GaddioORM

# Summary

GaddioORM is an object-relational mapping project for .NET that allows the user to:

* Avoid writing boilerplate data-access code against a relational database
* Provide simple, object-based access to a relational database, similar to Ruby’s ActiveRecord concept and Microsoft’s Entity Framework.
* Allow developers with little relational database experience to focus on writing business logic code.
* Avoid worrying about hardening code against SQL injection attacks, a common concern when writing dynamic queries.
* Perform complex chains of CRUD functions within transactions.

# Features

## Object-relational mapping

GaddioORM provides a simple mechanism to allow developers to associate a class and its properties to a database table and its columns.

### DBTable attribute

The DBTable attribute defines which table in the database the class represents. This is used by the DA class when generating the opening clause of a dynamic SQL queries to perform basic CRUD functions (Create, Read, Update, Delete).

[DBTable(“Products”)]

public class Product {

}

### DBField attribute

The DBField attribute defines which column a particular class property represents. This is used by the DA class when generating the SET, WHERE, and ORDER BY clauses in dynamic SQL queries.

### DBFieldKeys enum

The DBField.DBFieldKeys enum allows the user to specify column attributes such as whether the column participates in a primary key, whether the column is required to have a value when inserting a record, etc.

[DBField(“ProductId”, DBField.DBFieldKeys.Primary | DBField.DBFieldKeys.AutoIncrement)]

public int Id { get; set; }

[DBField(“ProductName”, DBField.DBFieldKeys.Insertable)]

public string Name { get; set; }

[DBField(“ProductWeight”)]

public double Weight { get; set; }

## CRUD Functions

The DA class provides a variety of methods that allow the user to perform basic CRUD operations without writing boilerplate queries, whether embedded in C# code or as stored procedures.

### Get<T>

The Get and GetList family of functions allows the user to retrieve one or more objects of a specified type T.

Example:

Product proto = new Product() {

Id = 71

}

Product result = DA.Get<Product>(proto, “Id”);

The DA class generates the following SQL command under the hood and executes it against the database.

SELECT \*

FROM Products

WHERE ProductId=@ProductId

Note: The @ProductId variable is assigned the value of 71 using Microsoft’s SqlCommand parameter interface, which provides protection against SQL injection attacks.

### Create<T>

The Create family of functions allows the user to insert records into the database.

Example:

Product proto = new Product() {

Id = 72,

Name = “Widget 2.0”

}

int newProductId = DA.Create<Product>(proto);

The DA class generates and executes the following SQL:

INSERT INTO Products

(Name)

VALUES

(@Name)

SELECT CAST(@@SCOPE\_IDENTITY() as int)

Note: since the DBField attribute on the Id property defined that column as a primary key participant and an auto-increment, the query generation did not include the ProductId column in the insert. Rather, it includes a SELECT statement that returns the auto-incremented ID for the new product.

### Update<T>

The Update family of functions allows the user to update specific columns in a particular record.

Example:

Product proto = new Product() {

Id = 72,

Name = “Widget 3.0” // changing from 2.0 to 3.0

}

DA.Update<Product>(proto, “Name”);

The DA class generates the following SQL code under the hood:

UPDATE Products

SET ProductName=@ProductName

WHERE ProductId=@ProductId

Note: the Weight property is not included in the UPDATE SQL code because it was not passed in the Update function call. To include multiple properties, use the following code:

DA.Update<Product>(proto, “Name”, “Weight”);

This produces:

UPDATE Products

SET ProductName=@ProductName, ProductWeight=@ProductWeight

WHERE ProductId=@ProductId

### Delete<T>

The delete family of functions performs a delete operation based on the primary key of the table, as defined by the DBField attributes. This operation, as with all delete operations, should be used with care.

DA.Delete<Product>(proto);

This produces:

DELETE FROM Products

WHERE ProductId=@ProductId

## Transactions

GaddioORM provides the ability to chain multiple CRUD operations together in transactions, ensuring data integrity. The DATransaction class encapsulates a .NET SqlConnection class and provides additional functionality to help keep track of and rollback transactions when necessary (e.g. when an exception is thrown). The example below show two database operations in a single transaction: 1) create a new product, 2) add that product to a group, i.e. create a ProductGroupMembership.

using (DATransaction tx = DATransaction.BeginProcessContextTransaction()) {

try {

Product proto = new Product() {

Id = 73,

Name = “New Widget”,

Weight = 4.5

};

int newProductId = DA.Create<Product>(proto, tx);

ProductGroupMembership membershipProto = new ProductGroupMembership() {

GroupId = 87,

ProductId = newProductId

};

DA.Create<ProductGroupMembership>(membershipProto, tx);

tx.Commit();

}

catch (Exception) {

tx.Rollback();

throw;

} // end try/catch

} // end using