

2010



USER GUIDE: CHAPTER 1

PROJECT WIZARD

Business Logic Layer – Part 1

Business Logic Layer Page

You can choose to include database views, and affect the construction process in multiple ways. When you are writing custom code for your application, the options available on this page are a real life saver.

Business Logic Layer

A business logic layer is a collection of data controllers representing entities of the logical domain model of your application. Business objects represent a [denormalized](#) view of data suitable for display in user interface and coding of business rules. All possible relationships are discovered automatically for each database table through the available foreign keys. Optional discovery depth can limit the number of fields in objects.

Advanced Options:

- Generate a shared business rules class to implement [global logging](#) of actions executed by data controllers of your application.
- Include database views as read-only data controllers for reporting and customization.
- Use custom discovery depth, labeling expressions, field exclusion rules, table keys, and table field mapping to compose business objects.
- Generate business logic layer code objects for this application for use in [business rules](#) and custom [user interface](#) forms.

Cancel Back Next

Views

The code generator will by default automatically include all database views in your project. These database views are treated as reports in the application, and can be used to import data from other database application. To enable views, check the second option on the business logic layer page.

The example below displays the database view called *Customer and Supplier by City*.

The screenshot shows a web application interface for 'MyCompany'. The main content area displays a report titled 'Customer and Suppliers by City'. The report includes a search bar, a 'Quick Find' button, and a 'View: Customer and Suppliers by City' dropdown. The data is presented in a table with columns for Company Name, City, Contact Name, and Relationship. The table lists 10 records, including companies like Alfreds Futterkiste, Ana Trujillo Emparedados y helados, and Antonio Moreno Taquería. The interface also features a navigation bar with 'Home', 'Customers', 'Employees', 'Categories', 'Customer Demographics', 'Region', 'Reports', and 'Membership'. A sidebar on the left contains 'About' and 'See Also' sections. The bottom of the page shows pagination information: '« Previous | Page: 1 2 3 4 5 6 7 8 9 10 ... | Next »' and 'Items per page: 10, 15, 20, 25 | Showing 1-10 of 120 items | Refresh'.

Company Name	City	Contact Name	Relationship
Alfreds Futterkiste	Berlin	Maria Anders	Customers
Ana Trujillo Emparedados y helados	México D.F.	Ana Trujillo	Customers
Antonio Moreno Taquería	México D.F.	Antonio Moreno	Customers
Around the Horn	London	Thomas Hardy	Customers
Aux joyeux ecclesiastiques	Paris	Guyène Nodier	Suppliers
Berglunds snabbköp	Luleå	Christina Berglund	Customers
Bigfoot Breweries	Bend	Cheryl Saylor	Suppliers
Blauer See Delikatessen	Mannheim	Hanna Moos	Customers
Blondesdél père et fils	Strasbourg	Frédérique Citeaux	Customers
Bólido Comidas preparadas	Madrid	Martin Sommer	Customers

This view combines data from the Customers and Suppliers tables. You can page, sort, and filter the data. You can also download and export, generate an RSS feed, or create reports. However, you cannot select or delete any records.

Working with Views

Normally, data controllers based on views do not permit data modification. If you explicitly specify a primary key, then you will gain the ability to select data. If you choose a physical table to save to, then you will be able to update and create data. You can also make stored procedures to update data tables hidden by views.

Planning For Code Customization

You can choose to generate business logic layer code objects. A shared business rules class can also be generated, useful for global logging.

Code Object CRUD Methods

When you activate code objects, you have the option of specifying different names for the CRUD object methods. By default, method Insert allows creation of data. Method Select and SelectSingle allow the reading of data. Method update updates the data, and method delete removes the data from the database. The code objects will be generated in the language of your choice.

Generate business logic layer code objects for this application for use in [business rules](#) and [custom user interface forms](#).

You can create object instances by calling *data selection* methods of object factories, manipulate object properties, *insert*, *update*, and *delete* objects. All objects of business logic layer are designed to support *data binding* with [ObjectDataSource](#) component.

Full support is provided for paging, sorting, filtering, and editing of records of any size with maximum flexibility and minimal database interaction. Business layer objects are provided with methods that allow to **create**, **read**, **update**, and **delete** database information. Please specify the method names.

Create method to insert a record in a database table:

Read method to retrieve **all records** that match specified filtering parameters:

Read method to retrieve a **single record** matched to a primary key value:

Update method to modify a record in a database table:

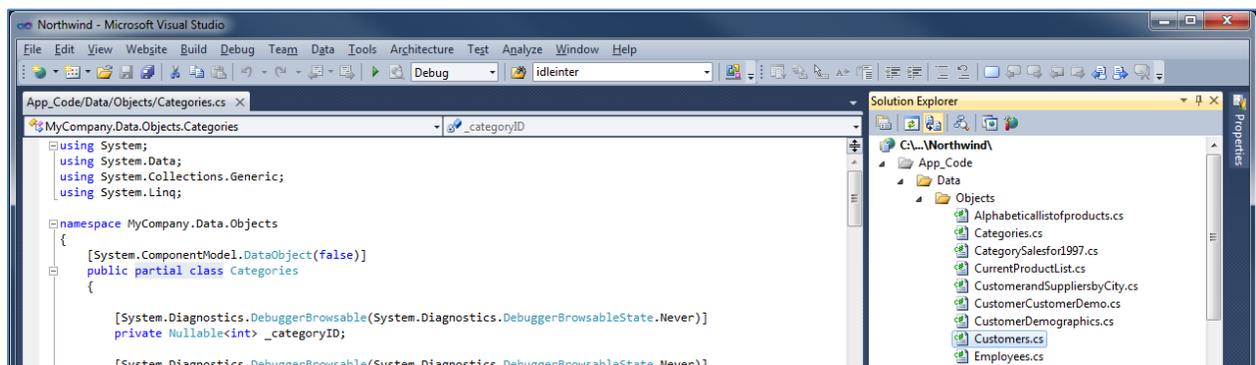
Delete method to remove a record from a database table:

If you change the business object method names then please make sure to use these names when following instructions in online tutorials.

Using Code Objects

Code objects are stored in [YourNamespace]/Data/Objects , and can be used to write custom business logic when you need to manipulate data.

Open the website in Visual Studio by clicking on File | Open Web Site, and select the correct folder location. Using the solution explorer, open up App_Code/Data/Objects. If you open Customers.cs, you can see that the class is declared as partial.



You can scroll down to see the Select, SelectSingle, Insert, Update, and Delete methods that have been generated for this controller. The code will be in the programming language of your choice.

```
...
public static List<MyCompany.Data.Objects.Categories> Select(MyCompany.Data.Objects.Categories qbe)
{
    return new CategoriesFactory().Select(qbe);
}

public static MyCompany.Data.Objects.Categories SelectSingle(Nullable<int> categoryID)
{
    return new CategoriesFactory().SelectSingle(categoryID);
}

public int Insert()
{
    return new CategoriesFactory().Insert(this);
}

public int Update()
{
    return new CategoriesFactory().Update(this);
}

public int Delete()
{
    return new CategoriesFactory().Delete(this);
}
...
```

Shared Business Rules

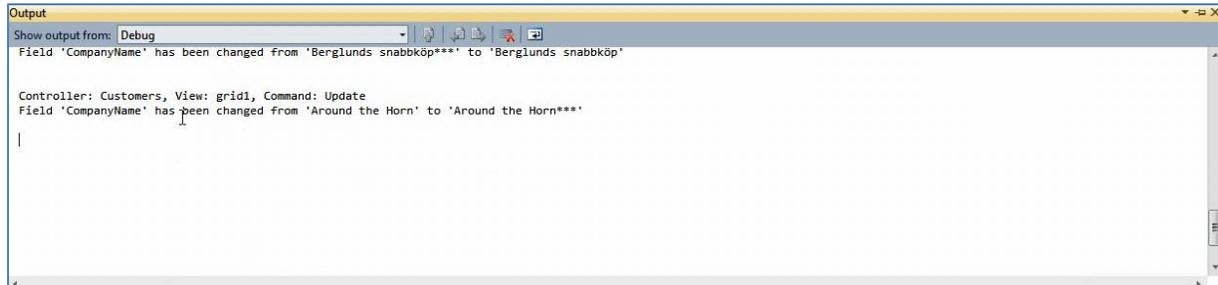
There is a single point of entry for Update, Insert, and Delete operations, which can be used to log all data manipulations. You can also use it to enrich data with common properties, such as Modified By, Modified On, and etcetera.

Take a look at the shared business rule that has been generated. Open Visual Studio, and navigate to App_Code/Rules/SharedBusinessRules.cs. You can write some code to implement some global logging. This code will monitor and log all updates. The code is displayed below.

```
namespace MyCompany.Rules
{
    public partial class SharedBusinessRules : MyCompany.Data.BusinessRules
    {
    }

    protected override void AfterSqlAction(ActionArgs args, ActionResult result)
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine();
        sb.AppendFormat(@"Controller: {0}, View: {1}, Command: {2}",
            args.Controller, args.View, args.CommandName);
        sb.AppendLine();
        switch (args.CommandName)
        {
            case "Update":
                foreach (FieldValue v in args.Values)
                {
                    if (v.Modified)
                    {
                        sb.AppendFormat(
                            "Field '{0}' has been changed from '{1}' to '{2}'",
                            v.Name, v.OldValue, v.NewValue);
                        sb.AppendLine();
                    }
                }
                break;
        }
        System.Diagnostics.Debug.WriteLine(sb.ToString());
    }
}
```

Let's see this code in action. Modify any record in the database application, and look at the Output field. You can see that a log of this change has been added.



```
Output
Show output from: Debug
Field 'CompanyName' has been changed from 'Berglunds snabbköp***' to 'Berglunds snabbköp'

Controller: Customers, View: grid1, Command: Update
Field 'CompanyName' has been changed from 'Around the Horn' to 'Around the Horn***'
```

Code OnTime LLC
<http://www.codeontime.com>