

## **How to develop ASP.NET 2.0 Application using Provider Design Pattern**

Provider Design Pattern is a new pattern that Microsoft formalized it in ASP.NET Whidbey. The pattern was officially named in the summer of 2002 when Microsoft was designing the new Personalization feature of ASP.NET Whidbey.

### **Benefits:**

1- We won't explicitly instantiate classes. We let .NET framework take care of instantiating our classes. Framework will be responsible and will manage class instantiation. Framework will re-use classes that have already been instantiated. This will have great effect on memory management of your application.

2- If for some reason you want to change the source of data for your application, for example, moving your database from SQL server to Oracle OR vice versa OR changing your SQL database server to some sort of XML data source, you will have very easy time to implement this requirement. All you have to do is to replace your existing concrete (implementer) class with a new concrete (implementer) class and inherit from your provider class. That's all. Your presentation and business logic layer will be kept intact. You don't have to make any changes into your presentation and business logic layers.

3- Learning Provider Design concept will make it very easy to customize built-in .NET framework providers.

Note: Before we start, I would like to remind you that please use the same exact naming convention and names that I used in this document to develop your solution. Once you build your first application based on this new model and got the whole picture, then you can use your own naming convention and naming.

Our solution will have 3 projects:

ASP.NET project (**agds**)

Business Logic project (**BusinessLogicLayer**)

Data Access project (**DataAccessLayer**)

Go to Visual Studio 2005 start page choose "Create Website" and name the project **agds**. Go back to start page and choose "Create Project", choose "Class library" name the project **BusinessLogicLayer** and make sure choose add into existing solution. Go back to start page and choose "Create Project", choose "Class library" name the project **DataAccessLayer** and make sure choose add into existing solution.

Right Click **agds** project and choose add reference, choose "Projects" tab and select "**BusinessLogicLayer**" and click add.

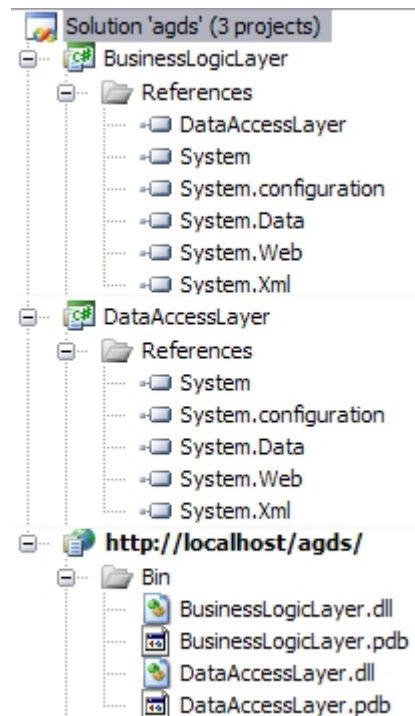
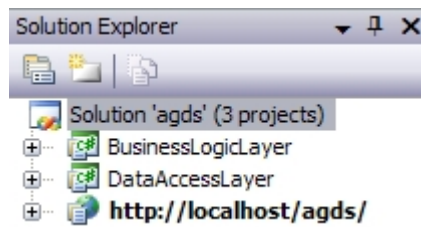
Right Click **BusinessLogicLayer** project and choose add reference, choose "Projects" tab and select "**DataAccessLayer**" and click add.

We need to add reference to below framework dlls for both **BusinessLogicLayer** and **DataAccessLayer** as well.

- System.Web

- System Configuration

Right click each of above mentioned project right click project, choose add reference and choose .NET tab. Find above two system dlls add them to above projects.



We will start setting up **Web.config** first:

ASP.NET 2.0 has defined a bunch of new XML nodes to make reading of particular section or group of nodes within config file easy, through a special class called "ConfigurationSection". *Pay attention to name attributes and correspondence nodes with similar color.*

```
<configuration>
  <configSections>
    <sectionGroup name="GroupSection">
      <section name="General" type="" />
      <section name="" type="" />
    </sectionGroup>
  </configSections>

  <GroupSection>
    <General>
      <providers>
        <add name="" type="" connectionStringName="SqlConnection" />
      </providers>
    </General>
  </GroupSection>

  <connectionStrings>
    <add name="SqlConnection" connectionString="sql_connection_string" />
  </connectionStrings>
</configuration>
```

Open **agds** project add web.config file and replace the content with below:

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <sectionGroup name="GroupSection">
      <section name="General"
        type="DataAccessLayer.SectionConfig, DataAccessLayer" />
    </sectionGroup>
  </configSections>

  <GroupSection>
    <General>
      <providers>
        <add name="SqlGeneral"
          type="DataAccessLayer.SqlGeneralProvider, DataAccessLayer"
          connectionStringName="SqlConnection" />
      </providers>
    </General>
  </GroupSection>

  <connectionStrings>
    <add name="SqlConnection"
      connectionString="sql_connection_string" />
  </connectionStrings>

  <appSettings/>

  <system.web>
    <compilation debug="true"/>
  </system.web>
</configuration>
```

We will explain web.config later.

We need to create a class that inherits from "ConfigurationSection" in order to be able to read web.config settings.

Note: From now on we will focus on **DataAccessLayer** project to complete our provider model.

Open **DataAccessLayer** project, delete the default class1 and add a new class file and name it "SectionConfig.cs". Add below code into this class:

```
using System;
using System.Configuration;
namespace DataAccessLayer
{
  public class SectionConfig : ConfigurationSection
  {
    [ConfigurationProperty("providers")]
    public ProviderSettingsCollection Providers{ get{return
(ProviderSettingsCollection)base["providers"];}}
  }
}
```

Above code reads all providers defined in your web.config.

That's all you have to do to make providers information available to other classes.

We need to create another class to have access to Framework provider collection and add our new provider(s) to provider collection.

Add a new class and name it "ProviderList.cs". Add below code into this class:

```
using System;
using System.Configuration.Provider;
namespace DataAccessLayer
{
    public class ProviderList : ProviderCollection
    {
        public override void Add(ProviderBase provider)
        {
            if (provider == null) throw new ArgumentNullException("The provider
parameter cannot be null.");
            base.Add(provider);
        }
    }
}
```

Now our **DataAccessLayer** project has all necessary classes for all providers to be developed later.

Here is the blue print on how we create our provider model:

**BaseProvider**-->**xxxProvider**-->**SQLxxxProvider**

xxx is the name of entity.

For example:

**CommerceProvider**-->**SQLCommerceProvider**

Note: CommerceProvider inherits from BaseProvider.

Suppose **agds** is a midsize business and **agds** Website contains 3 major sections:

- 1- **General Activities**: Saving Website's various general forms into database, displaying general data through various pages, and extra...
- 2- **eCommerce**: Selling product online.
- 3- **Customer Service**: Tagging customer inquires regarding eCommerce products, delegate inquires to proper departments and responding back to customer inquires and whatever else Customer Service needs are.

We are going to develop:

- A provider for **General** Activities.
- A provider for **Commerce** which has two implementations: One implementation for using SQL server database and another implementation for using Oracle database.
- A provider for **Customer Service**.

We will start with **General Activities** provider.

Add a new class and name it "GeneralProvider.cs". Add below code into this class.

```
using System;
using System.Configuration.Provider;
using System.Configuration;
using System.Web.Configuration;
namespace DataAccessLayer
{
    public class InitGeneral
    {
```

```

        protected static bool isInitialized = false;
        static InitGeneral()
        {
            Initialize();
        }
        private static void Initialize()
        {
            SectionConfig qc =
(SectionConfig)ConfigurationManager.GetSection("GroupSection/General");
            providerCollection = new ProviderList();
            ProvidersHelper.InstantiateProviders(qc.Providers,
providerCollection, typeof(GeneralProvider));
            providerCollection.SetReadOnly();
            isInitialized = true; //error-free initialization
        }

        //Public feature API
        private static ProviderList providerCollection;
        public static ProviderList Providers { get { return
providerCollection; } }
    }

    ////////////////////////////////////////
    // Define all methods below
    public abstract class GeneralProvider : ProviderBase
    {
        public abstract string GetDataGeneral();
    }
}

```

As you can see above "GeneralProvider.cs" contains two classes. "InitGeneral" and "GeneralProvider" abstract class which inherits from ProviderBase.

"InitGeneral" is responsible for instantiating our concrete (implementer) class (SqlGeneralProvider.cs") which has been defined within web.config.

```

(SectionConfig)ConfigurationManager.GetSection("GroupSection/General");
    providerCollection = new ProviderList();
    ProvidersHelper.InstantiateProviders(qc.Providers,
providerCollection, typeof(GeneralProvider));

```

Above code use "SectionConfig" class to read all providers define within web.config.

```

GetSection("GroupSection/General");

```

Above line returns all providers defined within <General></General> node of web.config, add them into provider collection, and instantiate our "GeneralProvider.cs" class which inherited by "SqlGeneralProvider.cs" class. Here is the web.config:

```

<configSections>
    <sectionGroup name="GroupSection">
        <section name="General"
            type="DataAccessLayer.SectionConfig, DataAccessLayer" />
    </sectionGroup>
</configSections>

```

```

        <!--above we are telling the application to look for config
        reader class within "DataAccessLayer.SectionConfig" name
        space.-->
    </sectionGroup>
</configSections>

<GroupSection>
    <General>
        <providers>
            <add name="SqlGeneral"
                type="DataAccessLayer.SqlGeneralProvider, DataAccessLayer"
                connectionStringName="SqlConnection" />
            <!--above we have given a name to our provider, we've defined
            our concrete class and the dll to which the concrete class
            resides and we have defined the connection string name-->
        </providers>
    </General>
</GroupSection>

////////////////////////////////////
// Define all methods below
public abstract class GeneralProvider : ProviderBase
{
    public abstract string GetDataGeneral();
}

```

Above, "GeneralProvider" abstract class is the class where we will define all abstract methods to be used/overridden by "SqlGeneralProvider.cs" class.

Add a new class file and name it "SqlGeneralProvider.cs". Add below code into this class.

```

using System;
using System.Configuration;
using System.Configuration.Provider;
namespace DataAccessLayer
{
    public class SqlGeneralProvider : GeneralProvider
    {
        private String connectionString;
        public override void Initialize(string name,
System.Collections.Specialized.NameValueCollection config)
        {
            //Let ProviderBase perform the basic initialization
            base.Initialize(name, config);
            string connectionStringName = config["connectionStringName"];
            ConnectionStringsSection cs =
(ConnectionStringsSection)ConfigurationManager.GetSection("connectionStrings")
;
            connectionString =
cs.ConnectionStrings[connectionStringName].ConnectionString;
            config.Remove("connectionStringName");
        }
    }
}
////////////////////////////////////

```

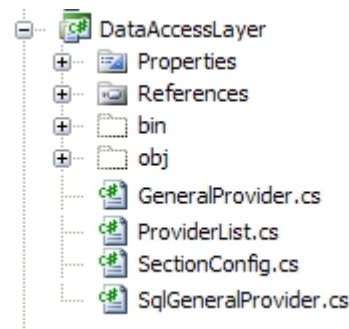
```

// Implement all methods below
public override string GetDataGeneral()
{
    //for simplicity to understand provider model better, we don't
    // access database and return result.
    return "SQL Data for GENERAL" + "<br> " + connectionString;
}
}
}

```

Note: "private String connectionString" reads connection string for this particular provider and makes it available for accessing databases to methods of this class.

Here how your **DataAccessLayer** project should look like this when you are done.



Next, we will create **Commerce** provider which has two implementations: One implementation for using SQL server database and another implementation for using Oracle database. Let's update web.config first:

Add below node within <sectionGroup name="GroupSection"> </sectionGroup>

```

<section name="Commerce"
        type="DataAccessLayer.SectionConfig, DataAccessLayer" />

```

Add below nodes within <GroupSection> </GroupSection>

```

<Commerce>
  <providers>
    <add name="SqlCommerce"
        type="DataAccessLayer.SqlCommerceProvider, DataAccessLayer"
        connectionStringName="SqlConnection" />
    <add name="OracleCommerce"
        type="DataAccessLayer.OracleCommerceProvider, DataAccessLayer"
        connectionStringName="OracleConnection" />
  </providers>
</Commerce>

```

Open **DataAccessLayer** project, copy "GeneralProvider.cs" and paste back into this project, and re-name the copy to "CommerceProvider.cs". Double click "CommerceProvider.cs" and change the followings:

```

public class InitGeneral TO public class InitCommerce

GetSection("GroupSection/General") TO GetSection("GroupSection/Commerce")

ProvidersHelper.InstantiateProviders(qc.Providers, providerCollection,
typeof(GeneralProvider)); TO
ProvidersHelper.InstantiateProviders(qc.Providers, providerCollection,
typeof(CommerceProvider));

public abstract class GeneralProvider : ProviderBase TO
public abstract class CommerceProvider : ProviderBase

public abstract string GetDataGeneral() TO
public abstract string GetDataCommerce()
You are done here.

```

**Commerce SQL implementation:**

Copy "SqlGeneralProvider.cs" and paste it back into this project, and re-name it to "SqlCommerceProvider.cs". Double click on "SqlCommerceProvider.cs" and change the followings:

```

public class SqlGeneralProvider : GeneralProvider TO
public class SqlCommerceProvider : CommerceProvider

public override string GetDataGeneral() TO
public override string GetDataCommerce()

return "SQL Data for GENERAL" + "<br> " + connectionString TO
return "SQL Data for COMMERCE" + "<br> " + connectionString
You are done here.

```

**Commerce Oracle implementation:**

Copy "SqlGeneralProvider.cs" and paste it back into this project, and re-name it to "OracleCommerceProvider.cs". Double click on "OracleCommerceProvider.cs" and change the followings:

```

public class SqlGeneralProvider : GeneralProvider TO
public class OracleCommerceProvider : CommerceProvider
public override string GetDataGeneral() TO
public override string GetDataCommerce()
return "SQL Data for GENERAL" + "<br> " + connectionString TO
return "ORACLE Data for COMMERCE" + "<br> " + connectionString
You are done here.

```

Now we create our last provider **Customer Service**. Update web.config as follow:  
Add below node within <sectionGroup name="GroupSection"> </sectionGroup>

```

<section name="CustomerService"
        type="DataAccessLayer.SectionConfig, DataAccessLayer" />

```

Add below nodes within <GroupSection> </GroupSection>

```

<CustomerService>
  <providers>

```

```

        <add name="SqlCustomerService"
            type="DataAccessLayer.SqlCustSveProvider, DataAccessLayer"
            connectionStringName="SqlConnection" />
    </providers>
</CustomerService>

```

Copy "GeneralProvider.cs" and paste back into this project, and re-name the copy to "CustSveProvider.cs". Double click "CustSveProvider.cs" and change the followings:

```
public class InitGeneral TO public class InitCustomerService
```

```
GetSection("GroupSection/General") TO
GetSection("GroupSection/CustomerService")
```

```
ProvidersHelper.InstantiateProviders(qc.Providers, providerCollection,
    typeof(GeneralProvider)); TO
ProvidersHelper.InstantiateProviders(qc.Providers, providerCollection,
    typeof(CustSveProvider));
```

```
public abstract class GeneralProvider : ProviderBase TO
public abstract class CustSveProvider : ProviderBase
```

```
public abstract string GetDataGeneral() TO
public abstract string GetDataSqlCustSve()
```

You are done here.

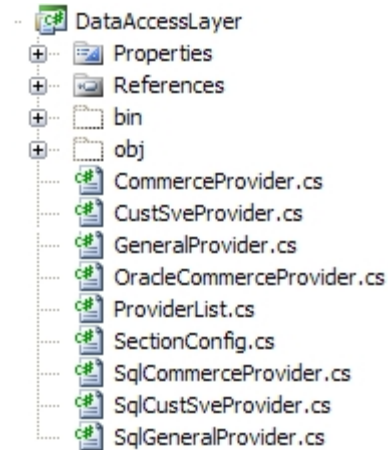
Copy "SqlGeneralProvider.cs" and paste it back into this project, and re-name it to "SqlCustSveProvider.cs". Double click on "SqlCustSveProvider.cs" and change the followings:

```
public class SqlGeneralProvider : GeneralProvider TO
public class SqlCustSveProvider : CustSveProvider
```

```
public override string GetDataGeneral() TO
public override string GetDataSqlCustSve()
```

```
return "SQL Data for GENERAL" + "<br> " + connectionString TO
return "SQL Data for CUSTOMER SERVICE" + "<br> " + connectionString
You are done here.
```

Your **DataAccessLayer** project should look like this:



Your complete web.config should like below:

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <sectionGroup name="GroupSection">
      <section name="General"
        type="DataAccessLayer.SectionConfig, DataAccessLayer" />
      <section name="Commerce"
        type="DataAccessLayer.SectionConfig, DataAccessLayer" />
      <section name="CustomerService"
        type="DataAccessLayer.SectionConfig, DataAccessLayer" />
    </sectionGroup>
  </configSections>

  <GroupSection>
    <General>
      <providers>
        <add name="SqlGeneral"
          type="DataAccessLayer.SqlGeneralProvider, DataAccessLayer"
          connectionStringName="SqlConnection" />
      </providers>
    </General>
    <Commerce>
      <providers>
        <add name="SqlCommerce"
          type="DataAccessLayer.SqlCommerceProvider, DataAccessLayer"
          connectionStringName="SqlConnection" />
        <add name="OracleCommerce"
          type="DataAccessLayer.OracleCommerceProvider, DataAccessLayer"
          connectionStringName="OracleConnection" />
      </providers>
    </Commerce>
    <CustomerService>
      <providers>
        <add name="SqlCustomerService"
          type="DataAccessLayer.SqlCustSveProvider, DataAccessLayer"
          connectionStringName="SqlConnection" />
      </providers>
    </CustomerService>
  </GroupSection>
</configuration>
```

```

    </CustomerService>
</GroupSection>

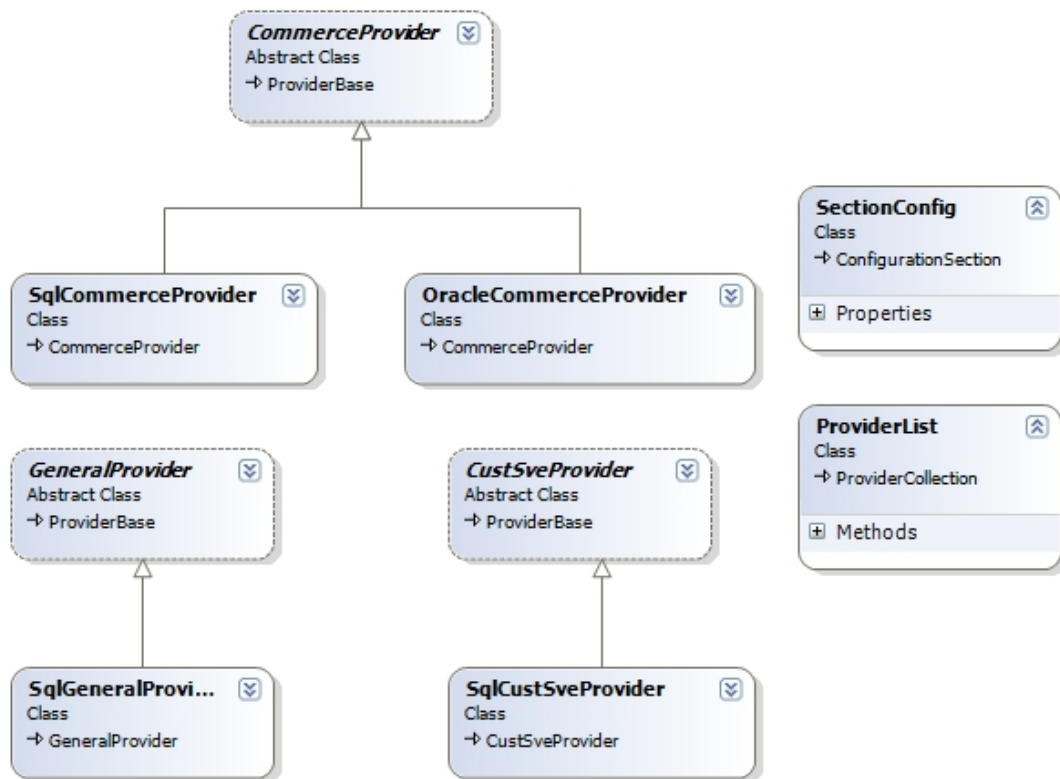
<connectionStrings>
  <add name="SqlConnection"
    connectionString="sql_connection_string" />
  <add name="OracleConnection"
    connectionString="oracle_connection_string" />
</connectionStrings>

<appSettings/>

<system.web>
  <compilation debug="true"/>
</system.web>
</configuration>

```

**DataAccessLayer** class diagram:



Next, we will develop our **BusinessLogicLayer**.

Open **BusinessLogicLayer** delete the default class1 and add three class files, name them "**General.cs**", "**SqlCommerce.cs**" "**OracleCommerce.cs**", "**CustomerService.cs**".

Note: It is a good idea to add a helper class into your **BusinessLogicLayer** project. This way, you can expose some common functionality to all of your **BusinessLogicLayer** classes just by inheriting from this helper class.

To illustrate helper class, create another class and name it "Helper.cs".

Open "Helper.cs" and add below code into this class:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;

namespace BusinessLogicLayer
{
    public abstract class Helper
    {
        protected static string CurrentUserIP{get{ return
HttpContext.Current.Request.UserHostAddress; }}
    }

    // Add more methods/properties below
}
```

Our Helper class returns one property, user IP address.

Open "General.cs" and add below into this class:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BusinessLogicLayer
{
    public abstract class General:Helper
    {
        public static string GetGeneralData()
        {
            // Here is how we access this provider
            DataAccessLayer.GeneralProvider SqlGeneral =
(DataAccessLayer.GeneralProvider)DataAccessLayer.InitGeneral.Providers["SqlGen
eral"];
            // You can use helper to provide common info./data needed OR to
            // message or add more info. to your data before sending it to
            // presentation.
            // Here we use helper class to get CurrentUserIP and pass it along
            // with data to presentation.
            return SqlGeneral.GetDataGeneral() + "<br> Current IP address: " +
CurrentUserIP.ToString();
        }
    }
}
```

Open "SqlCommerce.cs" and add below into this class:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BusinessLogicLayer
{
    public abstract class SqlCommerce : Helper
    {

```

```

        public static string GetCommerce()
        {
            // Here is how we access this provider
            DataAccessLayer.SqlCommerceProvider SqlCommerce =
            (DataAccessLayer.SqlCommerceProvider)DataAccessLayer.InitCommerce.Providers["S
            qlCommerce"];
            return SqlCommerce.GetDataCommerce();
        }
    }
}

```

Open "OracleCommerce.cs" and add below into this class:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace BusinessLogicLayer
{
    public abstract class OracleCommerce : Helper
    {
        public static string GetCommerce()
        {
            DataAccessLayer.OracleCommerceProvider oracle =
            (DataAccessLayer.OracleCommerceProvider)DataAccessLayer.InitCommerce.Providers
            ["OracleCommerce"];
            return oracle.GetDataCommerce();
        }
    }
}

```

Open "CustomerService.cs" and add below into this class:

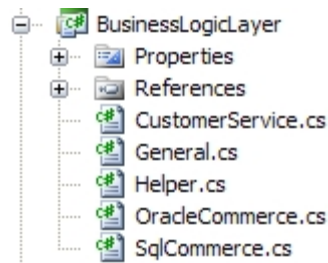
```

using System;
using System.Collections.Generic;
using System.Text;

namespace BusinessLogicLayer
{
    public abstract class CustomerService : Helper
    {
        public static string GetCustomerServiceData()
        {
            // Here is how we access this provider
            DataAccessLayer.SqlCustSveProvider CustomerService =
            (DataAccessLayer.SqlCustSveProvider)DataAccessLayer.InitCustomerService.Provid
            ers["SqlCustomerService"];
            return CustomerService.GetDataSqlCustSve();
        }
    }
}

```

We are done with **BusinessLogicLayer**.  
Your project should look like this:



Open "agds" project, double click on default.aspx, and add below code (HTML view).

```
<asp:Button ID="btnGeneral" runat="server" Text="General Data"
OnClick="btnGeneral_Click" />
    <asp:Button ID="btnSQLCommerce" runat="server"
OnClick="btnSQLCommerce_Click" Text="Commerce SQL" />
    <asp:Button ID="btnComerceOracle" runat="server"
OnClick="btnComerceOracle_Click"
Text="Comerce Oracle" />
    <asp:Button ID="btnCustSve" runat="server" OnClick="btnForum_Click"
Text="Customer Service" />&nbsp; <asp:Button
ID="btnClear" runat="server" OnClick="btnClear_Click" Text="Clear"
/><br />
    <br />

    <asp:Label ID="lb11" runat="server"></asp:Label><br /><br />
    <asp:Label ID="lb12" runat="server"></asp:Label><br /><br />
    <asp:Label ID="lb13" runat="server"></asp:Label><br /><br />
    <asp:Label ID="lb14" runat="server"></asp:Label>
```

Go to code behind (Code Beside?) and add below code:

```
protected void btnGeneral_Click(object sender, EventArgs e)
{
    lb11.Text = BusinessLogicLayer.General.GetGeneralData();
}
protected void btnSQLCommerce_Click(object sender, EventArgs e)
{
    lb12.Text = BusinessLogicLayer.SqlCommerce.GetCommerce();
}
protected void btnComerceOracle_Click(object sender, EventArgs e)
{
    lb13.Text = BusinessLogicLayer.OracleCommerce.GetCommerce();
}
protected void btnForum_Click(object sender, EventArgs e)
{
    lb14.Text =
BusinessLogicLayer.CustomerService.GetCustomerServiceData();
}
protected void btnClear_Click(object sender, EventArgs e)
{
    lb11.Text =string.Empty;
    lb12.Text =string.Empty;
```

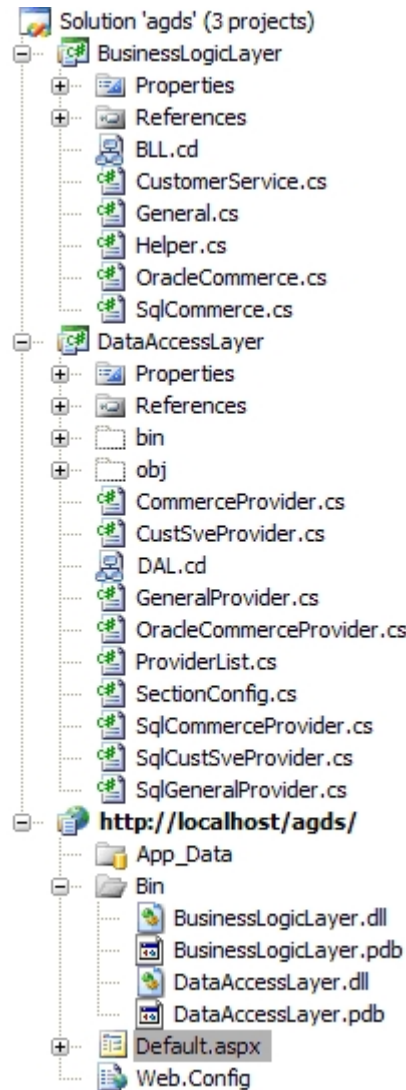
```

    lbl3.Text = string.Empty;
    lbl4.Text = string.Empty;
}

```

Compile your project.

Here is how your complete solution should look like:



**Author: Abdul Meraj**

Abdul Meraj is a certified .NET application developer with 8 years of experience in web development. He resides and works as a contractor in Orange County California. During 8 years, he was involved in developing web application projects for companies like Toshiba, Mazda, Allergan, and Western Digital. He can be reached at: [studio1@agdstudio.com](mailto:studio1@agdstudio.com).

If you would like to share your thoughts please visit:

<http://www.agdstudio.com/developers/forum.aspx>