Bill Pegram
12/16/2011

# Constructing a Multi-Tier Application in ASP.NET

The references provide different approaches to constructing a multi-tier application in ASP.NET.

* A “connected model ” where there is an open connection to the database vs. a “disconnected model” where an in-memory representation of the data is used. According to Walter et al (p. 889), while the latter provides more functionality, it runs slower. Since I will be constructing a very simple example, I will use the former.
* Extent to which Visual Studio tools and ASP.NET objects are used – Below I use Visual Studio and write less code than other approaches might use.

The steps outlined below sketch one incomplete approach to the topic.

Steps:

* Create the database – To create a new SQL Server Express Edition database with Visual Web Developer or Visual Studio, create a new website using the Empty ASP.NET template. To add a database, go to Solution Explorer, right click on the website name, choose Add New>SQL Server Database. You can respond yes about putting the database in the App\_Data folder.
* Creating Database Tables – In VWD or Visual Studio, using Database Explorer, right click the Tables folder and choose the Add a New Table option. Enter information for the field name, datatype, and Allow Null columns.
	+ To designate a field as the primary key, select the row showing the appropriate field name and then click the primary key icon in the toolbar or right click the row and choose Set Primary Key.
	+ To mark a field as auto-increment, select the appropriate row column and then in the Column Properties at the bottom, find the Identify Specification property, expand it and change the (Is Identity) option to Yes.

For this example, we will have a single table named Student and three fields –

* Id – int – primary key, auto-increment
* student – varchar(50)
* score – int

In this example, I will uncheck the “allow nulls” for all three fields.

* Adding Data to the Tables - Right click on the table in the Database Explorer and select the Show Table Data option and enter the data. Where you see red exclamation point icons, the record has yet to be created so click in the next record or Tab to it, to save the record.
* Adding a New DataSet to the Project – Right click on the project node in the Solution Explorer and choose Add a New Item. Select the DataSet option and give it a suitable name. When prompted about whether to add it to the App\_Code folder, choose Yes.
* Adding a TableAdapter to the DataSet – Right click>Add>TableAdapter. The TableAdapter Configuration Wizard begins by prompting you to select the database – choose it from the dropdown menu. After clicking Next, you will be asked if you want to save the connectionStinrg in the web.config file, and choose yes. You will then see the following screen and choose the “Use SQL statements” default.



You will then see the following screen where you can either type in a SQL statement or use the Query Builder to build it.



The following screen shows selecting all columns using the \* in the Query Builder



After the Query is built, click the Advanced Options button at the lower left and you will see the following screen:



And accept the defaults and clicking Next will take you to the following screen:



To keep the code simple for this example, I will uncheck the first and third boxes so as to only return a DataTable – you can rename this method to make it more specific.

Creating a Business Logic Layer

Rather than implementing the Business Logic Layer and Presentation layers as separate Class Library projects, we’ll implement these as classes in our project. To make things clearer, we will create separate folders – so right click on the App\_Code folder and add two new folders – DLL r and BLL. Drag the TypedDataset previously created into the DataAccess folder.

Right click on the BLL folder, choose New Item, and choose Class. We will use a separate class for each table in our database – data access layer (in this case, there is only one). Ultimately we will need four methods:

* + GetStudents() – return all students
	+ AddStudent(student, score) – inserts a new student into the database using these values
	+ UpdateStudent(student, score, id) – updates the specified record
	+ DeleteStudent(id) – deleted the specified record

but for now, we’ll just do the GetStudents() method.

We’ll begin with the following code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using MultiTierTableAdapters; //MultiTier is the name of the database

public class StudentBLL {

 private StudentTableAdapter \_studentAdapter = null; // Student is the table name

 public StudentTableAdapter Adapter // this is the property corresponding to the

 // fieldname \_studentAdapter

{

 get { return \_studentAdapter;

 }

}

 public MultiTier.StudentDataTable GetData() {

 return Adapter.GetData(); // this is the GetData method of the TableAdapter

 // it returns a MultiTier.StudentDataTable

 }

} // end class

At this point, there is no business logic, but for now, we will move on to the presentation layer.

If you do not have a Web folder, create one, and add a new Web form to this folder and switch to Design View. Drag in an instance of the ObjectDataSource from the toolbar, and click on the SmartTag and choose Configure Data Source. The dropdown list will likely only show the TableAdapters in the Typed Data Set (our Data Access Layer) and will not show the Business Object classes. To enable these to show up in the drop list, we will modify our previous code in the Business Logic layer by adding DataObject attributes as follows;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using MultiTierTableAdapters;

[System.ComponentModel.DataObject]

public class StudentBLL {

 private StudentTableAdapter \_studentAdapter = null;

 public StudentTableAdapter Adapter

{

 get { return \_studentAdapter;

 }

}

 public MultiTier.StudentDataTable GetData() {

 return Adapter.GetData();

 }

} // end class

Adding the DataObject attribute to the class enables the class to show up in the dropdown list – now if we come back to the Configure Data Source, we will see the Business Logic – choose this and then on the next screen you can choose the method, in this case GetData (since we only have one). and click through to Finish.

This will add code such as the following to your web page:

<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"

 OldValuesParameterFormatString="original\_{0}" SelectMethod="GetData"

 TypeName="StudentBLL"></asp:ObjectDataSource>

where StudentBLL is the name of the class and GetData the method in this class.

-----------------------------

Adding a Data Web Control and Binding it to the ObjectDataSource – Add a GridView and from its Smart Tag, choose the ObjectDataSource added just now. Right click on the page and choose View in Browser. You will likely get an NullReferenceException saying that the object reference is not set to an instance of an object, concerning the following two lines:

public MultiTier.StudentDataTable GetData() {

 return Adapter.GetData(); }

The problem seems to be that we have never created the StudentTableAdapter object. To remedy this, we can test to see if the value is null, and if so, call the constructor to create the object, as shown in the following code (as shown in the Mitchell tutorial). If one makes this change, the data should now display in the page and thus we have a simple, three-tier application although there are no validation rules yet in the data access or business logic layers.

-----------------------------------------------------------

**Student.cs (in the App\_Code/BLL folder)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using MultiTierTableAdapters;

[System.ComponentModel.DataObject]

public class StudentBLL {

 private StudentTableAdapter \_studentAdapter = null;

 public StudentTableAdapter Adapter

{

 get

 {

 if (\_studentAdapter == null) \_studentAdapter = new StudentTableAdapter();

 return \_studentAdapter;

 }

}

 public MultiTier.StudentDataTable GetData() {

 return Adapter.GetData();

 }

} // end class

**SimpleDisplay.aspx (presentation layer)**

The ObjectDataSource TypeName attribute is set equal to StudentBLL, the class in the BusinessLogic Layer and the SelectMethod attribute is set equal to GetData which Is the name of the method that returns a DataTable.

<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"

 OldValuesParameterFormatString="original\_{0}" SelectMethod="GetData"

 TypeName="StudentBLL"></asp:ObjectDataSource>

 <div>

 <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"

 DataKeyNames="id" DataSourceID="ObjectDataSource1">

 <Columns>

 <asp:BoundField DataField="id" HeaderText="id" InsertVisible="False"

 ReadOnly="True" SortExpression="id" />

 <asp:BoundField DataField="student" HeaderText="student"

 SortExpression="student" />

 <asp:BoundField DataField="score" HeaderText="score" SortExpression="score" />

 </Columns>

 </asp:GridView>

 </div>

Multitier.xsd (in App\_Code/DataAccess folder)

<?xml version="1.0" encoding="utf-8"?>

<xs:schema xmlns:mstns="http://tempuri.org/MultiTier.xsd" xmlns="http://tempuri.org/MultiTier.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:msprop="urn:schemas-microsoft-com:xml-msprop" targetNamespace="http://tempuri.org/MultiTier.xsd" elementFormDefault="qualified" attributeFormDefault="qualified" id="MultiTier">

 <xs:annotation>

 <xs:appinfo source="urn:schemas-microsoft-com:xml-msdatasource">

 <DataSource DefaultConnectionIndex="0" FunctionsComponentName="QueriesTableAdapter" Modifier="AutoLayout, AnsiClass, Class, Public" SchemaSerializationMode="IncludeSchema" xmlns="urn:schemas-microsoft-com:xml-msdatasource">

 <Connections>

 <Connection AppSettingsObjectName="Web.config" AppSettingsPropertyName="multitierConnectionString" ConnectionStringObject="" IsAppSettingsProperty="true" Modifier="Assembly" Name="multitierConnectionString (Web.config)" ParameterPrefix="@" PropertyReference="AppConfig.System.Configuration.ConfigurationManager.0.ConnectionStrings.multitierConnectionString.ConnectionString" Provider="System.Data.SqlClient"/>

 </Connections>

 <Tables>

 <TableAdapter BaseClass="System.ComponentModel.Component" DataAccessorModifier="AutoLayout, AnsiClass, Class, Public" DataAccessorName="StudentTableAdapter" GeneratorDataComponentClassName="StudentTableAdapter" Name="Student" UserDataComponentName="StudentTableAdapter">

 <MainSource>

 <DbSource ConnectionRef="multitierConnectionString (Web.config)" DbObjectName="dbo.Student" DbObjectType="Table" GenerateMethods="Get" GenerateShortCommands="false" GeneratorGetMethodName="GetData" GetMethodModifier="Public" GetMethodName="GetData" QueryType="Rowset" ScalarCallRetval="System.Object, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" UseOptimisticConcurrency="false" UserGetMethodName="GetData" UserSourceName="GetData">

 <DeleteCommand>

 <DbCommand CommandType="Text" ModifiedByUser="false">

 <CommandText>DELETE FROM [Student] WHERE (([id] = @Original\_id))</CommandText>

 <Parameters>

 <Parameter AllowDbNull="false" AutogeneratedName="" DataSourceName="" DbType="Int32" Direction="Input" ParameterName="@Original\_id" Precision="0" ProviderType="Int" Scale="0" Size="0" SourceColumn="id" SourceColumnNullMapping="false" SourceVersion="Original"/>

 </Parameters>

 </DbCommand>

 </DeleteCommand>

 <InsertCommand>

 <DbCommand CommandType="Text" ModifiedByUser="false">

 <CommandText>INSERT INTO [Student] ([student], [score]) VALUES (@student, @score)</CommandText>

 <Parameters>

 <Parameter AllowDbNull="false" AutogeneratedName="" DataSourceName="" DbType="AnsiString" Direction="Input" ParameterName="@student" Precision="0" ProviderType="VarChar" Scale="0" Size="0" SourceColumn="student" SourceColumnNullMapping="false" SourceVersion="Current"/>

 <Parameter AllowDbNull="false" AutogeneratedName="" DataSourceName="" DbType="Int32" Direction="Input" ParameterName="@score" Precision="0" ProviderType="Int" Scale="0" Size="0" SourceColumn="score" SourceColumnNullMapping="false" SourceVersion="Current"/>

 </Parameters>

 </DbCommand>

 </InsertCommand>

 <SelectCommand>

 <DbCommand CommandType="Text" ModifiedByUser="true">

 <CommandText>SELECT Student.\*

FROM Student</CommandText>

 <Parameters/>

 </DbCommand>

 </SelectCommand>

 <UpdateCommand>

 <DbCommand CommandType="Text" ModifiedByUser="false">

 <CommandText>UPDATE [Student] SET [student] = @student, [score] = @score WHERE (([id] = @Original\_id))</CommandText>

 <Parameters>

 <Parameter AllowDbNull="false" AutogeneratedName="" DataSourceName="" DbType="AnsiString" Direction="Input" ParameterName="@student" Precision="0" ProviderType="VarChar" Scale="0" Size="0" SourceColumn="student" SourceColumnNullMapping="false" SourceVersion="Current"/>

 <Parameter AllowDbNull="false" AutogeneratedName="" DataSourceName="" DbType="Int32" Direction="Input" ParameterName="@score" Precision="0" ProviderType="Int" Scale="0" Size="0" SourceColumn="score" SourceColumnNullMapping="false" SourceVersion="Current"/>

 <Parameter AllowDbNull="false" AutogeneratedName="" DataSourceName="" DbType="Int32" Direction="Input" ParameterName="@Original\_id" Precision="0" ProviderType="Int" Scale="0" Size="0" SourceColumn="id" SourceColumnNullMapping="false" SourceVersion="Original"/>

 </Parameters>

 </DbCommand>

 </UpdateCommand>

 </DbSource>

 </MainSource>

 <Mappings>

 <Mapping SourceColumn="id" DataSetColumn="id"/>

 <Mapping SourceColumn="student" DataSetColumn="student"/>

 <Mapping SourceColumn="score" DataSetColumn="score"/>

 </Mappings>

 <Sources/>

 </TableAdapter>

 </Tables>

 <Sources/>

 </DataSource>

 </xs:appinfo>

 </xs:annotation>

 <xs:element name="MultiTier" msdata:IsDataSet="true" msdata:UseCurrentLocale="true" msprop:Generator\_UserDSName="MultiTier" msprop:Generator\_DataSetName="MultiTier">

 <xs:complexType>

 <xs:choice minOccurs="0" maxOccurs="unbounded">

 <xs:element name="Student" msprop:Generator\_TableClassName="StudentDataTable" msprop:Generator\_TableVarName="tableStudent" msprop:Generator\_TablePropName="Student" msprop:Generator\_RowDeletingName="StudentRowDeleting" msprop:Generator\_UserTableName="Student" msprop:Generator\_RowChangingName="StudentRowChanging" msprop:Generator\_RowEvHandlerName="StudentRowChangeEventHandler" msprop:Generator\_RowDeletedName="StudentRowDeleted" msprop:Generator\_RowEvArgName="StudentRowChangeEvent" msprop:Generator\_RowChangedName="StudentRowChanged" msprop:Generator\_RowClassName="StudentRow">

 <xs:complexType>

 <xs:sequence>

 <xs:element name="id" type="xs:int" msdata:ReadOnly="true" msdata:AutoIncrement="true" msdata:AutoIncrementSeed="-1" msdata:AutoIncrementStep="-1" msprop:Generator\_ColumnVarNameInTable="columnid" msprop:Generator\_ColumnPropNameInRow="id" msprop:Generator\_ColumnPropNameInTable="idColumn" msprop:Generator\_UserColumnName="id"/>

 <xs:element name="student" msprop:Generator\_ColumnVarNameInTable="columnstudent" msprop:Generator\_ColumnPropNameInRow="student" msprop:Generator\_ColumnPropNameInTable="studentColumn" msprop:Generator\_UserColumnName="student">

 <xs:simpleType>

 <xs:restriction base="xs:string">

 <xs:maxLength value="50"/>

 </xs:restriction>

 </xs:simpleType>

 </xs:element>

 <xs:element name="score" type="xs:int" msprop:Generator\_ColumnVarNameInTable="columnscore" msprop:Generator\_ColumnPropNameInRow="score" msprop:Generator\_ColumnPropNameInTable="scoreColumn" msprop:Generator\_UserColumnName="score"/>

 </xs:sequence>

 </xs:complexType>

 </xs:element>

 </xs:choice>

 </xs:complexType>

 <xs:unique name="Constraint1" msdata:PrimaryKey="true">

 <xs:selector xpath=".//mstns:Student"/>

 <xs:field xpath="mstns:id"/>

 </xs:unique>

 </xs:element>

</xs:schema>

---------------------------------------------------------------------------

References

*ASP.NET 4 Unleashed* by Stephen Walther, Kevin Hoffman, Nate Dudek , SAMS, 2011, pp 757-770, 833-912

*Pro C# and the .NET Platform* by Andrew Troelsen, Apress, 2010, pp. 825-950

Tutorial 1:Creating a Data Access Layer, Scott Mitchell, June 2006 <http://msdn.microsoft.com/en-us/library/aa581776.aspx>

Tutorial 2: Creating a Business Logic Layer, Scott Mitchell, June 2006, at <http://msdn.microsoft.com/en-us/library/aa581779.aspx>

Tutorial 4: Displaying Data with the ObjectDataSource, Scott Mitchell, June 2006 at <http://msdn.microsoft.com/en-us/library/aa581783.aspx>