Consistent Page Layout with Master Pages

With most web sites, only part of the page changes when you go from one page to another. The parts that don't change usually include common regions like the header, a menu, and the footer. To create web pages with a consistent layout you need a way to define these relatively static regions in a single template file.

The biggest benefit of master pages is that they allow you to define the look and feel of all the pages in your site in a single location. This means that if you want to change the layout of your site — for instance if you want to move the menu from the left to the right — you only need to modify the master page and the pages based on this master will pick up the changes automatically.

To some extent, a master page looks like a normal ASPX page. It contains static HTML such as the <html>, <head>, and <body> tags, and it can also contain other HTML and ASP.NET Server Controls. Inside the master page, you set up the markup that you want to repeat on every page, like the general layout of the page and the menu. However, a master page is not a true ASPX page and cannot be requested in the browser directly; it only serves as the template that real web pages — called content pages — are based on. Instead of the @ Page directive, a master page uses a @ Master directive that identifies the file as a master page:

<%@ Master Language="C#" %>

Just like a normal ASPX page, a master page can have a Code Behind file, identified by its CodeFile and Inherits attributes:

<%@ Master Language="C#" CodeFile="MasterPage.master.cs" Inherits="Masterpages_MasterPage" %>

The page-specific content is then put inside an <asp:Content> control that points to the relevant ContentPlaceHolder:

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">

</asp:Content>

At runtime, when the page is requested, the markup from the master page and the content page are merged, processed, and sent to the browser. Figure 1 shows a diagram of the master page and the content page that result in the final page that is sent to the browser.

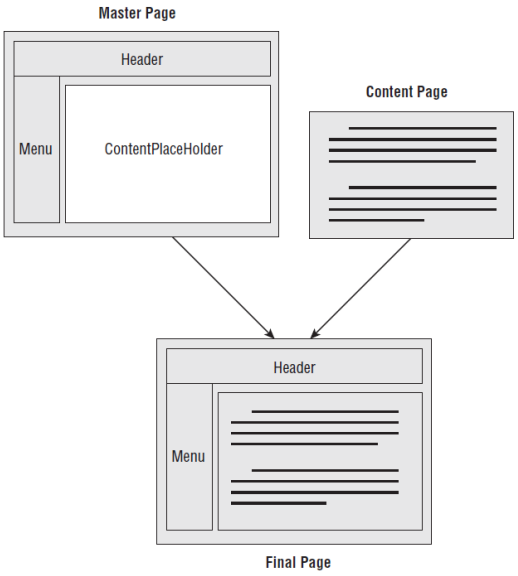

Figure 1: a diagram of the master page and the content page that result in the final page

Creating Master Pages

Master pages are added to the site using the Add New Item dialog box. They can be placed anywhere in the site, including the root folder, but from an organizational point of view, it's often easier to store them in a separate folder. Just like normal ASPX pages, they support the inline code model as well as the Code Behind model.

File→New→File→Master Page

Creating Content Pages

A master page is useless without a content page that uses it. Generally, you'll only have a few master pages, while you can have many content pages. To base a content page on a master page, you can check the option Select Master Page at the bottom right of the Add New Item dialog box.

ActiveX Data Objects.Net (ADO.NET)

Here are the objectives:

- Learn what ADO.NET is? Understand what a data provider is?
- Understand what a connection object is.
- Understand what a command object is? Understand what a DataReader object is?

ADO.NET is: an object-oriented set of libraries that allows you to interact with data sources. Commonly, the data source is a database, but it could also be a text file, an Excel spreadsheet, or an XML file. As you are probably aware, there are many different types of databases available. For example, there is Microsoft SQL Server, Microsoft Access, Oracle and Borland Interbase.

Data Providers: We know that ADO.NET allows us to interact with different types of data sources and different types of databases. However, there isn't a single set of classes that allow you to accomplish this universally. Since different data sources expose different protocols, we need a way to communicate with the right data source using the right protocol. Some older data sources use the ODBC protocol, many newer data sources use the OleDb protocol, and there are more data sources every day that allow you to communicate with them directly through .NET ADO.NET class libraries. ADO.NET provides a relatively common way to interact with data sources, but comes in different sets of libraries for each way you can talk to a data source. **These libraries are called Data Providers** and are usually named for the protocol or data source type they allow you to interact with. Table 1 lists some well known data providers, the API prefix they use, and the type of data source they allow you to interact with.

Provider Name	API prefix	Data Source Description
ODBC Data Provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDb Data Provider	OleDb	Data Sources that expose an OleDb interface, i.e. Access or Excel.
Oracle Data Provider	Oracle	For Oracle Databases.
SQL Data Provider	Sql	For interacting with Microsoft SQL Server.
Borland Data Provider	Bdp	Generic access to many databases such as Interbase, SQL Server, IBM DB2, and Oracle.

Table 1: ADO.NET Data Providers are class libraries that allow a common way to interact with specific data sources or protocols

One of the first ADO.NET objects you'll learn about is the connection object, which allows you to establish a connection to a data source. If we were using the OleDb Data Provider to connect to a data source that exposes an OleDb interface, we would use a connection object named OleDbConnection. Similarly, the connection object name would be prefixed with Odbc or Sql for an OdbcConnection object on an Odbc data source or a SqlConnection object on a SQL Server database, respectively. Now we will know the important ADO.NET Objects.

The SqlConnection Object

To interact with a database, you must have a connection to it. The connection helps identify the database server, the database name, user name, password, and other parameters that are required for connecting to the data base. A connection object is used by **command objects** so they will know which database to execute the command on. Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated Security=True;User Instance=True

Connection String Parameter Name	Description
Data Source	Identifies the server. Could be local machine, machine domain name, or IP Address.
Initial Catalog	Database name.
Integrated Security	to make connection with user's Windows login
User ID	Name of user configured in SQL Server.
Password	Password matching SQL Server User ID.

Table 2: ADO.NET Connection Strings contain certain key/value pairs for specifying how to make a database connection

The purpose of creating a SqlConnection object is so you can enable other ADO.NET code to work with a database. Other ADO.NET objects, such as a SqlCommand and a SqlDataAdapter take a connection object as a **parameter**. The sequence of operations occurring in the lifetime of a SqlConnection is as follows:

1. Instantiate the SqlConnection.
2. Open the connection.

3. Pass the connection to other ADO.NET objects.

4. Perform database operations with the other ADO.NET objects.

5. Close the connection.

We've already seen how to instantiate a SqlConnection. The rest of the steps, opening, passing, using, and closing are shown in Listing 1.

Listing 1: Using a SqlConnection

```
using System;
using System.Data;
using System.Data.SqlClient;
/// Demonstrates how to work with SqlConnection objects
protected void Button1_Click(object sender, EventArgs e)
    {
            // 1. Instantiate the connection
            SqlConnection conn = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated
Security=True;User Instance=True");
            SqlDataReader rdr = null;
        try
        {
            // 2. Open the connection
            conn.Open();
            // 3. Pass the connection to a command object
            SqlCommand cmd = new SqlCommand("select * from Customer", conn);
            // 4. Use the connection
            // get query results
            rdr = cmd.ExecuteReader();
            // print the CustomerID of each record
            while (rdr.Read())
            {
                Response.Write   (rdr[0]);
            }
        }
        finally
        {
            // close the reader
            if (rdr != null)
            {
                rdr.Close();
            }
            // 5. Close the connection
            if (conn != null)
            {
                conn.Close();
            }
        }

    }
```

As shown in Listing 1, you open a connection by calling the Open() method of the SqlConnection instance, conn. Any operations on a connection that was not yet opened will generate an exception. So, you must open the connection before using it. Before using a SqlCommand, you must let the ADO.NET code know which connection it needs. In Listing 1, we set the second parameter to the SqlCommand object with the SqlConnection object, conn. Any operations performed with the SqlCommand will use that connection. The code that uses the connection is a SqlCommand object, which performs a query on the Customers table. **The result set is returned as a SqlDataReader** and the while loop reads the first column from each row of the result set, which is the CustomerID column. When you are done using the connection object, you must close it. Failure to do so could have serious consequences in the performance and scalability of your application. There are a couple points to be made about how we closed the connection in Listing 1: the Close() method is called in a finally block and we ensure that the connection is not null before closing it. Notice that we wrapped the ADO.NET code in a try/finally block. **Finally blocks help guarantee that a certain piece of code will be executed**, regardless of whether or not an exception is generated. Since connections are scarce system resources, you will want to make sure they are closed in finally blocks. Another precaution you should take when closing connections is to make sure the connection object is not null. If something goes wrong when instantiating the connection, it will be null and you want to make sure you don't try to close an invalid connection, which would generate an exception. This example showed how to use a SqlConnection object with a SqlDataReader, which required explicitly closing the connection.

The SqlCommand Object

The process of interacting with a database means that you must specify the actions you want to occur. This is done with a command object. You use a command object to send SQL statements to the database. A command object uses a connection object to figure out which database to communicate with.

The SqlDataReader Object

Many data operations require that you only get a stream of data for reading. **The data reader object allows you to obtain the results of a SELECT statement from a command object.** For performance reasons, **the data returned from a data reader is a fast forward-only stream of data. This means that you can only pull the data from the stream in a sequential manner. This is good for speed**, but if you need to manipulate data, **then a DataSet is a better object** to work with.

ASP.NET includes data access tools that make it easier than ever for you to design sites that allow your users to interact with databases through Web pages. The .NET Framework includes two data providers for accessing enterprise databases: the .NET Framework Data Provider for OLE DB and the .NET Framework Data Provider for SQL Server.

Connect with SqlServer Database:

1- Connect Code

```
SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated Security=True;User Instance=True");
```

2- Open Connection

```
con.Open();
```

3- Insert new row for database

```
INSERT INTO tablename (column name1, column name2, column name3) values (value1, value2, value3);
```

Example:

```
INSERT INTO employee (Name,emp_id,designation)values('"+textbox1.text+"',"+textbox2.text+",'"+textbox3.text+"');
```

Note: If you use Html control we must use textbox1.value. If you use asp.net controls we have to use textbox1.text.

First Way:

```
SqlCommand cmd = con.CreateCommand();
cmd.CommandText = string .Format ("insert into Student(Id,FName,LName)values(@1,@2,@3)");
```

1- Clear Parameters and insert the values from textbox

```
cmd.Parameters.Clear();
cmd.Parameters.AddWithValue("@1", TextBox1.Text);
cmd.Parameters.AddWithValue("@2", TextBox2.Text);
cmd.Parameters.AddWithValue("@3", TextBox3.Text);
```

2- Execute the Query

```
cmd.ExecuteNonQuery();
```

3- Close the Connection

```
con.Close();
```

4- We can appear message box for user and empty the text boxes

```
int result=cmd.ExecuteNonQuery();

        if (result ==1 )
        {
MessageBox.Show("Your Data Insert Successfully", "Insert Data",
MessageBoxButtons.OK, MessageBoxIcon.Information,
MessageBoxDefaultButton.Button1);
      txtId.Text = "";
      txtFName.Text = "";
      txtLName.Text = "";
        }
```

Second Way:

```
cmd.CommandText = string.Format("insert into TestDB (Id,FName,LName)
values ('" + txtId.Text + "','" + txtFName.Text + "','" + txtLName.Text
+ "')");
```

5- Update Data:

The following code illustrates how to **update** data from specific table.

```
cmd.CommandText = string.Format("update Student set LName=@3 where
FName='" + TextBox2.Text + "' ");
     cmd.Parameters.Clear();
     cmd.Parameters.AddWithValue("@2", TextBox2.Text);
     cmd.Parameters.AddWithValue("@3", TextBox3.Text );
     con.Open();

protected void txtUpdate_Click(object sender, EventArgs e)
    {
        SqlConnection con = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\TestDB.mdf;Inte
grated Security=True;User Instance=True");
        SqlCommand cmd = con.CreateCommand();
        cmd.CommandText = string.Format("update Student set
LName=('"+txtLName .Text+ "'),FName=('"+txtFName .Text
+"'),Mark=('"+txtMark.Text +"') where FName=('" + txtEnterUpdate
.Text+ "') ");
        con.Open();
        cmd.ExecuteNonQuery();
        con.Close();

    }
```

6- Delete Data:

The following code illustrates how to **delete** data from specific table.

```
protected void btnDelete_Click(object sender, EventArgs e)
    {
```

```
        SqlConnection con = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\TestDB.mdf;Integra
ted Security=True;User Instance=True");
        con.Open();
        SqlCommand cmd = con.CreateCommand();
        cmd.CommandText = string.Format("delete from Student where
Mark= ('" + txtDeterDelete.Text + "')");
        DialogResult result1= MessageBox .Show("Are you Sure Delete
Data","Confirm Window",MessageBoxButtons .YesNo ,MessageBoxIcon
.Question ,MessageBoxDefaultButton .Button2 );
        if (result1 == DialogResult.Yes )
        {
            int result = cmd.ExecuteNonQuery();
            if (result == 1)
            {
                MessageBox.Show("Your Data Delete Successfuly", "Delete
Window", MessageBoxButtons.OK, MessageBoxIcon.Information,
MessageBoxDefaultButton.Button1);
                txtDeterDelete.Text = "";
            }
        }
        con.Close();
    }
```