.Sort = Employees.ColumnNames.LastName + " DESC";
```

Set the filter expression TOC

```
emps.Filter = Employees.ColumnNames.LastName + " LIKE A%";
```

Calculated columns (used by your application in the dataset, but not stored in the database) TOC

AddColumn
SetColumn
GetColumn
IsColumnNull
SetColumnNull

Example:

```
if(emps.LoadAll())
{
       DataColumn col = emps.AddColumn("FullName",
           Type.GetType("System.String"));
       col.Expression = Employees.ColumnNames.LastName +
           "+ ', ' + " + Employees.ColumnNames.FirstName;
       string fullName = emps.GetColumn("FullName") as string;
}
```

ConnectionString TOC

Define the connection string in your config file, naming it "dbConnection".
If you choose not to do that or you are using more than one database, you can use the ConnectionString property of the dOOdad.

```
emps.ConnectionString =
   "User=me;Password=pw;Database=Employees;DataSource=MyServer"
```

Set the current row TOC

Before you can change values in a row or delete a row, the row to change must be assigned to the DataRow property (the "current row"). This is accomplished in several ways:

1. **LoadAll** or **Query.Load** – the current row is the first row
2. **LoadByPrimaryKey** – the current row is the returned row
3. **AddNew** – the current row is the new row before it is inserted into the database
4. **Rewind** and **MoveNext** – *Rewind* sets the current row to the first row in the existing DataTable; *MoveNext* sets the current row to the row it moves to.

String properties TOC

Another *very* handy feature of dOOdads is called "string properties". This simplifies the handling of null values in both string and non-string columns. For each data column in the dOOdad, there is a string property in addition to the column property.

For example:

```
emps.Salary and emps.s_Salary
emps.HireDate and emps.s_HireDate
```

Check if a value is null:

```
if(emps.s_Salary == "")
if(emps.s_HireDate == "")
```

Set a value to null:

```
emps.s_LastName = "";
emps.s_HireDate = "";
```

DYNAMIC QUERY

The dynamic query property of the dOOdad (MyDoodad.Query) allows you to select records in an ad hoc fashion without having to write a bunch of little stored procedures. (And its design precludes the possibility of SQL-injection attacks.)

WhereParameter enums (see object model diagram for C#) TOC

Conjunctions (WhereParameter.Conj)

- And
- Or
- UseDefault

Directions (WhereParameter.Dir)

- ASC
- DESC

Operands (WhereParameter.Operand)

- Between
- Equal
- GreaterThan
- GreaterThanOrEqual
- In
- IsNotNull
- IsNull
- LessThan
- LessThanOrEqual
- Like
- NotEqual
- NotIn
- NotLike

Retrieve rows TOC

```
emps.Where.LastName.Value = "%A%";
emps.Where.LastName.Operator = WhereParameter.Operand.Like;
//Note: Conjuction not Conjunction
emps.Where.HireDate.Conjuction = WhereParameter.Conj.Or;
emps.Where.HireDate.BetweenBeginValue = "2001-01-01 00:00:00.0";
emps.Where.HireDate.BetweenEndValue = "2001-12-31 23:59:59.9";
emps.Where.HireDate.Operator = WhereParameter.Operand.Between;
emps.Query.Load();
```

Limit columns returned (Save() cannot be called after limiting columns.) TOC

```
emps.Query.AddResultColumn(Employees.ColumnNames.EmployeeID);
emps.Query.AddResultColumn(Employees.ColumnNames.LastName);
emps.Query.Load();
```

Order By TOC

```
emps.Query.AddOrderBy(Employees.ColumnNames.HireDate,
       WhereParameter.Dir.DESC);
```

Select Distinct TOC

```
emps.Query.Distinct = true;
```

Select Top N TOC

```
emps.Query.Top = 10;
```

Parentheses TOC

```
emps.Query.OpenParenthesis();
emps.Query.CloseParenthesis();
```

GenerateSQL TOC

A diagnostic function that returns the SQL statement created for the dynamic query. After calling this you cannot load the object. Better to use *LastQuery*.

LastQuery TOC

A string property that contains the SQL text of the most recently generated SQL statement.

ReturnReader TOC

```
SqlDataReader reader = emps.Query.ReturnReader() as SqlDataReader;
```

DATA BINDING

Data binding to a datagrid (Web app)

TOC

1. Create a typed Dataset (.xsd file) with a DataTable to match the dOOdad columns.
2. Add the Dataset to your form.
3. Add a DataView and assign the new Dataset.DataTable to its Table property.
4. Bind the datagrid to the DataView and design the DataGrid as you wish.
5. In the code-behind, change the DataView's Table property to the DefaultView property of the dOOdad:

```
if(emps.LoadAll())
{
        dvEmps = emps.DefaultView;
        dgEmps.DataSource = dvEmps;
        dgEmps.DataBind();
}
```

(NOTE: The only reason to use a dataset here is to help design the datagrid visually at design time. The dataset is not referenced in code at all.)

6. If you wish, you can store the dataview in the Session as follows:

```
Session["dvEmps"] = emps.DefaultView;
dvEmps = Session["dvEmps"];
```

Data binding to a ComboBox (WinForms app)

TOC

```
//get data
emps.Query.AddResultColumn(Employees.ColumnNames.EmployeeID);
emps.Query.AddResultColumn(Employees.ColumnNames.LastName);
emps.Query.AddOrderBy(Employees.ColumnNames.LastName,
      WhereParameter.Dir.ASC);
emps.Query.Load();

//bind to combobox
cmbEmployees.DisplayMember = Employees.ColumnNames.LastName;
cmbEmployees.ValueMember = Employees.ColumnNames.EmployeeID;
cmbEmployees.Datasource = emps.DefaultView;
```

Data binding to a DropDownList (Web app)

TOC

```
//get data
emps.Query.AddResultColumn(Employees.ColumnNames.EmployeeID);
emps.Query.AddResultColumn(Employees.ColumnNames.LastName);
emps.Query.AddOrderBy(Employees.ColumnNames.LastName,
      WhereParameter.Dir.ASC);
emps.Query.Load();

//bind to drop-down list
ddlEmployees.Datasource = emps.DefaultView;
ddlEmployees.DataTextField = Employees.ColumnNames.LastName;
ddlEmployees.DataValueField = Employees.ColumnNames.EmployeeID;
ddlEmployees.Databind();
```

SPECIAL FUNCTIONS

Executing custom stored procedures and SQL statements: TOC

Method Name	*Overloads*	*Returns*	*Fills dOOdad*
LoadFromSql	(s) (s, p)[L] (s, p, c)[L]	Boolean (true if any rows are found)	Yes
LoadFromSqlNoExec	(s) (s, p)[L] (s, p, c, t)[L]	Integer (number of rows affected)	No
LoadFromSqlReader	(s) (s, p)[L] (s, p, c)[L]	DataReader	No
LoadFromSqlScalar	(s) (s, p)[L] (s, p, c, t)[L]	Object (first column of first row in resultset)	No
LoadFromRawSql	(s, p)[P]	Boolean (true if any rows are found)	Yes

s : stored procedure (or SQL statement depending on command type)
p : parameters (ListDictionary[L] or ParamArray[P])
c : CommandType (sproc, table, SQL statement)
t : timeout (-1 is standard)

These functions can be used to extend the concrete class (in fact, they cannot be accessed outside of the concrete class). See the examples on the next page.

Examples:

LoadFromSql

```
ListDictionary Parms = new ListDictionary();
Parms.Add(Employees.Parameters.EmployeeID, 152);
if(this.LoadFromSql("proc_GetSpecialEmployee", Parms))
{
      ...
}
```

LoadFromSqlNoExec

```
int NbrOfChecks = this.LoadFromSqlNoExec("proc_CalculatePayroll");
```

LoadFromSqlReader

```
SqlDataReader rdr =
      this.LoadFromSqlReader("proc_GetSpecialEmployee", Parms)
      as SqlDataReader;
```

LoadFromSqlScalar

```
DateTime EarliestHireDate;
Parms.Clear();
Parms.Add("@Active", 1);
EarliestHireDate =
      Convert.ToDateTime(this.LoadFromSqlScalar("GetEarliestHireDate",
      Parms));
```

LoadFromRawSql

```
this.LoadFromRawSql("SELECT MIN(EM_HireDate) FROM tEmployees WHERE EM_Active =
   {0}","1");
```

FromXml / ToXml TOC

```
emps.Query.Load();              //emps.RowCount = 200
emps.LastName = "Griffinski";   //Change first row
emps.GetChanges();              //emps.RowCount now = 1
string xml = emps.ToXml();

//Now reload that single record into a new Employees object and save it
Employees empsClone = new Employees();
empsClone.FromXml(xml);
empsClone.Save();
```

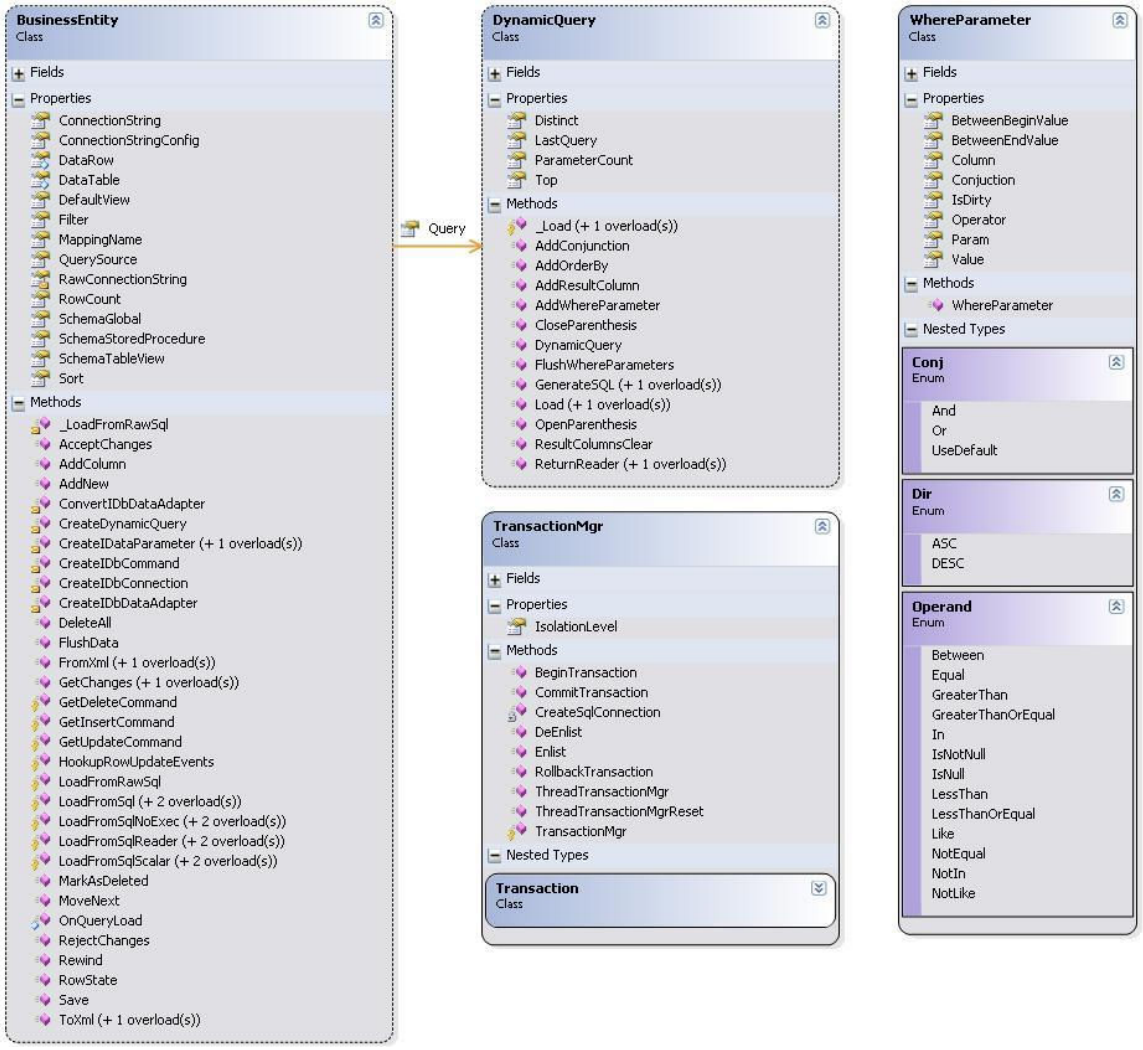
TOC

TRANSACTIONMGR

```
TransactionMgr tx = TransactionMgr.ThreadTransactionMgr();

try
{
	Employees emps = new Employees();
	emps.AddNew();
	emps.FirstName = "Jimmy";
	emps.LastName = "Doe";

	Products prds = new Products();
	prds.AddNew();
	prds.ProductName = " Lunch Box ";
	prds.Discontinued = false;

	tx.BeginTransaction();
	emps.Save();
	prds.Save();
	tx.CommitTransaction();
}
catch(Exception ex)
{
	tx.RollbackTransaction();
	TransactionMgr.ThreadTransactionMgrReset();
}
```

dOOdads Object Model

(This is the C# flavor. There are slight differences in the Enums in the VB version.)



ADDENDA

How to create a typed dataset: TOC

Assuming you're in Visual Studio:

1. From the menu, click **File** | **Add new item**.
2. Select **DataSet** from the item types presented and give it a name.
3. When the .xsd appears, you can do one of two things:
    1. Drag and drop a table or view from Server Explorer, or
    2. Hand-build a DataTable:
        i. Right-click on it and select **Add** | **New element**.
        ii. A DataTable will be created; type the name of the table in place of "element 1".
        iii. List the fields and datatypes to match your dOOdad. This can match a table, a view, the fields selected in a dynamic query – anything that a dOOdad can represent.
4. Save the xsd.
5. Go to your web form and add a dataset from the Toolbox. Select the dataset you just created.