4 »Hallo, Welt!« – Ein ASP.NET-Crashkurs

»Hallo, Welt!« ist das Thema dieses Kapitels. Zahlreiche kleine ASP.NET-Skripte vermitteln Ihnen ein erstes Feeling davon, wie ASP.NET funktioniert.

In diesem Kapitel schreiben Sie Ihre ersten ASP.NET-Seiten. Wenn Sie die Beispiele selbst nachvollziehen möchten, müssen auf Ihrem Rechner die Internet Information Services und das .NET Framework SDK installiert sein. Falls diese Voraussetzungen noch nicht gegeben sein sollten, können Sie sich in Kapitel 2, Installation, über die notwendigen Installationsschritte informieren.

Vielleicht verstehen Sie einige der folgenden Beispiele noch nicht vollständig. Das macht nichts. Mit Hilfe der kleinen Skripte sollen Sie zunächst ein Gefühl dafür bekommen, wie ASP.NET funktioniert. Wenn Sie einfach mitmachen, verfügen Sie am Ende des Kapitels bereits über ein Grundgerüst für Webformulare, das Sie für Ihre eigenen Programmierexperimente verwenden können.

4.1 Der Aufbau von ASP.NET-Seiten

Bei der herkömmlichen ASP-Programmierung können Abschnitte aus HTML-Code ziemlich beliebig mit JavaScript-Routinen und ASP-Schnipseln vermischt werden. Wer sich bei der Entwicklung nicht selbst zu Disziplin zwingt, kann dadurch ziemlich undurchsichtige Seiten mit einem nur schwer wartbaren Codegemisch erzeugen. Mit ASP.NET sind diese Spaghetti-Code-Zeiten vorbei, denn ASP.NET-Seiten sind nach einem festen Schema aufgebaut.

ASP.NET-Seiten, die im Browser direkt aufgerufen werden, haben die Dateinamensergänzung `.aspx`. Diese aspx-Seiten bestehen aus drei Teilen:

1. Am Anfang steht die `@Page`-Direktive: `<%@ Page ... %>`
2. Es folgt ein Block mit serverseitigem Skriptcode:
   `<script runat="server"> ... </script>`
3. Am Schluss steht der HTML-Code: `<html>...</html>`

Das Grundgerüst für eine aspx-Seite hat damit diese Form:

```
<%@ Page Language="VB" Debug="True" Strict="True" %>
<script runat="server">
   ' Hier steht der VB-Code
</script>
```

```
<html>
   <!-- Und hier steht der HTML-Code -->
</html>
```

4.1.1 Die @Page-Direktive

```
<%@ Page Language="VB" Debug="True" Strict="True" %>
```

Die `@Page`-Direktive enthält Anweisungen für den Compiler. Mit `Language="VB"` wird die Programmiersprache festgelegt, die auf dieser Seite verwendet wird. Hier können alle Sprachen angegeben werden, die .NET unterstützt. Microsoft unterstützt von Haus aus `VB`, `C#` und `JScript .NET`. Die Beispiele dieses Buches sind alle in VB.NET erstellt. Also haben alle `@Page`-Direktiven in diesem Buch den Eintrag `Language="VB"`.

Das Attribut `Debug="True"` veranlasst den Compiler, spezielle Debug-Symbole in den kompilierten Code einzutragen, die das Debuggen erleichtern. Auch dieses Attribut wird auf allen Beispielseiten des Buches verwendet.

Achtung Wenn Sie Seiten jedoch produktiv einsetzen, sollten Sie `Debug="false"` einstellen, weil sonst ein unnötiger Overhead erzeugt wird.

Das Attribut `Strict="True"` ist nur wirksam, wenn die Sprache VB.NET verwendet wird. Es bewirkt, dass der VB.NET-Code mit `Option Strict` kompiliert wird. Das bedeutet zweierlei:

- Variablen müssen stets deklariert und auch typisiert werden.
- Implizite Typkonvertierungen sind nur von einem engeren zu einem weiteren Typ möglich, z. B. `LongTypVar = IntegerTypVar`. Typeinengende Konvertierungen, z. B. `IntegerTypVar = LongTypVar`, müssen dagegen stets explizit erfolgen.

Um einen sauberen Programmierstil zu fördern, wird in allen Beispielen im Buch diese strikte Vorgabe befolgt. Dabei stehen einem kleinen Nachteil mehrere große Vorteile gegenüber, die sich langfristig auszahlen. Der Nachteil: Der Entwickler muss lernen, wann er überhaupt konvertieren muss, und sich dafür eine Handvoll Konvertierungsfunktionen merken. Die Vorteile:

- Der Entwickler ist gezwungen, sich genau klarzumachen, was eigentlich passiert. Die Strenge bei der Codierung fördert die Klarheit beim Denken.
- Der Code enthält weniger Fehler.
- Das Erlernen einer Sprache wie C# fällt leichter, weil auch hier explizit konvertiert werden muss.

4.1.2 Der serverseitige Skript-Block

```
<script runat="server"> ... </script>
```

Der Skript-Block enthält die Definition globaler Variablen und von Prozeduren und Funktionen. Dieser Block darf keine frei stehenden Anweisungen enthalten. Jeder Code mit Ausnahme von Variablendeklarationen muss im Rahmen einer `Sub` oder `Function` stehen.

Oft finden Sie hier Ereignisprozeduren wie etwa `Page_Load`. Sie können aber auch beliebige eigene Prozeduren erstellen.

Achtung Vergessen Sie beim Anlegen des Skript-Blocks nicht das Attribut `runat="server"`. Wenn Sie das vergessen sollten, wird der Code nicht auf dem Server ausgeführt, sondern zum Browser geschickt. Der aber kann mit dem hier definierten Code nichts anfangen.

4.1.3 Der Block mit HTML-Code

Der HTML-Code-Block gehorcht zunächst einmal den Regeln der traditionellen HTML-Programmierung. In diesen herkömmlichen HTML-Code werden außerdem die speziellen ASP.NET-Steuerelemente eingefügt, die Sie im Laufe dieses Buches kennen lernen werden. Diese haben die Form `<asp:Name Attribut="Wert" ... > ... </asp:Name>` und gehorchen der XML-Syntax. Die Tags müssen also auch korrekt geschlossen werden. Hier ein Beispiel:

```
<asp:Label id="ausgabe" runat="server" />
```

Bei einigen serverseitigen Steuerelementen erlaubt ASP.NET Ausnahmen von der Regel, dass ein Tag auch geschlossen werden muss. Wenn man aber immer auch ein schließendes Tag setzt beziehungsweise die gezeigte abkürzende Schreibweise verwendet, dann macht man garantiert nichts falsch.

Bei der Auswertung einer aspx-Seite wertet ASP.NET diese serverseitigen Steuerelemente aus und generiert daraus HTML-Code. Dort, wo Sie bei der herkömmlichen ASP-Programmierung mit Code-Schnipseln in der Form `<% ... %>` arbeiten mussten, können Sie bei ASP.NET stattdessen sehr oft diese neu eingeführten serverseitigen Steuerelemente verwenden.

Zusätzlich ist es aber immer noch möglich, im HTML-Block Code-Schnipsel einzufügen. Den Inhalt einer String-Variablen können Sie beispielsweise nach wie vor in dieser Form ausgeben:

```
<% = myVar %>
```

4.2 Das erste ASP.NET-Skript: »Hallo, Welt!«

4.2.1 »Hallo Welt!« – fast wie ASP

Die erste Programmieraufgabe ist immer die gleiche. Ein Entwickler beginnt mit dem Erlernen einer neuen Sprache, indem er der Welt seinen Gruß entrichtet. `hallo01.aspx` zeigt, wie das mit ASP.NET funktioniert:

```
<!-- hallo01.aspx -->
<%@ Page Language="VB" Debug="True" Strict="True"  %>
<script runat="server">
Dim tmp As String ="Hallo, Welt!"
</script>
<html><head><title ><% = tmp %></title></head><body>
<h1><% = tmp %></h1>
</body></html>
```

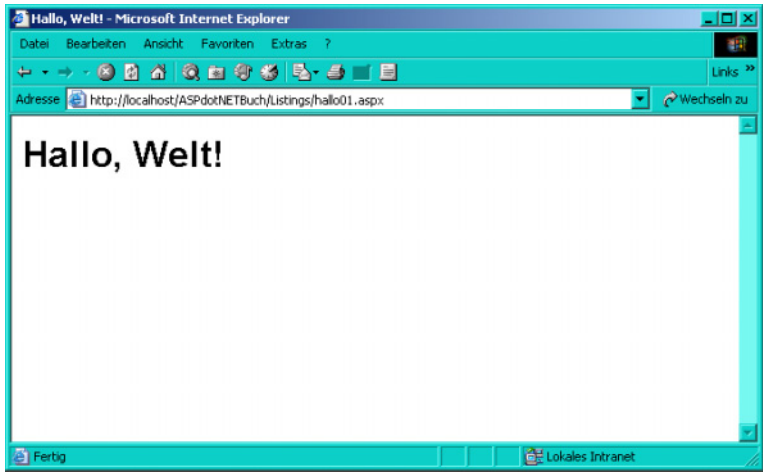

Abbildung 4.1 Geschafft!

Abbildung 4.1 zeigt das Ergebnis. Dieses erste Beispiel ist wenig spektakulär. Im Prinzip arbeitet es noch so ähnlich wie das klassische ASP. Im Skriptblock wird die Variable `tmp` definiert, die im HTML-Block zweimal mit `<% = tmp %>` ausgegeben wird.

Gerade am Anfang vergisst man leicht, an den nötigen Stellen das Attribut `runat="server"` anzugeben. Falls Sie beispielsweise statt `<script runat="server">` nur `<script>` schreiben, erhalten Sie im Browser eine Fehlermeldung wie in Abbildung 4.2. Hier hat ASP.NET beim Kompilieren bemerkt, dass es den Inhalt der Variablen `tmp` ausgeben soll, dass es diese aber gar nicht kennt.

Auch das ist ein bedeutender Unterschied zum klassischen ASP. Während man dort bei Fehlern oft auf Vermutungen angewiesen war, unterstützt ASP.NET den Entwicklungsprozess durch aussagekräftige Fehlermeldungen.

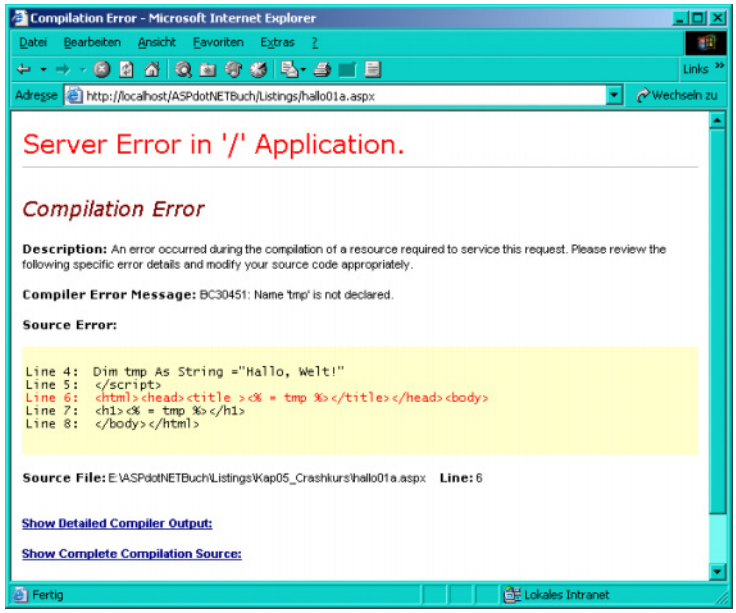

Abbildung 4.2 Abbildung 4.2:ASP.NET erzeugt aussagekräftige Fehlermeldungen.

4.2.2 »Hallo, Welt!« im ASP.NET-Stil

Das gleiche Ergebnis kann auf verschiedene Arten erzeugt werden. Es entspricht etwas mehr dem ASP.NET-Stil, wenn Sie den Variableninhalt nicht einfach als Schnipsel in den HTML-Code fallen lassen, sondern die beiden HTML-Elemente, die den variablen Inhalt erhalten sollen, aus dem Code-Block der aspx-Seite heraus steuern. Eine Möglichkeit wäre etwa folgende:

```
<!-- hallo02.aspx -->
<%@ Page Language="VB" Debug="True" Strict="True"  %>
<script runat="server">
Sub Page_Load (ByVal Sender As Object, _
               ByVal E As EventArgs)
   Dim tmp As String ="Hallo, Welt!"
   myTitle.InnerText = tmp
   myH1.InnerText = tmp
End Sub
</script>
<html><head>
```

```
<title id="myTitle" runat="server" ></title></head>
<body><h1 id="myH1" runat="server"></h1>
</body></html>
```

Der HTML-Block wird hier nicht mehr mit `<% ... %>`-Einsprengseln vermischt. Die beiden Elemente, die den variablen Text zugewiesen bekommen sollen, sind auf eine charakteristische Art und Weise ergänzt worden. Beiden haben ein `id`-Attribut und das Attribut `runat="server"` erhalten. Dadurch wird es möglich, den Inhalt dieser Elemente über den Skript-Code zu beeinflussen.

Auch im Skript-Block gibt es eine signifikante Neuerung. Hier wird die Ereignisprozedur `Page_Load` definiert. Eine Prozedur mit diesem Namen wird beim Laden der Seite stets automatisch ausgeführt. `Page_Load` verfügt immer über die folgende Signatur, die auch für die meisten anderen Ereignisprozeduren charakteristisch ist.

```
Sub Page_Load (ByVal Sender As Object, _
               ByVal E As EventArgs)
```

Mit dem Wort Ereignisprozedur wird bereits auf ein charakteristisches Merkmal von ASP.NET hingewiesen. ASP.NET ist durch und durch objektorientiert, und das bedeutet auch ereignisorientiert. Beim Laden, beim Anzeigen, beim Eingeben und Klicken – unablässig werden Ereignisse ausgelöst, auf die der Entwickler mit eigenen Routinen reagieren kann. `Page_Load` ist eine solche Ereignisprozedur, die in sehr vielen aspx-Seiten verwendet wird, weil Sie an dieser Stelle die Seite bequem initialisieren können.

Genau das machen Sie auch in `hallo02.aspx`. Über ihre `id` sind die beiden serverseitigen Elemente im Code bequem zugänglich, und Sie können den Text dynamisch zuweisen:

```
myTitle.InnerText = tmp
myH1.InnerText = tmp
```

Wo aber kommt `InnerText` her? `InnerText` ist eine Eigenschaft der Klasse `HtmlGenericControl`. Genauer gesagt hat die Klasse `HtmlGenericControl` diese Eigenschaft von der abstrakten Basisklasse `HtmlContainerControl` geerbt. ASP.NET hat das `title`- und das `h1`-Element in Objekte vom Typ `HtmlGenericControl` umgewandelt, als es im HTML-Code jeweils das Attribut `runat="server"` vorgefunden hat. Alle so genannten Container-Elemente verfügen über die Eigenschaft `InnerText`. Unter Container-Elementen versteht man solche HTML-Elemente, die selbst andere HTML-Elemente enthalten können, und das gilt für die meisten HTML-Elemente. Das `img`-Element wäre etwa ein Beispiel für ein HTML-Element, das kein Container-Element ist, denn es kann keine weiteren HTML-Elemente in sich aufnehmen.

Eine weitere nützliche Eigenschaft von `HtmlGenericControl` neben `innerText` ist `InnerHtml`. Damit können Sie dem jeweiligen Element nicht nur puren Text, sondern auch komplette HTML-Sequenzen zuweisen. Die Überschrift umbricht beispielsweise auf zwei Zeilen, wenn Sie folgenden Code verwenden:

```
myH1.InnerHtml = "Hallo, <br>Welt!"
```

Achtung Wenn Sie hier stattdessen `InnerText` verwendet hätten, würde im Browser der Text Hallo,
Welt! genau in dieser Form erscheinen. `InnerText` sorgt dafür, dass eventuelle Sonderzeichen, wie etwa die spitzen Klammern, in das jeweilige HTML-Äquivalent umgeformt werden. ASP.NET würde bei der Zuweisung

```
myH1.InnerText = "Hallo, <br>Welt!"
```

folgenden HTML-Code erzeugen:

```
<h1 id="myH1">Hallo, &lt;br&gt;Welt!</h1>
```

Sie können das selbst nachprüfen, wenn Sie den Skript-Code entsprechend anpassen, das Skript im Browser aufrufen und sich dann über **Ansicht · Quelltext** den HTML-Quellcode anzeigen lassen. Abbildung 4.3 zeigt alle drei Ansichten: den ASP.NET-Quellcode, die Darstellung im Browser und den HTML-Code, der im Browser angekommen ist.

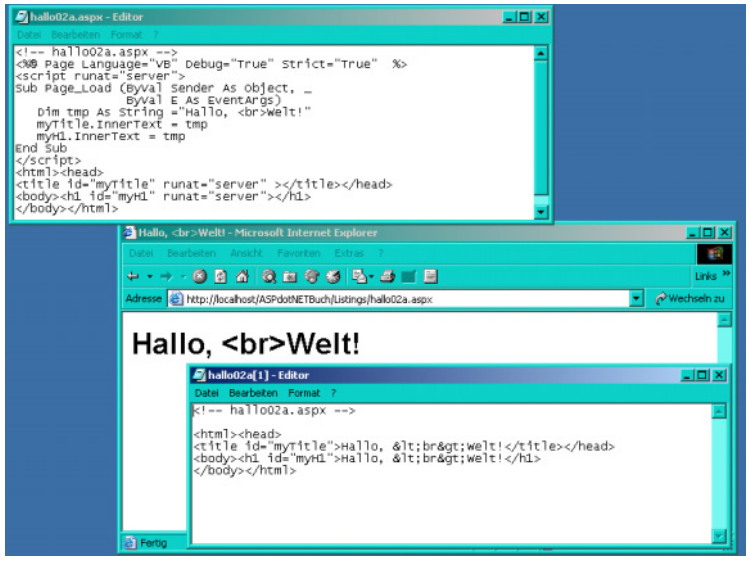

Abbildung 4.3 Die Eigenschaft InnerText zitiert den übergebenen Text. InnerHTML würde das
-Tag unverändert lassen.

4.2.3 »Hallo, Welt!« mit serverseitigen Steuerelementen

Im vorangegangenen Beispiel haben Sie den HTML-Code komplett selbst erstellt, und mit Hilfe des Skriptcodes haben Sie innerhalb eines HTML-Elements Text eingefügt. Noch mehr dem ASP.NET-Stil entspricht es, wenn Sie stärker von der HTML-Codierung abstrahieren. Vom typischen ASP.NET-Ansatz her würden Sie beispielsweise sagen: An dieser Stelle soll eine Meldung ausgegeben werden. Die Meldung soll so und so aussehen, und wie das am Ende in HTML codiert wird, interessiert mich eigentlich nicht. ASP.NET soll selbst den benötigten HTML-, CSS- und JavaScript-Code generieren.

Das Listing **hallo03.aspx** gibt eine zusätzliche Meldung aus, die mit Hilfe eines solchen serverseitigen Steuerelements realisiert wird. Abbildung 4.4 zeigt die Darstellung im Browser und den von ASP.NET generierten HTML-Code.

```
<!-- hallo03.aspx -->
<%@ Page Language="VB" Debug="True" Strict="True"  %>
<%@ Import Namespace="System.Drawing.Color" %>
<script runat="server">
Sub Page_Load (ByVal Sender As Object, _
               ByVal E As EventArgs)
   Dim tmp As String ="Hallo, Welt!"
   myTitle.innerText = tmp
   myH1.InnerText = "Hallo, Welt!"
   meldung.Text = "Oh, Du schöner Westerwald!"
   meldung.Font.Bold = True
   meldung.Font.Size = new FontUnit(12)
   meldung.BackColor = System.Drawing.Color.FromArgb _
                                          (200, 0, 100)
   meldung.ForeColor = System.Drawing.Color.LightGreen
End Sub
</script>
<html><head>
<title id="myTitle" runat="server" ></title></head>
<body><h1 id="myH1" runat="server"></h1>
<asp:Label id="meldung" runat="server" />
</body></html>
```

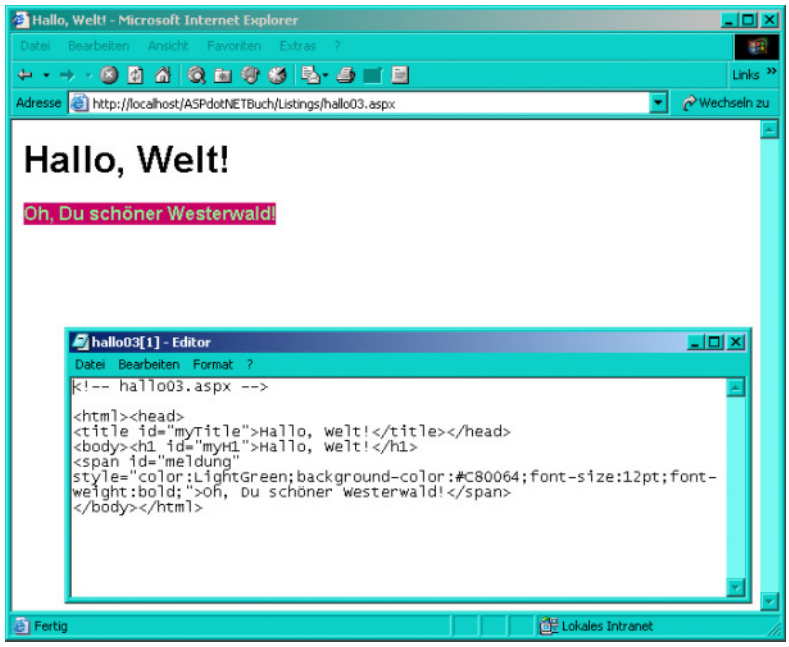

Abbildung 4.4 Serverseitige ASP.NET-Steuerelemente generieren selbstständig HTML-, CSS- und JavaScript-Code.

Dieses Beispiel zeigt ein grundlegendes ASP.NET-Prinzip recht deutlich. Der Entwickler schreibt in den HTML-Block der aspx-Datei lediglich ein recht abstraktes Element, das mit dem herkömmlichen HTML-Code nichts mehr zu tun hat:

```
<asp:Label id="meldung" runat="server" />
```

Mit Hilfe des Skriptcodes beeinflusst er die Eigenschaften dieses Elements. Das geschieht hier wieder innerhalb der Ereignisprozedur `Page_Load`:

```
meldung.Text = "Oh, Du schöner Westerwald!"
meldung.Font.Bold = True
meldung.Font.Size = new FontUnit(12)
meldung.BackColor = System.Drawing.Color.FromArgb _
                                        (200, 0, 100)
meldung.ForeColor = System.Drawing.Color.LightGreen
```

Die Umsetzung in HTML- und CSS-Code übernimmt ASP.NET. So entsteht:

```
<span id="meldung" style="color:LightGreen;background-
color:#C80064;font-size:12pt;font-weight:bold;">Oh, Du schöner
Westerwald!</span>
```

Achtung ASP.NET ist ein Code-Generator. Als gestandener Webentwickler muss man sich erst daran gewöhnen, dass ASP.NET einem die HTML-Bausteine aus der Hand nimmt und selbst anfängt zu bauen.

Wenn Sie es gewöhnt sind, dass im Browser exakt der HTML-Code ankommt, den Sie selbst codiert haben, müssen Sie sich bei ASP.NET umgewöhnen. Der erste Eindruck beim Blick in den HTML-Code, den ASP.NET generiert hat, ist oft eher »Und das soll von mir kommen?«

4.3 Formulare mit ASP.NET auswerten: »Hallo, Anwender!«

Die meisten Entwickler beginnen sich für die serverseitige Webprogrammierung zu interessieren, weil sie Webformulare auswerten wollen. Deswegen sollen Sie an dieser Stelle bereits eine ungefähre Vorstellung davon erhalten, wie ASP.NET mit Formularen umgeht.

4.3.1 ASP.NET begrüßt den Anwender

Zunächst ein einfaches Beispiel. In der Seite **name01.aspx** wird der Anwender nach seinem Namen gefragt. Wenn er ihn eingibt, begrüßt ASP.NET den Anwender mit seinem Namen.

```
<!-- name01.aspx -->
<%@ Page Language="VB" Debug="True" Strict="True"  %>
<script runat="server">
Sub Page_Load (ByVal Sender As Object, _
               ByVal E As EventArgs)
   If IsPostBack Then
      meldung.Text = "Hallo, " & txtName.Value
   End If
End Sub
</script>
<html><head><title>Begrüßung</title></head>
<body><h1>Begrüßung</h1>
<form runat="server">
Wie heißen Sie?
<br><br>
<input runat="server" id="txtName" type="text">
<br><br>
<input runat="server" type="submit" value=" OK " >
```

```
<br><br>
<asp:Label id="meldung" runat="server" />
</form></body></html>
```

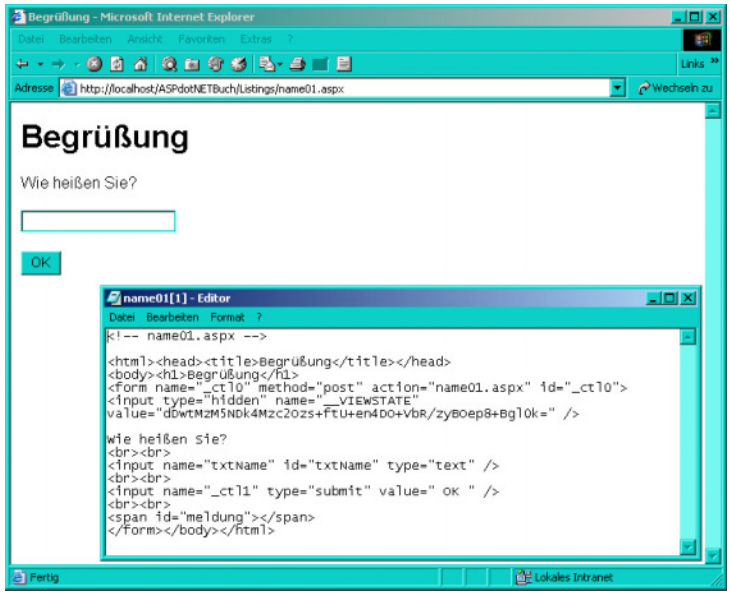

Abbildung 4.5 Das erste ASP.NET-Formular

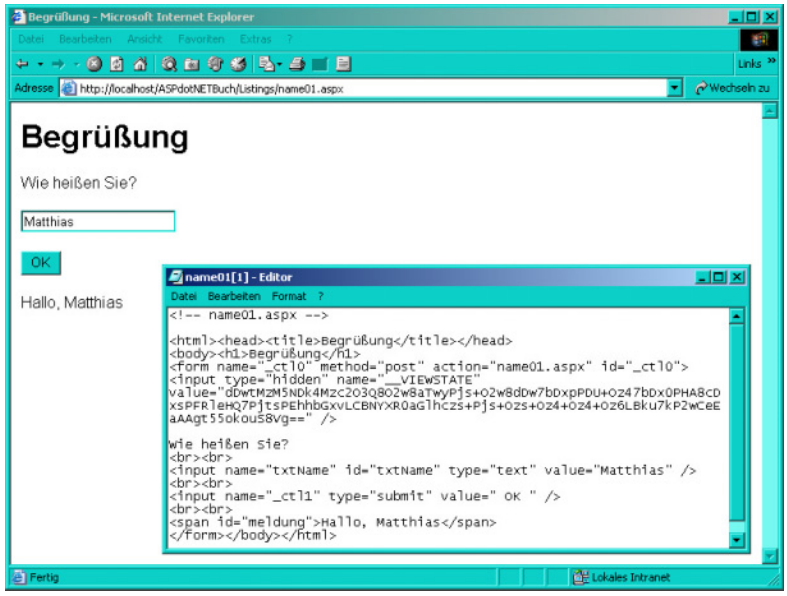

Abbildung 4.6 ASP.NET benutzt ein verstecktes VIEWSTATE-Feld, um den Formularstatus zu speichern.

Abbildung 4.5 zeigt die Darstellung beim ersten Aufruf des Formulars. Abbildung 4.6 zeigt den Browser, nachdem der Anwender seinen Namen eingegeben und **OK** angeklickt hat. Bei den Screenshots ist auch der jeweilige HTML-Code mit eingeblendet, den ASP.NET erzeugt hat.

Beim Blick auf den ASP.NET-Code einerseits und auf die erzeugten Seiten im Browser mit dem jeweiligen HTML-Code andererseits entstehen sicherlich viele Fragen. Sie können die Funktionsweise am ehesten nachvollziehen, wenn Sie mit der Code-Analyse dort anfangen, wo das Formular selbst definiert wird. In der aspx-Seite findet sich hier nur dieser Eintrag:

```
<form runat="server">
```

Wo aber ist das `action`-Attribut? Woher weiß das Skript eigentlich, welche Seite es aufrufen soll, wenn der Anwender den **OK**-Button anklickt? Ein Blick auf die Screenshots zeigt, dass ASP.NET automatisch diesen HTML-Code erzeugt hat:

```
<form name="_ctl0" method="post" action="name01.aspx" id="_ctl0">
```

ASP.NET verwendet für Formulare standardmäßig die Einstellung `method="post"` und ruft außerdem immer sich selbst auf.

Das dürfte nicht jedermanns Sache sein. Mancher Entwickler möchte beispielsweise lieber eine Seite `frage.aspx` entwickeln und die Antwort in `antwort.aspx` liefern.

Achtung Unter ASP.NET ruft ein Formular grundsätzlich immer sich selbst auf, weil im Formular die Ereignisprozeduren definiert sind, die das Formular abarbeiten muss. Wer von einer Seite zur nächsten wechseln möchte, kann das höchstens innerhalb dieser Ereignisprozeduren über die Anweisung `Response.Redirect` oder mit `Server.Transfer` erreichen.

Selbst wenn Sie im Code ausdrücklich das `action`-Attribut angeben, wird es von ASP.NET ignoriert und gegen den Namen der aufrufenden Seite ersetzt.

Der übrige HTML-Code der aspx-Seite ist einigermaßen nachvollziehbar. Ein Eingabefeld, eine **OK**-Schaltfläche und ein Platzhalter für die spätere Ausgabe werden bereitgestellt.

Der Anwender gibt einen Namen ein und klickt auf **OK**. Das Formular ruft sich selbst auf und gibt die Begrüßung aus. Woher weiß das Formular, dass der Anwender Daten eingegeben hat und dass es die Daten verarbeiten soll?

Des Rätsels Lösung liegt hier in der Ereignisprozedur `Page_Load`. Wie der Name vermuten lässt, wird `Page_Load` bei jedem Laden der Seite ausgeführt. In dieser

Prozedur prüft die Abfrage `If IsPostBack`, ob die Seite erstmalig aufgerufen wurde oder ob der Anwender Daten eingegeben und die Seite zur Verarbeitung zum Server zurückgeschickt hat. Hier spielt das versteckte Feld mit dem Namen `_VIEWSTATE` eine entscheidende Rolle. In diesem versteckten Feld speichert ASP.NET in codierter Form den Status einer Seite. Wenn der Status einer Seite erkennen lässt, dass der Anwender Daten eingegeben hat, dann trifft die Eigenschaft `IsPostBack` zu. Im Beispiel wird dann der Code innerhalb der `If`-Abfrage ausgeführt.

```
meldung.Text = "Hallo, " & txtName.Value
```

Diese Codezeile sorgt dafür, dass der Anwender mit seinem Namen begrüßt wird. En passant passieren noch ein paar weitere Kleinigkeiten. Auf der Begrüßungsseite bleibt der Name des Anwenders im Eingabefeld stehen. Auch das ist typisch für ASP.NET. Alle Eingaben des Anwenders bleiben in den entsprechenden Steuerelementen erhalten, ohne dass sich der Entwickler eigens darum kümmern müsste.

Außerdem erzeugt ASP.NET stets wohlgeformten XHTML-Code, was sich unter anderem daran zeigt, dass alle Tags auch wieder korrekt geschlossen werden. Die folgende Zeile ist kein korrektes XHTML, weil der abschließende Schrägstrich fehlt:

```
<input runat="server" type="submit" value=" OK " >
```

ASP.NET erzeugt daraus diesen Code:

```
<input name="_ctl1" type="submit" value=" OK " />
```

Neben dem abschließenden Schrägstrich ist erkennbar, dass ASP.NET offenkundig auch automatisch ein `name`-Attribut vergibt, wenn der Entwickler selbst kein `name`-Attribut definiert hat.

Achtung Wenn Sie für ein Element sowohl ein `name`-Attribut als auch ein `id`-Attribut definieren, übernimmt ASP.NET auch für das `name`-Attribut den Wert des `id`-Attributs. Hier ein Beispiel. Ihr Code lautet:

```
<input runat="server" type="text"
       id="txtMeineID"
       name="txtMeinName">
```

Daraus erzeugt ASP.NET folgenden Code:

```
<input name="txtMeineID" id="txtMeineID" type="text" />
```

Das kann tückisch sein, wenn Sie mit JavaScript-Code über das `name`-Attribut auf einzelne Elemente zugreifen wollten und alle Ihre `name`-Attribute auf einmal völlig andere Inhalte haben.

Tipp Daher mein Tipp: Verwenden Sie das `name`-Attribut nicht mehr. Verwenden Sie nur noch das `id`-Attribut. ASP.NET erzeugt dann stets zusätzlich ein `name`-Attribut mit identischem Inhalt. Die einzige Ausnahme bilden Optionsfelder. Hier benötigen Sie das `name`-Attribut, um die Zugehörigkeit eines Optionsfeldes zu einer Optionsfeldgruppe festzulegen.

4.3.2 Ein Grundgerüst für ASP.NET-Formulare

Der Aufbau der aspx-Seite aus dem vorherigen Abschnitt lässt sich als Grundgerüst für viele Formulare verwenden. Auf die minimale Form gebracht, ergibt sich diese Form:

```
<!-- grundgeruest.aspx -->
<%@ Page Language="VB" Debug="True" Strict="True"  %>
<script runat="server">
Sub Page_Load (ByVal Sender As Object, _
               ByVal E As EventArgs)
   If Not IsPostBack Then
      ' Hier steht der Code, der nur beim ersten Laden
      ' der Seite ausgeführt wird, z.B. kann die Seite
      ' hier mit Startwerten etc. initialisiert werden.
   Else
      ' Hier steht der Code, der nach dem
      ' Formularversand ausgeführt wird, d.h. hier
      ' werden die Eingaben des Anwenders verarbeitet.
   End If
End Sub
</script>
<html><head><title>Titel</title></head>
<body><h1>Titel</h1>
<form runat="server" id="myForm">
<!-- Hier steht der Inhalt des Formulars. -->
</form>
</body></html>
```

Wichtig ist das `form`-Element im HTML-Block der aspx-Seite, das mit dem Attribut `runat="server"` versehen sein muss. Nur wenn dieses serverseitige `form`-

Element definiert ist, kann ASP.NET Formulare korrekt verarbeiten. Die folgende Angabe können Sie in allen Formularen verwenden:

```
<form runat="server" id="myForm">
```

Das `action`-Attribut dürfen Sie nicht definieren. ASP.NET fügt hier automatisch den Namen der aktuellen aspx-Seite selbst ein. ASP.NET lässt für jede aspx-Seite nur ein einziges `form`-Element zu.

Die Eingaben des Anwenders können Sie am leichtesten im Rahmen der Ereignisprozedur `Page_Load` bearbeiten. Dafür muss innerhalb dieser Prozedur die `IsPostBack`-Eigenschaft überprüft werden. Beim erstmaligen Laden der Seite ist die Eigenschaft `False`. In diesem Zweig der `If`-Abfrage können Sie die Seite initialisieren. Wenn der Anwender irgendwelche Daten eingegeben und abschließend **OK** angeklickt hat, wird die Seite zum Server versandt. Beim erneuten Laden der Seite ist die Eigenschaft `IsPostBack True`. In diesem Zweig der `If`-Abfrage können Sie die Eingaben des Anwenders auswerten.

Viele Beispiele in diesem Buch bauen auf diesem Grundgerüst auf. Sie können es auch gut für Ihre eigenen Experimente verwenden.

4.3.3 Validierung von Anwendereingaben

Wenn der Anwender in dem Beispiel mit der Begrüßung überhaupt nichts eingibt, sondern nur **OK** anklickt, wird die wenig sinnvolle Meldung **Hallo,** zurückgegeben. Vor der Bearbeitung sollte das Skript überprüfen, ob der Anwender einen Namen eingegeben hat. Wenn nicht, dann sollte eine Fehlermeldung den Anwender darauf hinweisen.

Unter ASP.NET ist die Realisierung solcher Validierungsfunktionen sehr einfach. Im Beispiel müssen Sie lediglich hinter dem Text-Eingabefeld ein Webserversteuerelement vom Typ `RequiredFieldValidator` ergänzen, in der `Page_Load`-Ereignisprozedur die Seite validieren und entsprechend dem Ergebnis der Überprüfung reagieren. Die aspx-Seite gewinnt damit diese Form:

```
<!-- name02.aspx -->
<%@ Page Language="VB" Debug="True" Strict="True"  %>
<script runat="server">
Sub Page_Load (ByVal Sender As Object, _
               ByVal E As EventArgs)
   If IsPostBack Then
      Page.Validate
      If IsValid Then
         meldung.Text = "Hallo, " & txtName.Value
```

```
      End If
   End If
End Sub
</script>
<html><head><title>Begrüßung</title></head>
<body><h1>Begrüßung</h1>
<form runat="server">
Wie heißen Sie?
<br><br>
<input runat="server" id="txtName" type="text" >
<asp:RequiredFieldValidator
     id="reqTxtName"
     ControlToValidate="txtName"
     Display="dynamic"
     runat="server">
     Bitte geben Sie hier Ihren Namen ein.
</asp:RequiredFieldValidator>
<br><br>
<input runat="server" type="submit" value=" OK " >
<br><br>
<asp:Label id="meldung" runat="server" />
</form></body></html>
```

Abbildung 4.7 Für die Validierung von Anwendereingaben stellt ASP.NET zahlreiche Steuerelemente zur Verfügung.

Das Formular reagiert jetzt wie gewünscht. Wenn der Anwender nichts eingibt, erscheint neben dem Feld in roter Schrift der Hinweis: **Bitte geben Sie hier Ihren Namen ein.** Wenn der Anwender den Namen eingibt und **OK** anklickt, verschwindet der Hinweis und die Begrüßung erscheint.

Wenn im Browser des Anwenders JavaScript aktiviert ist, findet die Überprüfung bereits im Webbrowser mit Hilfe von JavaScript statt. Auf jeden Fall wird die Überprüfung auch auf der Serverseite vorgenommen.

Wenn Sie sicherstellen möchten, dass der Anwender im Eingabefeld `txtName` auf jeden Fall eine Eingabe vornimmt, dann bleibt für Sie als Entwickler nicht mehr viel zu tun. Sie definieren im HTML-Block der aspx-Seite das Element `asp:RequiredFieldValidator` und vergeben neben `runat="server"` das Attribut `ControlToValidate="txtName"`. Das reicht bereits aus. Mit der Umsetzung in clientseitigen und serverseitigen Code haben Sie sonst nichts mehr zu tun.

Es lohnt sich, einmal den generierten HTML/JavaScript-Quellcode anzusehen. Erschrecken Sie aber nicht!

```
<!-- name02.aspx -->
<html><head><title>Begrüßung</title></head>
<body><h1>Begrüßung</h1>
<form name="_ctl0" method="post" action="name02.aspx" langu-
age="javascript" onsubmit="ValidatorOnSubmit();" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTI1Mjg5Mjg4OTs7PiKrEhu30GwCswlvC944CRBKCSeL" />
<script language="javascript" src="/aspnet_client/system_web/1_
0_3705_0/WebUIValidation.js"></script>
Wie heißen Sie?
<br><br>
<input name="txtName" id="txtName" type="text" />
<span id="reqTxtName" CausesValidation="True" controltovali-
date="txtName" display="Dynamic" evaluationfunction="Required-
FieldValidatorEvaluateIsValid" initialvalue=""
style="color:Red;display:none;">
     Bitte geben Sie hier Ihren Namen ein.
</span>
<br><br>
<input language="javascript" onclick="if (typeof(Page_ClientVa-
lidate) == 'function') Page_ClientValidate(); " name="_ctl1"
type="submit" value=" OK " />
<br><br>
```

```
<span id="meldung"></span>
<script language="javascript">
<!--
   var Page_Validators =  new Array(document.all["reqTxtName"]);
   <$tab>// -->
</script>
<script language="javascript">
<!--
var Page_ValidationActive = false;
if (typeof(clientInformation) != "undefined" && clientInforma-
tion.appName.indexOf("Explorer") != -1) {
    if (typeof(Page_ValidationVer) == "undefined")
        alert("Unable to find script library '/aspnet_client/
system_web/1_0_3705_0/WebUIValidation.js'. Try placing this file
manually, or reinstall by running 'aspnet_regiis -c'.");
    else if (Page_ValidationVer != "125")
        alert("This page uses an incorrect version of WebUIVali-
dation.js. The page expects version 125. The script library is "
+ Page_ValidationVer + ".");
    else
        ValidatorOnLoad();
}
function ValidatorOnSubmit() {
    if (Page_ValidationActive) {
        ValidatorCommonOnSubmit();
    }
}
// -->
</script></form></body></html>
```

Diesen Code haben Sie selbst mit Hilfe von ASP.NET erstellt! Und dabei ist der eigentliche Code, der die Eingabeüberprüfung vornimmt, hier noch gar nicht enthalten. Er ist in der externen Skript-Datei `WebUIValidation.js` enthalten.

Keine Angst, den generierten Quellcode gehe ich jetzt nicht Zeile für Zeile durch. Für das Verständnis von ASP.NET ist es auch nicht erforderlich, sich in die Details dieser JavaScript-Routinen zu vertiefen. Als ASP.NET-Entwickler sind Sie dafür zuständig, die Programmlogik festzulegen. Die Umsetzung in den entsprechenden HTML- und JavaScript-Code nimmt ASP.NET für Sie vor.

Neben dem `RequiredFieldValidator` bietet ASP.NET noch zahlreiche andere Webserversteuerelemente für die Validierung. Sie können beispielsweise über-

prüfen, ob die Eingabe zu einem bestimmten Datentyp passt, ob die Eingabe einem Muster entspricht, ob sie in einen bestimmten Bereich fällt und manches andere. Kurz und gut: Aus der Mühe, die Anwendereingaben zu überprüfen, wird mit ASP.NET ein Vergnügen.

4.4 Zusammenfassung und Ausblick

Damit ist der Crashkurs beendet. Sie haben den grundlegenden Aufbau von ASP.NET-Seiten kennen gelernt. Sie haben einige Beispielseiten gesehen und auch schon ein erstes Formular verarbeitet. Vielleicht konnten Sie bereits einen ersten Eindruck davon gewinnen, wie ASP.NET arbeitet. In den folgenden Kapiteln werden Sie ASP.NET von Grund auf kennen lernen.

An dieser Stelle möchte ich einen kurzen Ausblick geben, mit welchen Themen es weitergeht. Die wichtigste Grundlage für das Erstellen von ASP.NET-Applikationen ist die Kenntnis der serverseitigen Steuerelemente. ASP.NET kennt zwei Typen von serverseitigen Steuerelementen:

- HTML-Serversteuerelemente und
- Webserversteuerelemente.

HTML-Serversteuerelemente haben Sie bereits im Abschnitt 4.2.2, »Hallo, Welt!« im ASP.NET-Stil, kennen gelernt. Es ist sehr einfach, HTML-Serversteuerelemente zu erstellen. Fügen Sie einfach zu einem beliebigen HTML-Element den Zusatz `runat="server"` hinzu, und Sie haben ein HTML-Serversteuerelement erstellt. ASP.NET verarbeitet HTML-Serversteuerelemente mit Hilfe spezialisierter Klassen. Diese Klassen werden Sie im folgenden Kapitel genauer kennen lernen.

Webserversteuerelemente abstrahieren stärker vom HTML-Code. Hier schreibt der Entwickler keine HTML-Tags mit dem Zusatz `runat="server"` in den HTML-Block der aspx-Seite, sondern er setzt HTML-unabhängige Elemente in den HTML-Block. Webserversteuerelemente können Sie daran erkennen, dass sie mit `<asp:` beginnen. Bei der Verarbeitung erzeugt ASP.NET aus Webserversteuerelementen selbstständig den HTML-Code. Wenn Sie Webserversteuerelemente verwenden, müssen Sie kein HTML-Experte mehr sein. Solche Details erledigt ASP.NET für Sie.