Unit 4 – Master Pages and Theme, Caching, Application Pages and Data

Master pages

Master pages and Themes can be used when you want to create web pages that contain some common elements and you want to customize only a part of the web page.
ASP.NET master pages allow you to create a consistent layout for the pages in your application.
"**A master page is a page that defines the look and standard behavior that you want for all of the pages (or a group of pages) in your application."**
You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.
Master pages actually consist of two parts –

1) The master page itself
2) One or more content pages.

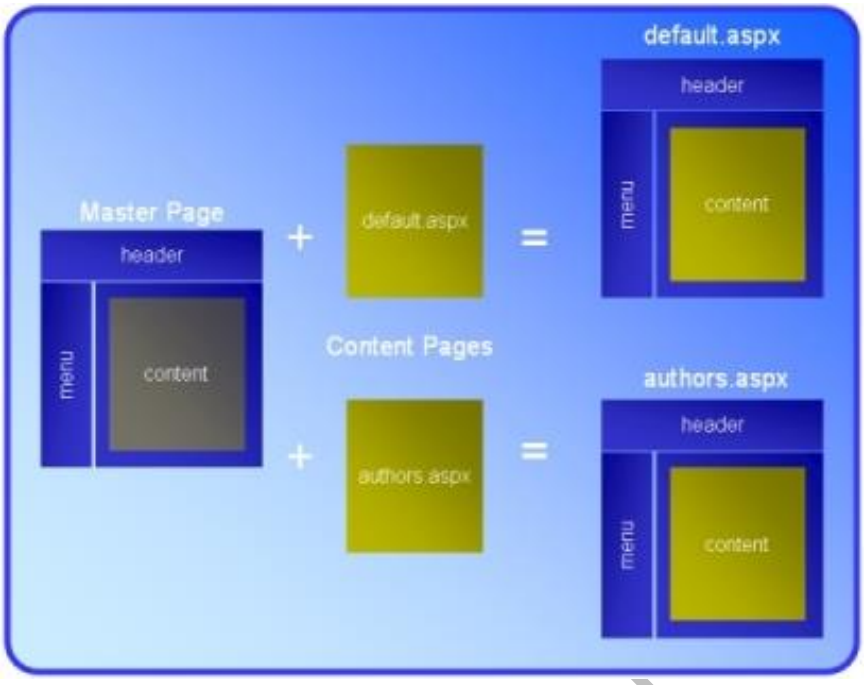
1) **Master Pages** : A master page is an ASP.NET file with the extension .master with a predefined layout that can include static text, HTML elements, and server controls. The master page is identified by a special @ Master directive that replaces the @ Page directive that is used for ordinary .aspx pages.

2) **Content Pages** : You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page. The binding is done in the content page's @ Page directive by including a MasterPageFile attribute that points to the master page to be used.

Advantages of Master pages

Master pages provide functionality that developers have traditionally created by copying existing code, text, and control elements repeatedly; using framesets; using include files for common elements; using ASP.NET user controls; and so on. Advantages of master pages are as follows:

1. They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
2. They make it easy to create one set of controls and code and apply the results to a set of pages. For example, you can use controls on the master page to create a menu that applies to all pages.
3. They provide an object model that allows you to customize the master page from individual content pages.



Themes

A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server. Themes define the style properties of a web page. There are 3 elements of themes such as skins, CSS and images.

A skin is a file with .skin extension that contains property settings for controls such as button, textbox, label etc. Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme. For example, the following is a control skin for a Button control:

<asp:button runat="server" BackColor="red" ForeColor="black" />

You create .skin files in the Theme folder. A .skin file can contain one or more control skins for one or more control types. You can define skins in a separate file for each control or define all the skins for a theme in a single file.

Types of themes

You can define themes for a single Web application, or as global themes that can be used by all applications on a Web server. Themes are of 2 types –

1) Page Themes and 2) Global Themes

1) **Page Themes** : A page theme contains the control skins, style sheets, graphics files and other resources stored inside a subfolder of "**App_Themes**" folder in your Web site. A page theme is applied to a single page of the web site. For different themes, separate subfolders are created in the App_Themes folder.
2) **Global Themes** : A global theme is a theme that you can apply to all the Web sites on a server. Global themes allow you to define an overall look for your domain when you

maintain multiple Web sites on the same server. Global themes are placed in the "Themes" folder that is global to the web server and can be accessed by any web site.

Overview of Caching

Caching is the mechanism used to improve the performance of a web application.
The slowest operation that you can perform in an ASP.NET page is database access. Opening a database connection and retrieving data is a slow operation.
By taking advantage of caching, you can cache your database records in memory. Retrieving data from a database is very slow. Retrieving data from the cache, on the other hand, is very fast.
The ASP.NET 3.5 Framework supports the following types of caching:

1. Page Output Caching
2. Partial Page Caching
3. DataSource Caching
4. Data Caching

Page Output Caching

Page output caching can be used to dramatically improve the performance of the Web site. Instead of executing a page each time the page is requested, you can cache the output of the page and send the cached copy to satisfy browser requests.

Consider, for example, that your site includes a page that displays product information retrieved from a database table. By default, every time someone requests the product page, the page must be executed and the information must be retrieved from the database. If you enable page output caching, however, the page is executed only once, and the information is retrieved from the database only once. This means less work both for your Web application and for your database server.

To enable output caching for a page, include the following page directive in the page:

<%@ OutputCache Duration="15" VaryByParam="none" %>

For example, the following page caches its contents for 15 seconds.

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<%@ OutputCache Duration=”15” VaryByParam=”none” %>
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>

<html xmlns=”http://www.w3.org/1999/xhtml” >
```

```
<head id="Head1" runat="server">
        <title>Cache Page Output</title>
</head>
<body>
        <form id="form1" runat="server">
        <div>
                <asp:Label id="lblTime" Runat="server" />
        </div>
        </form>
</body>

</html>
```

C# code

```
using System;
public partial class _Default : System.Web.UI.Page
{
        protected void Page_Load(object sender, EventArgs e)
        {
                lblTime.Text = DateTime.Now.ToString("T");
        }
}
```

The above page displays the current server time in a Label control. If you refresh the page multiple times, you will notice that the time is not updated until at least 15 seconds have passed. When you cache a page, the contents of the page are not regenerated each time you request the page.

Partial Page Caching

Partial Page Caching is used to cache particular regions of a page.
Partial Page Caching makes sense when a page contains both dynamic and static content. For example, you might want to cache a set of database records displayed in a page, but not cache a random list of news items displayed in the same page.

There are two methods for enabling Partial Page Caching like –

1) Post-cache substitution
2) Using User controls

1) **Post-cache substitution** : In some cases, you might want to cache an entire page except for one small area. You can use post-cache substitution to cache an entire page except for a particular region. For example, you might want to display the current username dynamically at the top of a page but cache the remainder of a page. In these cases, post-cache substitution is used.

2) **Using User control** : Using post-cache substitution is appropriate only when working with a string of text or HTML. If you need to perform more complex partial page caching, then you should take advantage of User Controls. You can use User Controls to cache particular regions in a page, but not the entire page.

DataSource Caching

Instead of caching at the page or User Control level, you can cache at the level of a DataSource control. Three of the four standard ASP.NET DataSource controls—the SqlDataSource, ObjectDataSource, and XmlDataSource controls—include properties that enable you to cache the data that the DataSource control represents The LinqDataSource control does not support caching.

One advantage of using the DataSource controls when caching is that the DataSource controls can reload data automatically when the data is updated. The DataSource controls are also smart enough to share the same data across multiple pages.

Absolute Cache Expiration

When you use an absolute cache expiration policy, the data that a DataSource represents is cached in memory for a particular duration of time. Using an absolute cache expiration policy is useful when you know that your data does not change that often.

For example, if you know that the records contained in a database table are modified only once a day, then there is no reason to keep taking the same records every time someone requests a web page.

Example : The following page displays a list of movies that are cached in memory. The page uses a SqlDataSource control to cache the data.

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.1//EN”
“http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml” >
<head id=”Head1” runat=”server”>
        <title>DataSource Absolute Cache</title>
</head>
<body>
        <form id=”form1” runat=”server”>
        <div>
        <asp:GridView id=”grdMovies” DataSourceID=”srcMovies” Runat=”server” />

        <asp:SqlDataSource id=”srcMovies” EnableCaching=”True” CacheDuration=”3600”
        SelectCommand=”SELECT * FROM Movies”
        ConnectionString=”<%$ ConnectionStrings:Movies %>” Runat=”server” />
        </div>
```

```
    </form>
</body>
</html>
```

In above example, two properties of the SqlDataSource control related to caching are set. First, the "**EnableCaching**" property is set to the value True. Next, the "**CacheDuration"** property is set to the value 3,600 seconds (1 hour). The Movies are cached in memory for a maximum of 1 hour. If you don't supply a value for the CacheDuration property, the default value is Infinite.
You can test whether the above page is working by opening the page and temporarily turning off your database server. If you refresh the page, the data is displayed even though the database server is unavailable.

Note: It is important to understand that there is no guarantee that the SqlDataSource control will cache data for the amount of time specified by its CacheDuration property. Behind the scenes, DataSource controls use the Cache object for caching. This object supports searching. When memory resources become low, the Cache object automatically removes items from the cache.

Sliding Cache Expiration

If you need to cache a lot of data, then a sliding expiration policy is more useful than an absolute expiration policy. When you use a sliding expiration policy, data remains in the cache as long as the data continues to be requested within a certain interval.
For example, suppose you want to rewrite the Amazon website with ASP.NET. The Amazon website displays information on billions of books. You couldn't cache all this book information in memory. However, if you use a sliding expiration policy, then you can cache the most frequently requested books automatically.
Example : The following page shows how you can enable a sliding cache expiration policy. The cache duration is set to 15 seconds. As long as no more than 15 seconds pass before you request the page, the movies are kept cached in memory.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
<title>DataSource Sliding Cache</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <p>
    <asp:Label id="lblMessage" EnableViewState="false" Runat="server" />
```

```
        </p>
        <asp:GridView id=”grdMovies” DataSourceID=”srcMovies” Runat=”server” />

        <asp:SqlDataSource
        id=”srcMovies”
        EnableCaching=”True”
        CacheExpirationPolicy=”Sliding”
        CacheDuration=”15”
        SelectCommand=”SELECT * FROM Movies”
        ConnectionString=”<%$ ConnectionStrings:Movies %>”
        OnSelecting=”srcMovies_Selecting”
        Runat=”server” />

        </div>
        </form>
</body>
</html>
```

C# code

```
using System;
public partial class _Default : System.Web.UI.Page
{
        protected void srcMovies_Selecting(object sender,
        SqlDataSourceSelectingEventArgs e)
        {
                lblMessage.Text = “Selecting data from database”;
        }
}
```

The above page includes a srcMovies_Selecting() event handler. This handler is called only when the movies are retrieved from the database rather than from memory. In other words, you can use this event handler to detect when the movies are dropped from the cache.

Data Caching

Data Caching is the fundamental caching mechanism. Behind the scenes, all the other types of caching use Data Caching.
You can use Data Caching to cache random objects in memory. For example, you can use Data Caching to cache a DataSet across multiple pages in a web application.

All the various caching mechanisms included in the ASP.NET Framework use the “**Cache**” object. You can cache items in memory by taking advantage of the Cache object, which is an instance of the Cache class. Each ASP.NET application has a single Cache object that remains valid until the application is restarted.

Any items you add to the cache can be accessed by any other page, control, or component contained in the same application.

The Cache object has the following properties:
1) **Count**—Represents the number of items in the cache.
2) **EffectivePrivateBytesLimit**—Represents the size of the cache in kilobytes.

The Cache object has the following methods:
1) **Add**—Enables you to add a new item to the cache. If the item already exists, this method fails.
2) **Get**—Enables you to return a particular item from the cache.
3) **GetEnumerator**—Enables you to iterate through all the items in the cache.
4) **Insert**—Enables you to insert a new item into the cache. If the item already exists, this method replaces it.
5) **Remove**—Enables you to remove an item from the cache.