

TABLE OF CONTENTS

Preparation	Database Design Tips	2
	Installation and Setup	2
	CRUD procedures	3
	dOODads for tables	3
	dOODads for views	3
	Concrete classes	3
	ConnectionString	4
	Enhancing concrete classes	4
Common tasks	Retrieve all rows	5
	Retrieve one row by primary key	5
	Retrieve one or more rows based on other criteria	5
	Insert row	5
	Delete row	5
	Update row	5
	Get number of rows	5
	Iterate through rows	5
	Set the sort expression	6
	Set the filter expression	6
	Calculated columns	6
	Set ConnectionString explicitly	6
	Set the current row	6
	String properties	6
DynamicQuery	WhereParameter enums	8
	Retrieve rows	8
	Limit columns returned	9
	Order By	9
	Select Distinct	9
	Select Top N	9
	Parentheses	9
	GenerateSQL	9
	ReturnReader	9
Data binding	Datagrid	10
	ComboBox	10
	DropDownList	10
Special functions	LoadFromSql*	11
	FromXml/ToXml	12
	TransactionMgr	13
	Object model	14
Addenda	Creating a typed dataset	15

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\VB.Net\MyGeneration.dOODads\dOODads.vbproj) 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
    Inherits _Employees
End Class
```

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

```
Dim oEmps As New Employees
Dim EmpID as Long
```

Retrieve all rows

```
oEmps.LoadAll
```

[TOC](#)**Retrieve one row by primary key**

```
oEmps.LoadByPrimaryKey (EmpID)
```

[TOC](#)**Retrieve one or more rows based on other criteria**

See [DYNAMIC QUERY](#)

[TOC](#)**Insert row**

```
With oEmps
  .AddNew()
  .EM_LastName = "Smith"
  .EM_HireDate = Now
  .Save()
End With
EmpID = oEmps.EM_ID 'oEmp returns new key value
```

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

```
'already have the desired row set as the current row, then...
With oEmps
  .MarkAsDeleted()
  .Save()
End With
```

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

```
'already have the desired row set as the current row, then...
With oEmps
  .EM_LastName = "Jones"
  .Save()
End With
```

[TOC](#)**Get number of rows**

```
oEmps.RowCount
```

[TOC](#)**Iterate through rows**

```
With oEmps
  If .RowCount > 0 Then
    .Rewind() 'move to first record
    Do While .MoveNext
      'do something with the current row
    Loop
  End If
End With
```

[TOC](#)

Set the sort expression[TOC](#)

```
With oEmps
    .Sort = .ColumnNames.EM_LastName & " DESC"
End With
```

Set the filter expression[TOC](#)

```
With oEmps
    .Filter = .ColumnNames.EM_LastName & " LIKE A%"
End With
```

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

```
AddColumn
SetColumn
GetColumn
IsColumnNull
SetColumnNull
```

Example:

```
If oEmps.LoadAll() Then
    Dim col As DataColumn
    col = oEmps.AddColumn("FullName", Type.GetType("System.String"))
    col.Expression = oEmps.ColumnNames.LastName + " + ' , ' + " +
        oEmps.ColumnNames.FirstName
    Dim fullName As String
    fullName = CType(oEmps.GetColumn("FullName"), 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.

```
oEmps.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:

```
oEmps.EM_Salary and oEmps.s_EM_Salary  
oEmps.EM_HireDate and oEmps.s_EM_HireDate
```

Check if a value is null:

```
if oEmps.s_EM_Salary = ""  
if oEmps.s_EM_HireDate = ""
```

Set a value to null:

```
oEmps.s_EM_Salary = ""  
oEmps.s_EM_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

Retrieve rows

[TOC](#)

```
With oEmps
    .Where.EM_LastName.Value = "%A%"
    .Where.EM_LastName.Operator = WhereParameter.Operand.Like_
    .Query.AddConjunction(WhereParameter.Conj.OR_)
    .Where.EM_HireDate.BetweenBeginValue = "'1/1/2001 00:00AM'"
    .Where.EM_HireDate.BetweenEndValue = "'12/31/2001 11:59PM'"
    .Where.EM_HireDate.Operator = WhereParameter.Operand.Between
    .Query.Load()
End With
```


Limit columns returned[TOC](#)

```
With oEmps
    .Query.AddResultColumn(.ColumnNames.EM_ID)
    .Query.AddResultColumn(.ColumnNames.EM_LastName)
    .Query.Load
End With
(NOTE: .Save() cannot be called after limiting columns.)
```

Order By[TOC](#)

```
oEmps.Query.AddOrderBy(.ColumnNames.EM_HireDate, WhereParameter.Dir.DESC)
```

Select Distinct[TOC](#)

```
oEmps.Query.Distinct = True
```

Select Top N[TOC](#)

```
oEmps.Query.Top = 10
```

Parentheses[TOC](#)

```
oEmps.Query.OpenParenthesis
oEmps.Query.CloseParenthesis
```

GenerateSQL[TOC](#)

A diagnostic function that returns the SQL statement created for the dynamic query.

Better to use [LastQuery](#).

(NOTE: After calling this you cannot load the object.)

LastQuery[TOC](#)

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

ReturnReader[TOC](#)

```
Dim rdr as SqlClient.DataReader
rdr = oEmp.ReturnReader
```

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 oEmps.LoadAll() Then
    dvEmps = oEmps.DefaultView
    dgEmps.DataSource = dvEmps
    dgEmps.DataBind()
End If
```

(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") = oEmps.DefaultView
dvEmps = Session("dvEmps")
```

Data binding to a ComboBox (WinForms app)

[TOC](#)

```
'get data
With oEmps
    .Query.AddResultColumn(.ColumnNames.EM_ID)
    .Query.AddResultColumn(.ColumnNames.EM_LastName)
    .Query.AddOrderBy(.ColumnNames.EM_LastName, WhereParameter.Dir.ASC)
    .Query.Load()
End With

'bind to combobox
With cmbEmployees
    .DataSource = oEmps.DefaultView
    .DisplayMember = oEmps.ColumnNames.EM_LastName
    .ValueMember = oEmps.ColumnNames.EM_ID
    .DataBind()
End With
```

Data binding to a DropDownList (Web app)

[TOC](#)

```
'get data
With oEmps
    .Query.AddResultColumn(.ColumnNames.EM_ID)
    .Query.AddResultColumn(.ColumnNames.EM_LastName)
    .Query.AddOrderBy(.ColumnNames.EM_LastName, WhereParameter.Dir.ASC)
    .Query.Load()
End With

'bind to drop-down list
With ddlEmployees
    .DataSource = oEmps.DefaultView
    .DataTextField = oEmps.ColumnNames.EM_LastName
    .DataValueField = oEmps.ColumnNames.EM_ID
    .DataBind()
End With
```

SPECIAL FUNCTIONS

Executing custom stored procedures and SQL statements:

[TOC](#)

<i>Method Name</i>	<i>Overloads</i>	<i>Returns</i>	<i>Fills dOOdad</i>
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 (spc, 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 examples on next page.

LoadFromSql* Examples:LoadFromSql

```
Dim Parms As ListDictionary = New ListDictionary
With MyDoodad
    Parms.Add(.Parameters.EM_ID, 152)
    If .LoadFromSql("proc_GetSpecialEmployee", Parms) Then
        ...
    End If
End With
```

LoadFromSqlNoExec

```
NbrOfChecks = MyDoodad.LoadFromSqlNoExec("proc_CalculatePayroll")
```

LoadFromSqlReader

```
Dim rdr As New SqlDataReader
rdr = MyDoodad.LoadFromSqlReader("proc_GetSpecialEmployee", Parms)
```

LoadFromSqlScalar

```
Dim EarliestHireDate As DateTime
Parms.Clear
Parms.Add("@EM_Active", 1)
EarliestHireDate = MyDoodad.LoadFromSqlScalar("GetEarliestHireDate", Parms)
```

LoadFromRawSql

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

FromXml / ToXml[IOC](#)

```
Dim oEmps As New Employees
oEmps.Query.Load()
oEmps.FirstName = "Griffinski"           'Change first row
oEmps.GetChanges()                       'oEmps.RowCount now = 1
dim xml As String = oEmps.ToXml           'changed row is saved to string as XML

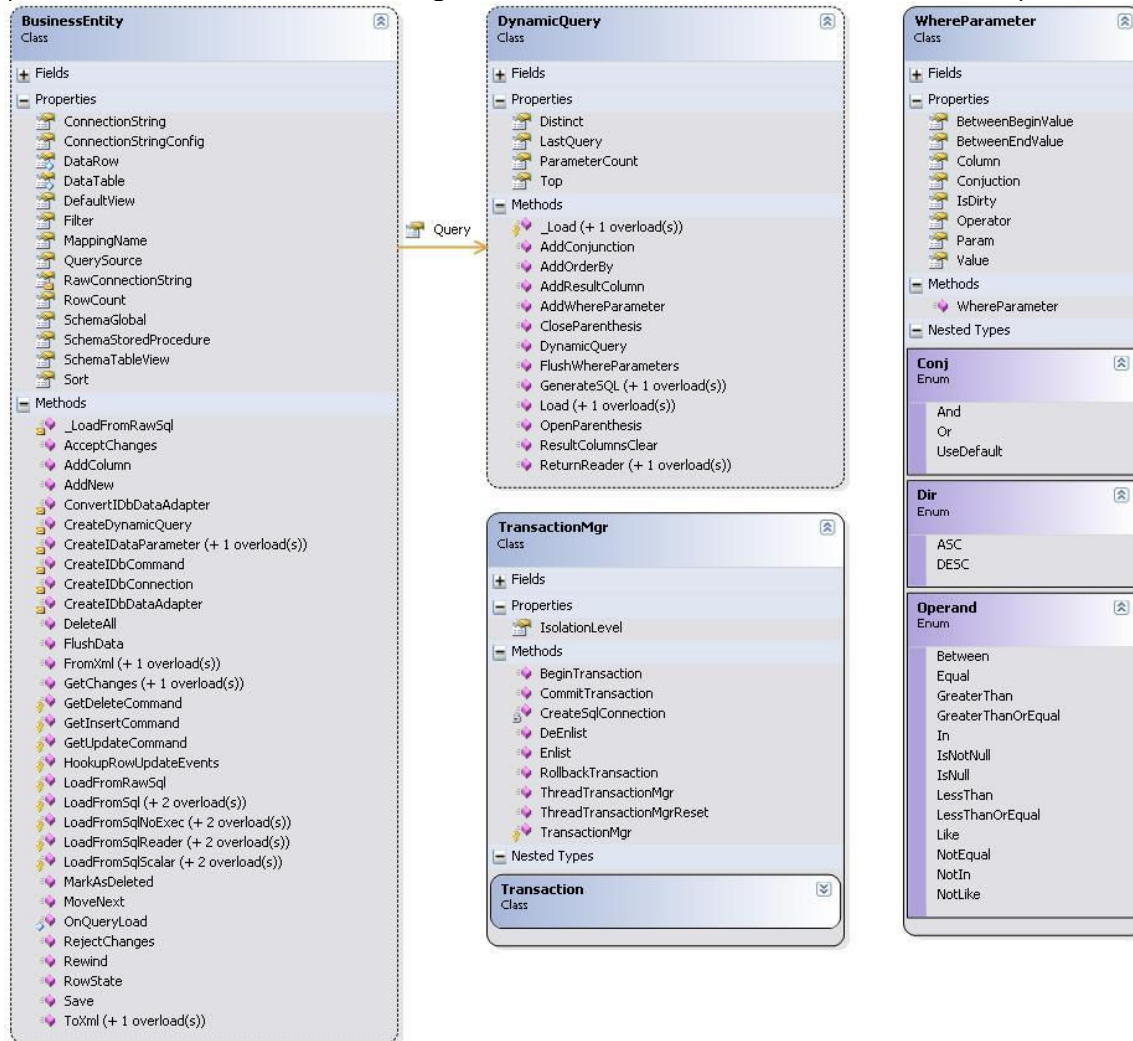
'Now reload that single record into a new Employees object and save it
Dim oEmpsClone As New Employees
oEmpsClone.FromXml(xml)                   'load row values from the XML string
oEmpsClone.Save()
```

TRANSACTIONMGR

```
Dim tx as TransactionMgr
tx = TransactionMgr.ThreadTransactionMgr()
Try
    Dim oEmps As New Employees
    oEmps.AddNew()
    oEmps.FirstName = "Jimmy"
    oEmps.LastName = "Doe"
    Dim oProds As New Products
    oProds.AddNew()
    oProds.ProductName = "Lunch Box"
    oProds.Discontinued = False
    tx.BeginTransaction()
    oEmps.Save()
    oProds.Save()
    tx.CommitTransaction()
Catch ex As Exception
    tx.RollbackTransaction()
    tx.ThreadTransactionMgrReset()
End Try
```

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:
 - (a) Drag and drop a table or view from Server Explorer, or
 - (b) 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.