Chapter 1
Web Development and ASP.NET
Introduction

Content

1. Goals to be Achieved 3
2. Building Your First Official ASP.NET Application 3
2.1. The ASP.NET Namespaces 3
2.2. Creating a Simple C# Web Application 3
3. The architecture of an ASP.NET Web Application 4
3.1. The System.Web.UI.Page Type 4
3.2. Caching of data in a Round Trip? 6
3.3. The *.aspx/CodeFile Connection 7
3.4. An Example 7
4. Examples 10
4.1. Exa141_A: „Simple Pocket Calculator“ 10

Web Development and ASP.NET

1. Goals to be Achieved

- In this chapter we begin to explore how the .NET platform facilitates the construction of browser-based presentation layers.
- We will partition the HTML presentation logic and business logic into discrete locations using a technique called *CodeFile.*
- Essential features are the *Page type* the classic ASP-like *Request, Response, Session* and *Application* properties.

2. Building Your First Official ASP.NET Application

2.1. The ASP.NET Namespaces

The .NET class libraries contain numerous *namespaces* that represent Web-based technologies. These *namespaces* can be grouped into three major categories: *Core Web Atoms* (e.g. HTTP types, configuration types, security types), *UI* (*WebForm* controls) and *Web services*. The following table shows the essentials of the *Core Types* of the `System.Web` namespace:

System.WEB Type	Meaning
`HttpApplication`	Defines the members common to all ASP.NET applications.
`HttpApplicationState`	Enables developers to share global information across multiple requests, sessions, and pipelines in an ASP.NET application.
`HttpRequest`	Provides an object-oriented way to enable browser-to-server communication (e.g. used to gain access to the HTTP request data supplied by a client).
`HttpResponse`	Provides an object-oriented way to enable server-to-browser communication (e.g. used to send output to a client)

2.2. Creating a Simple C# Web Application

First, create a new C# Web Application project workspace named `FirstWebApplication`: Project Types: *Visual C# Projects*, Templates: *ASP.NET Web Site*.

In this case the IIS must be installed locally on your computer!

Open Visual Web Developer. On the *File* menu, click *New Web Site*. The New Web Site dialog box appears:

1. Under Visual Studio installed templates, click **ASP.NET Webapplication (.NET Framework)**.
2. Click **Forward**.
3. Click **Open** and type the name **`FirstWebApplication`**.
4. Right Click on Project and add a new Item **WebForm** and name it **Default.aspx**
5. Visual Web Developer creates the new Web Site and opens a new class named **Default.aspx**, which is the default Web Site.
6. Now start implementing your Web Application.

Notice that the location text box maps to a specific folder on your hard drive in case of **File System.** Whereas to the URL of the machine hosting this Web application in case of **Local IIS**. When the project has been created, you will notice that a design time template has been opened automatically (Fig. 1) which can be viewed in *Design*-, *Split*-, or *Source*-mode. **The Source mode is an XML-like document in which you can manipulate your controls manually!**

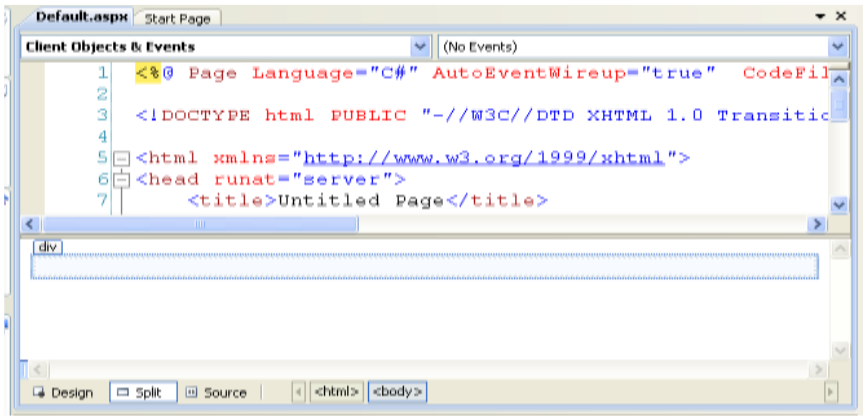

Fig: 1. The Source File / Design Mode in VS.Net

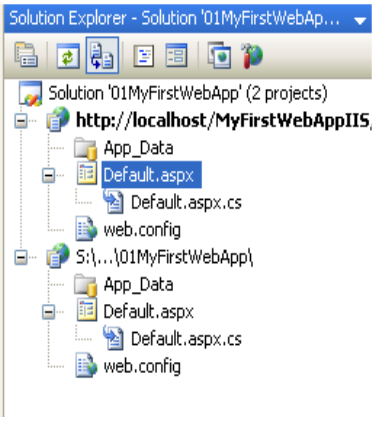

Fig. 2: The Solution Explorer

This template represents the visual appearance of the *.aspx file you are constructing. Given that this will be the page requested by the outside world named `Default.aspx`.

Next, look at your *Solution Explorer* window (Figure 2). You have been given a number of new files and external assembly references. If you open IIS, you will see that a new virtual directory (First-WebApplication) has been automatically created on your behalf.

If you examine the HTML behind your *.aspx file, you will see that you have been given the minimal set of tags that establish a basic HTML form. The first point of interest is the **`runat`** attribute appearing in the `<form>`-tag. It is used to mark an item as a candidate for processing by the ASP.NET runtime to generate HTML to return to the browser, as shown here:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!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 runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

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

The *web.config* file (last file in the *Solution Explorer*) contains XML data used to control various aspects of your Web application's configuration.

3. The architecture of an ASP.NET Web Application

3.1. The System.Web.UI.Page Type

The Page class defines the properties, methods, and events common to all pages processed on the server by the ASP.NET runtime. Some of these are:

SYSTEM.WEB.UI.Page properties	Meaning

`Application`	Gets the HttpApplicationState object provided by the runtime. Works like a Collection!
	`Application["Test"] = 12.34f;`
	`Application.Add("Test", 13.45f);`
	Enables sharing of global information across multiple sessions and requests within an ASP.NET application.

`Cache`	Indicates the Cache object in which to store data for the pages's application.
`IsPostBack`	Gets a value indicating whether the page is being loaded in response to a client postback, or if it is being loaded and accessed for the first time.
`Request`	Gets the HttpRequest object that provides access data from incoming HTTP requests. I.e. to **read cookies** `String name = Request.Cookies.Get(i).Name;`

`Response`	Gets the HttpResponse object that allows you to send HTTP response data back to a client browser. I.e. to set cookies
	`HttpCookie cook01 = new HttpCookie("Cook01");`
	`cook01.Value = "DrChess";`
	`cook01.Expires = DateTime.Now.AddSeconds(10);`
	`Response.Cookies.Add(cook01);`

`Server`	Gets the HttpServerUtility object supplied by the HTTP runtime. `Server.HtmlEncode(cook01.Name);`

`Session`	Gets the System.Web.SessionState.HttpSessionState object, which provides information about the current request's session.
	`Session["Calls"] = numOfCallsWithSession;`
	`Session.Add("Calls", numOfCallsWithSession);`
	Session object are not discarded when the user moves from page to page in the application; instead, these variables persist as long as the user is accessing pages in your application.

SYSTEM.WEB.UI.Page Events	**Meaning**
`OnInit`	Is fired when the page is initialised and is the first step in the page's life cycle.
`OnLoad`	Is fired when initialised. Here you can configure any WebForm controls with an initial look and feel.
`OnUnload`	Occurs when the control is unloaded from memory. Controls should perform any final cleanup before termination. `sqlCon.Close();`

Tab 1: Page members

The *event* handler for the *Load* event is a perfect place to **connect to a data source** (to populate a given WebForm DataGrid) and perform any necessary pre-work. The *Unload* handler is a perfect place to **clean up any allocated resources**.

3.1.1. Comments on Cookies

Creating/Writing Cookies: Cookies will be created with `Response.` There are 2 ways to create a cookie. In this example we create a cookie *named UN* with *value UserName* and *expire time 10* sec.:

```
HttpCookie unCookie = new HttpCookie("UN");
unCookie.Value = "UserName";
unCookie.Expires = DateTime.Now.AddSeconds(10);
Response.Cookies.Add(unCookie);
```

In this example we create a cookie *named PW* with *value Password* and *expire time 15* sec.:

```
Response.Cookies["PW"].Value = "Password";
Response.Cookies["PW"].Expires = DateTime.Now.AddSeconds(15);
```

Reading/Getting Cookies: Cookies will be read by `Request.` In this example we read the name of a cookie in the Cookies Request-list:

```
HttpCookie cookie = Request.Cookies[i];
Response.Write("<br />Cookie name___: " + cookie.Name);
```

Remove/ChangeValie from Cookies: You also can remove Cookies from the Cookie-list or change the value :

```
Request.Cookies.Remove("UN");
Request.Cookies["UN"].Value = "Dr.Chess";
```

Use of Cookies:

Cookies may be used for *authentication*, *identification* of a *user session*, *user's preferences*, *shopping cart contents*, or anything else that can be accomplished through storing text data. Cookies can also be used for *travelling of data from one page to another*.

3.2. Caching of data in a Round Trip?

A ***round trip*** is a process that initiates an action (i.e. button click) on the client side. This action calls the event in the .aspx-side that is located on the server side. The event processes the request and returns the result as an html-page. Caching data in a *round trip* we can decide between caching on the server side or on the client side.

- **Caching data on the server side:** Here we can use both the *Session* state and *Application* state.
- **Caching data on the client side:** Here we can use both the *View stat* as well as the *hidden field*. This data is stored only at the same page. When we redirect to a different page and redirect back the hidden value will be lost! It is recommended to apply a

Caching data on the server side may use additional resources on the server. This may lead to a decrease in the performance of the server and may influence the scalability of the system.

Example:

In this example we want to read the value 4711 from a text box in the *Default.aspx* page. This value will be cached in a Session. In the 2nd aspx-file (i.e. Test.aspx) we will read this data from the Session and output in a text box in the 2nd aspx-file.

Implementation of the „Caching"- Default.aspx:

```
Session["Wert"] = Convert.ToInt32(tbxSession.Text);
```

Implementation of the Caching – Test.aspx::

```
int wert = Convert.ToInt32(Session["Wert"]);
```

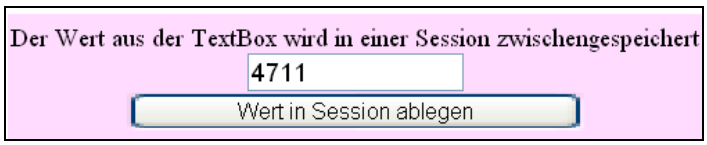

Abb. 3: Default.aspx: 4711 wird in Session gecacht

Abb. 4: Test.aspx: Ausgabe der gecachten Information

3.3. The *.aspx/CodeFile Connection

The first major point of interest is the mysterious `CodeFile` (former `Codebehind`) attribute in the initial script block. The *.aspx page, which is requested by an external client, is represented by a unique C# class, identified by the `CodeFile` attribute. To access the Codebehind-file, simply right-click an open *.aspx file and select "View Code".

```
using System;
using System.Configuration;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {    }
}
```

The default skeleton code is not too complicated. The constructor `Page_Load` of the Page-derived class will be executed whenever a post back from the server occurs.

ASP.NET provides a way for separating the logic that dynamically generates the returned HTML (the *.aspx) file from the implantation of your page's logic (e.g., the *.aspx.cs file). Now when you are writing code in the *.aspx file, you can reference the custom methods and properties defined in the *.aspx.cs file.

3.4. An Example

This example is to demonstrate both the binding of the *CodeFile* as well as the role of the *Page_Load* method in the *CodeFile*. The application consists of 2 pages: `Default` and `Pag02`. Both pages consist of 2 Buttons each. The page Default implements 3 Integer-Variables that have to be incremented by one on each post back and will be initialised with the value = 1 when starting the application in the page `Default`:

- ✓ `staticNumOfCalls:` A static int-variable.
- ✓ `dynamicNumOfCalls:` An int-variable..
- ✓ `numOfSessionCalls:` An int-variable that will be stored in a Session-object.

The following operations should give the following results:

- ✓ Give the actual time (Response-Object) in the page `Default` (Fig. 1).
- ✓ Click 4 times the button *PostBack*. The result in page `Default` is given in Fig. 2.
- ✓ Click the button *ToPage02* in page `Default`. Navigate to page `Page02`. View result in Fig. 3.
- ✓ Click the button *PostBack* in page `Page02`. The result is given in Fig. 4.
- ✓ Click the button *ToDefault02* in page `Page02`. This redirects to page `Default`. The result is given in Fig. 5.
- ✓ Click button *PostBack* twice. The result is given in Fig. 6.

When starting the web application, the following *HTML* file will be displayed (Fig. 5). This is the page, when no "postback" occurred rather when you open the start page.

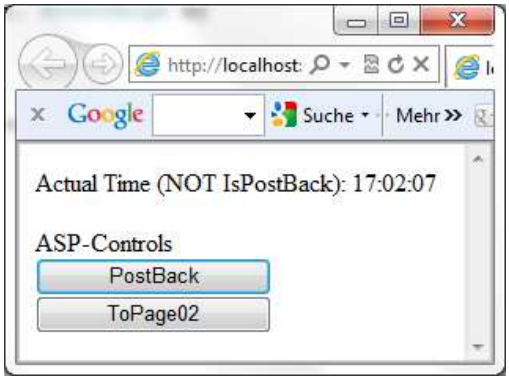

Fig. 1: The page "Default" at start time

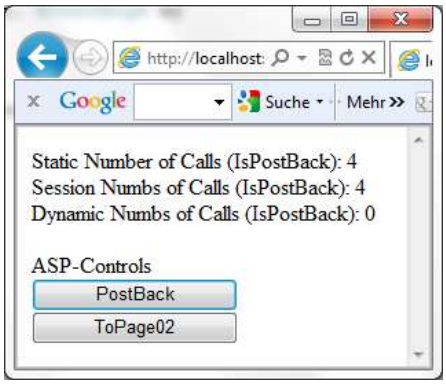

Fig. 2: The page "Default" after 4 post backs

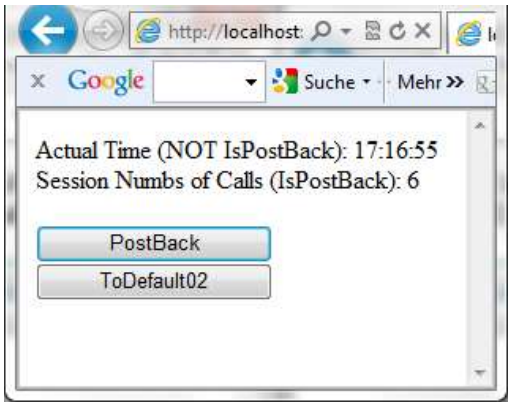

Fig. 3: Redirect to Page02

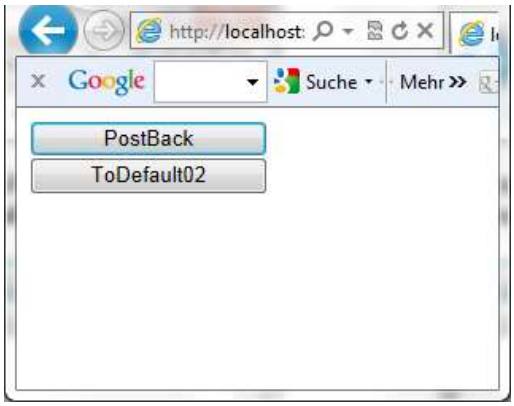

Fig. 4 PostBacks in page Page02

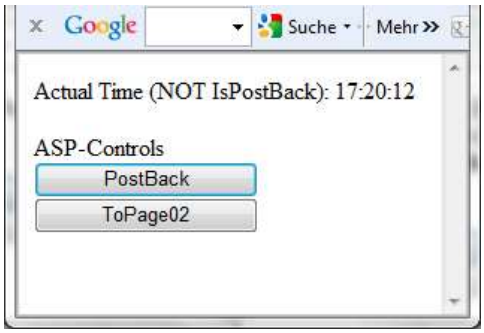

Fig. 5: Redirect to page "Default"

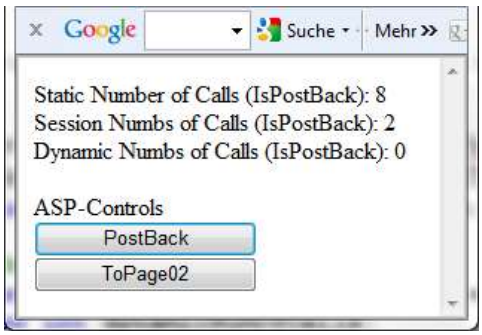

Fig. 6: Another 2 post backs in page "Default"

This is almost the complete code for the page *Default.aspx*:

Default.aspx.cs: Execution of the code lines of `!IsPostBack` in the *Page_Load* function of the *Default.aspx.cs* (code behind file). This gives the Actual Time in the HTML-File.

```
public partial class _Default : System.Web.UI.Page
{
    // Fields to show stateless forms
    protected static int staticNumOfCalls = 0;
    protected int dynamicNumOfCalls;
    protected int numOfSessionCalls;
    protected void Page_Load(object sender, EventArgs e)
    {
        // The "TRUE" part will be called in case of istantiation =
        // surfing this page the first time!!!
        if (!IsPostBack)
        {
            Response.Write("Actual Time (!IsPostBack): " +
                                    DateTime.Now.ToLongTimeString());
            staticNumOfCalls++;
            dynamicNumOfCalls++;
```

```
            numOfSessionCalls++;
            Session["Calls"] = numOfSessionCalls;
        }
        // The "FALSE" part will be called in case of any triggering by
        // asp-controls!!!
        else
        {
            // Print the "Static Number Of Calls"
            Response.Write("Static Number Of Calls (IsPostBack): " +
                                              staticNumOfCalls + "<br />");
            staticNumOfCalls++;
            // Print the "Number Of Calls With Session"
            …
            // Print the "Dynamic Number Of Calls".
            Response.Write("Dynamic Nums Of Calls (IsPostBack): " +
                                                    dynamicNumOfCalls);
            dynamicNumOfCalls++;
        }
    }
    /// <summary>
    /// This function is called by the button to navigate to next page.
    ///
    protected void onToPage02Click(object sender, EventArgs e)
    {
        Response.Redirect("SecondPage.aspx");
    }
    /// <summary>
    /// This function is called by the aspx-file via Response.Write
    /// </summary>
    protected String getDateTime()
    {
        return "Call the function getDateTime() from CodeFile: " +
                                            DateTime.Now.ToUniversalTime();
    }
}
```

Default.aspx: Initiation of the HTML-file as given in the *Default.aspx* as given below. This initiates the `Response.Writes` as well as the *web-controls*.

```
<body>
  <form id="form1" runat="server">
    <div>
      <%Response.Write("ASP-Controls"); %><br />
      <asp:Button ID="btnPostBack" runat="server" Text="PostBack"
                  OnClick="onPostBackClick" Width="150px" /><br />
      <asp:Button ID="btnToPage02" runat="server" Text="ToPage02"
                  OnClick="onToPage02Click" Width="150px" />
    </div>
  </form>
</body>
```

ASP-Controls

When ever you trigger a **post back** the method *Page_Load* is the first method that will be executed followed by the ASP-control event, in this case the `onPostBackClick(…)`. Most of all ASP controls are *post back able*! ASP-Controls do access functions in a code-behind file, an *aspx.cs*! As in this case the following html-file will be produced. This is done by execution of the code lines following the `IsPostBack` in the *Page_Load* function of the *Default.aspx.cs* and execution of the event `onPostBackClick(…)`. Confer figures above (Fig. 1 – Fig. 6)

4. Examples

4.1. Exa141_A: „Simple Pocket Calculator“

Create an ASP.Net application that simulates a primitive pocket calculator. It consists of only the following web controls: 4 buttens (*Operations +, -, *, /)*, 3 text fields *(operand 1, operand 2, result*), 1 label for the operator sign. The headline is a simple Write statement. You have to implement 4 functions in Java Script that performs the required operations.

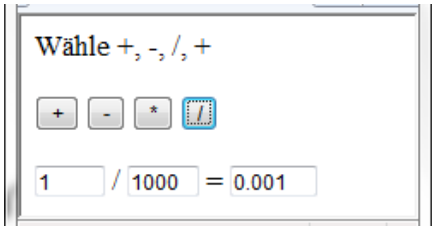

Fig. 7: The pocket calculator

Erstelle ein Loginfenster, Gib in einem HiddenFields bzw. Session 5 Passwörter vor, die beim Login

- ✓ übertrage die Login-Werte vom Hidden-Field auf die Admin-Seite und zeige diese an. Ändere diese Werte und navigiere zur Login-Seite