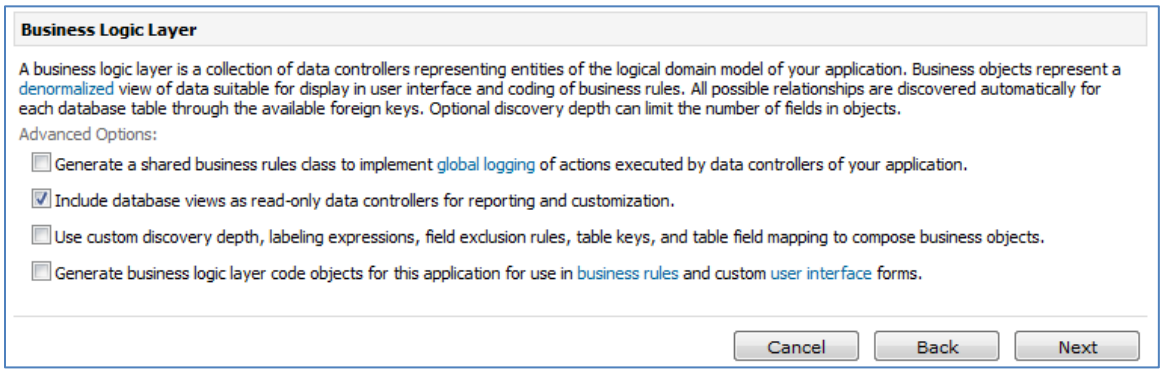
2010

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.



Views

The code generator will by default automatically include all database views in your project. Thee 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*.

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

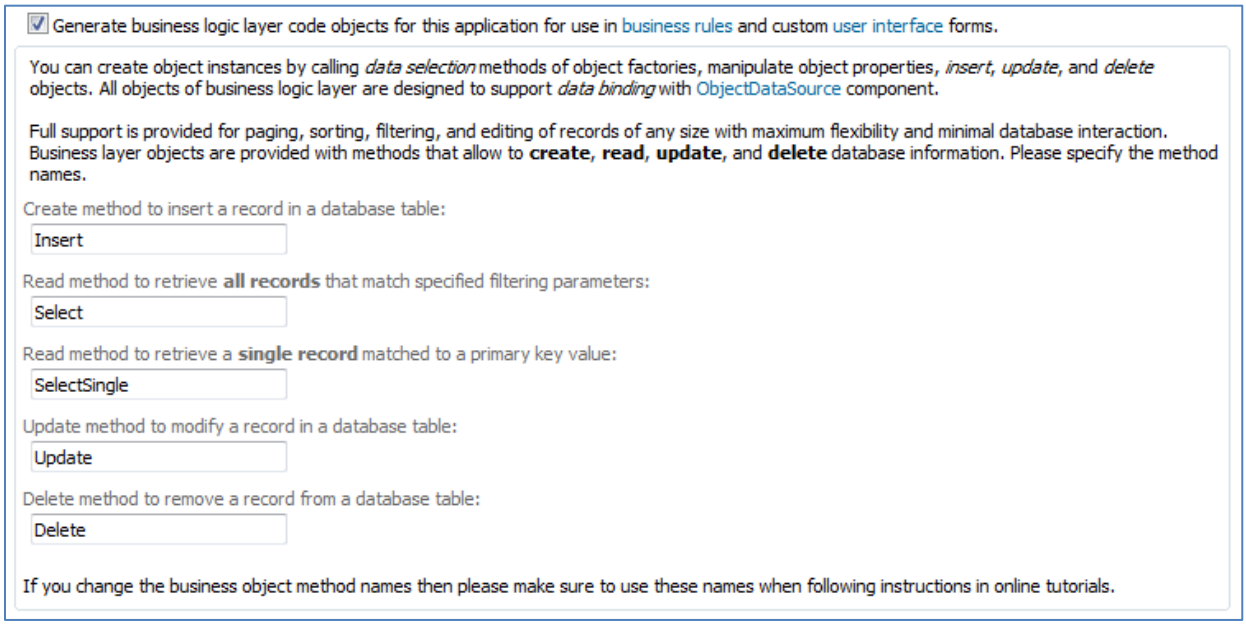
Normally, data controllers based on views do not permit data modification. If you explicitly specify a primary key, then you will gain the ability to select data. If you choose a physical table to save to, then you will be able to update and create data. You can also make stored procedures to update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

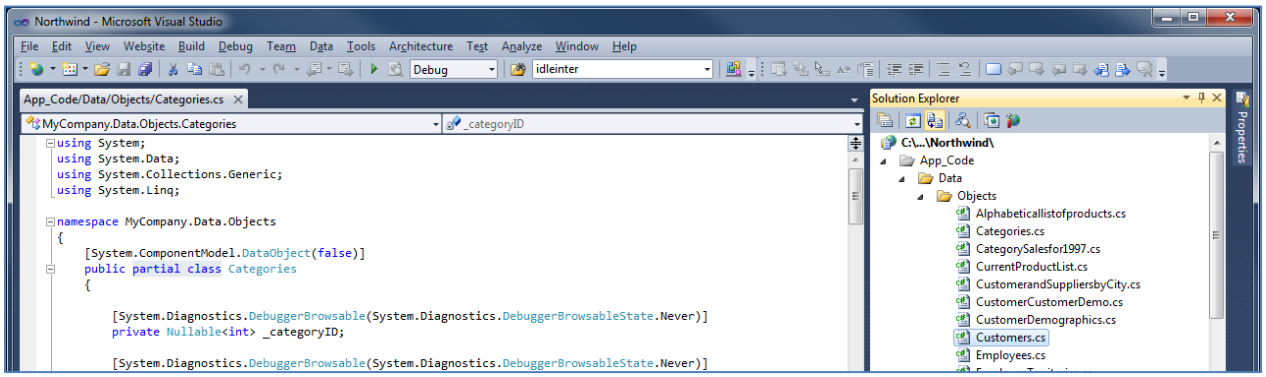
When you activate code objects, you have the option of specifying different names for the CRUD object methods. By default, method Insert allows creation of data. Method Select and SelectSingle allow the reading of data. Method update updates the data, and method delete removes the data from the database.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



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.

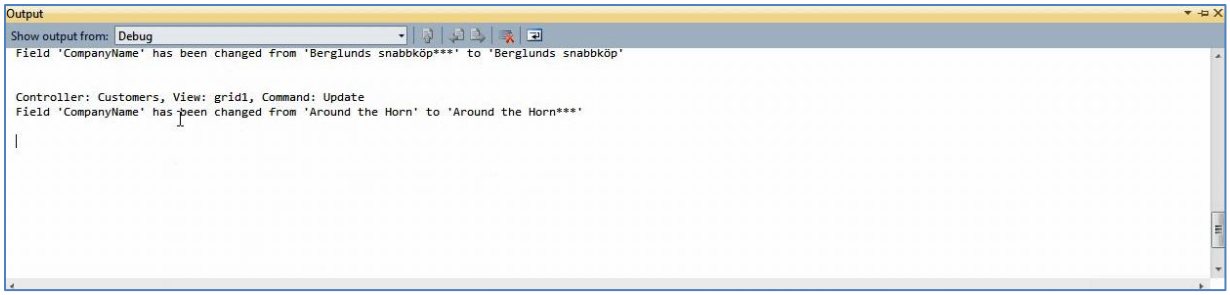
Take a look at the shared business rule that has been generated. Open Visual Studio, and navigate to App_Code/Rules/SharedBusinessRules.cs. You can write some code to implement some global logging. This code will monitor and log all updates. The code is displayed below.

```
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.



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