9 Ein erstes „Hello World“-Projekt

Nach all der Theorie wollen wir uns nun an ein erstes Beispielprojekt heranwagen. In diesem Projekt werden wir eine kleine „Webseite“ erstellen, die einen Button enthält, der bei einem Klick einen Text anzeigt.

9.1 Ein neues Projekt in Visual Studio anlegen

Öffnen Sie Visual Studio. Auf dem Startbildschirm von Visual Studio können Sie links auf NEUE WEBSITE klicken, oder Sie wählen über das Menü DATEI → NEUE WEBSITE aus (Bild 9.1).

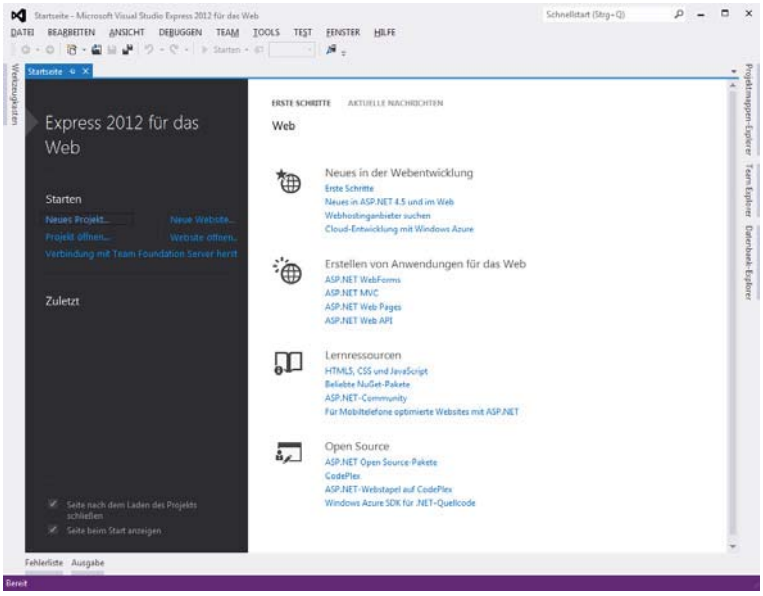

Bild 9.1
Visual Studio Startbildschirm

Nun öffnet sich ein neues Fenster, in dem Sie die Sprache und Art des Projekts sowie den Pfad, in dem es gespeichert werden soll, auswählen können (Bild 9.2). Im Buch werden wir immer zuerst das VB.NET-Beispiel und danach das C#-Beispiel angehen. Sie müssen aber natürlich nicht beide durchgehen oder nachbauen. Falls Sie sich schon nach den beiden Einführungen für eine der beiden Sprachen entschieden haben, wählen Sie nun in der linken Spalte unter Installiert → Vorlagen die gewünschte Sprache aus. In der mittleren Spalte wählen Sie *Leere ASP.NET-Website* aus. Unten können Sie den Pfad auswählen, in dem Sie den Quelltext ablegen wollen. Danach klicken Sie auf OK.

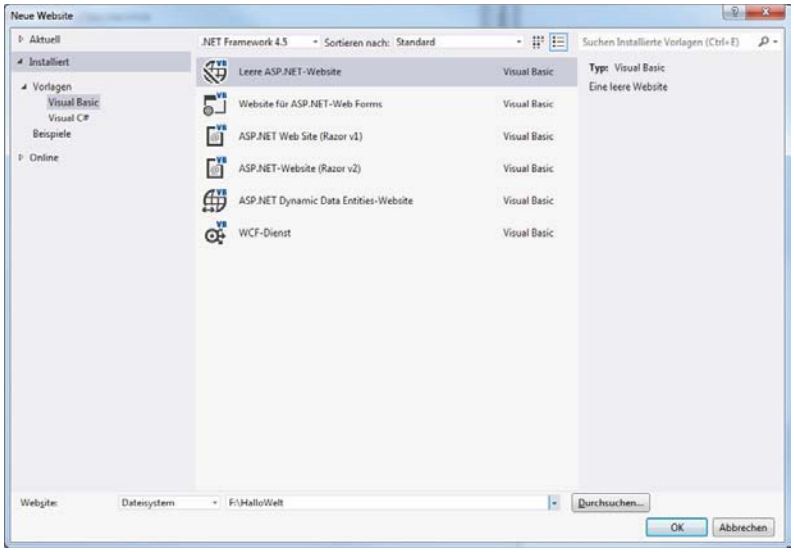

Bild 9.2
Neue Website erstellen

Nun öffnet Visual Studio das neu erstelle Projekt. Rechts sehen Sie einen Reiter, der *Projektmappen-Explorer* heißt. Wenn Sie dort draufklicken, sehen Sie alle aktuellen Dateien des Projekts. Derzeit hat unser Projekt nur eine Datei, die *web.config*-Datei.

web.config-Datei

Die *web.config*-Datei ist die Hauptkonfigurationsdatei unseres Webprojekts. Es können in Unterordnern auch *web.config*-Dateien liegen, die dann bestimmte Konfigurationen für den Ordner und die Unterordner beinhalten. Die Datei ist ein XML-Dokument, und in ihr werden grundlegende Einstellungen zum Projekt oder auch bestimmte Module eingebunden und Datenbank-Verbindungsstrings abgelegt. Zur *web.config*-Datei werden im Laufe des Buches immer wieder ein paar Einträge hinzukommen, die dann auch in dem entsprechenden Schritt erklärt werden, z. B. wenn wir eine Datenbank anbinden.

9.2 Eine Webform hinzufügen

Um nun in unserer Website etwas programmieren zu können, brauchen wir eine Webform. Webforms sind Seiten, die aus HTML-Code, Controls (Steuerelementen) und Server Code (Code-behind) bestehen. Hierbei wird auch der Benutzeroberflächencode, der im Browser angezeigt wird, getrennt vom Servercode abgelegt, was eine Website sehr übersichtlich für Programmierer macht.

Um eine neue Webform anzulegen, klicken Sie bitte im Menü auf Website und dort auf Neues Element hinzufügen. Nun erscheint wieder ein ähnlicher Dialog wie in Bild 9.2. In diesem wählen Sie links die Sprache aus, in der Ihr Projekt erstellt wurde (im Normalfall ist hier schon die richtige vorausgewählt), in der Mitte wählen Sie *Webform* aus, und unten beim Namen tragen Sie *Default.aspx* ein. Rechts unten setzen Sie noch einen Haken bei *Code in gesonderter Datei ablegen* und klicken auf OK. Jetzt sollten Sie im Projektmappen-Explorer oberhalb der *web.config*-Datei eine neue Datei namens *Default.aspx* sehen. Links neben dem Dateinamen befindet sich ein Pfeil, mit dem sich die Datei ausklappen lässt. Nun erscheint unterhalb der *Default.aspx*-Datei eine weitere Datei *Default.aspx.vb*. In dieser *.aspx.vb*-Datei wird der Servercode abgespeichert, und in der *.aspx*-Datei werden der HTML-Code, die Steuerelemente und alles Weitere abgelegt, was im Browser erscheint bzw. eingebunden wird, auch JavaScript und CSS. Durch einen Klick auf die Datei *Default.aspx* öffnet Visual Studio die Datei, wobei es drei Varianten gibt, diese anzuzeigen:

- *Quelle:* Hier sieht man nur den HTML-Quelltext der Datei.
- *Entwurf:* Hier sieht man eine Vorschau der HTML-Datei wie im Browser, allerdings mit einigen Hilfslinien und Hilfseinblendungen, um das Erstellen der Oberfläche zu erleichtern.
- *Teilen:* Dies ist ein geteilter Modus, in dem oben Variante 1 und unten Variante 2 zu sehen ist.

Wechseln Sie nun in den Quelltext-Modus. Hier sehen Sie nun das Grundgerüst einer HTML-Seite. Allerdings ist in der ersten Zeile eine Neuerung zu sehen, die wir bisher nicht kannten. In dieser Zeile wird definiert, in welcher Sprache der Servercode der Code-behind-Dateien geschrieben wird und in welcher Datei dieser steht (Listing 9.1 und Listing 9.2).

Listing 9.1 Default.aspx VB.NET

```
<%@ Page Language="VB" CodeFile="Default.aspx.vb" Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
```

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

Listing 9.2 Default.aspx C#

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

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

Diese erste Zeile wird als *Page Direktive* bezeichnet. Direktiven sind für den Compiler Anweisungen, wie dieser Controls oder Seiten verarbeiten soll. In diesem Fall geben wir mit dem Attribut `Language` an, in welcher Sprache die Seite geschrieben ist. Mit `CodeFile` geben wir an, in welcher Datei der Code-behind-Code liegt, und mit `Inherits` geben wir die Klasse an, in der der Code der Seite liegt. Diese Zeile wird in Visual Studio automatisch generiert und muss in Projekten wie denen in diesem Buch eigentlich nicht manuell angepasst werden.

Als Nächstes werden wir unserer Webform einen Button hinzufügen. Dazu wechseln wir in die Entwurfsansicht und klicken links am Bildschirmrand auf den Reiter *Werkzeugkasten*. Hier sehen Sie eine Liste mit Steuerelementen, die Sie der Webform hinzufügen können. Diese sind alle standardmäßig bei ASP.NET dabei, können aber noch durch Drittanbieter-Module erweitert werden. Im ersten Schritt begnügen wir uns aber mit den Standardsteuerelementen. Um den Button nun hinzuzufügen, ziehen wir das Steuerelement *Button* per Drag & Drop in den `div`-Container in unserem HTML-Quelltext. Wenn wir ihn dann dort abgelegt haben, generiert Visual Studio automatisch den benötigten Quelltext, um den Button im Browser anzuzeigen (Listing 9.3).

Listing 9.3 Button-Code

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Der hier hinzugefügte Button gehört zu den Websteuerelementen, diese Websteuerelemente generieren beim Ausführen HTML- und JavaScript-Code. Es gibt komplexe und weniger komplexe Websteuerelemente. Ein Button wie der gerade eingebaute wird nicht so viel Code generieren wie beispielsweise ein *Calendar Control*. Websteuerelemente erkennen Sie immer an dem Präfix `asp:` und dem Attribut `runat="server"`. Auf Elemente mit diesem Attribut können Sie dann im Code-behind zugreifen. Aus unserem oberen Button wird im HTML-Code des Browsers dann dieses HTML-Element (Listing 9.4).

Listing 9.4 Button-Code in HTML gerendert

```
<input type="submit" name="Button1" value="Button" id="Button1" />
```

Um unser „Hallo Welt“-Projekt noch weiter auszubauen, brauchen wir noch ein Element, in dem unsere Nachricht angezeigt wird. Dieses Element ist ein Label. Ziehen Sie jetzt ein Label vom Werkzeugkasten in Ihre Quellansicht unter den Button. Visual Studio generiert nun wieder den Code des hinzugefügten Elements (Listing 9.5).

Listing 9.5 Label-Code

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

Jetzt haben wir alles, was wir brauchen, um auf einen Button zu klicken, der uns daraufhin eine Nachricht anzeigt. Allerdings wollen wir noch eine Kleinigkeit ändern, um die Übersicht zu verbessern. Die beiden hinzugefügten Elemente haben eine Attribut-ID, diese ID lässt uns später im Programmcode auf das Element zugreifen. Wenn man irgendwann mal viele solcher Elemente auf einer Seite hat, wird es durch die Bezeichnungen Label1, Label2, Label3 usw. ziemlich schnell unübersichtlich, und man weiß nicht mehr, welcher Bezeichner für welches Element steht. Deshalb geben wir unseren Elementen nun aussagekräftigere Namen. Dafür klicken Sie im Quelleneditor auf den Code des Buttons und drücken die F4-Taste. Dadurch springt der Fokus von Visual Studio in das Eigenschaftsfenster des Elements. Hier sehen Sie nun alle Eigenschaften, die solch ein Button hat. Sie können hier unter anderem die Farbe der Schrift oder die Größe der Schrift der Button-Bezeichnung einstellen, oder auch, wie breit und hoch er sein soll oder ob er sichtbar sein soll oder deaktiviert und vieles Weitere. Auf diese Eigenschaften können Sie später auch im Codebehind zugreifen und sie auslesen oder verändern. Wir wollen jetzt nur die ID ändern und klicken dafür in die Zeile, in der `ID` steht, und rechts davon in die Spalte, in der `Button1` steht. Als neue Bezeichnung tragen Sie bitte `btnShowHello` ein. Das `btn` steht hierbei als Abkürzung für Button, und mit `ShowHello` beschreiben wir kurz die Funktion des Buttons. Wenn Sie jetzt irgendwo anders im Visual Studio klicken, wird Visual Studio die ID des Buttons sofort im Code ändern. Als Nächstes ändern wir noch die ID des Labels auf `lblHelloWorld`, `lbl` steht in diesem Fall für Label. Als Letztes löschen wir noch den Text der Label-Eigenschaft *Text*. Das können Sie im Eigenschaftsfenster machen, indem Sie einfach den Text rauslöschen, oder Sie ändern es im Quellcode, indem Sie aus `Text="Label"` `Text=""` machen. Die letzte Eigenschaft, die wir ändern, ist die Visible-Eigenschaft. Setzen Sie diese bitte auf `False`. Diese Eigenschaft gibt an, ob das Element im Browser sichtbar ist.

Ihre *Default.aspx*-Seite sollte nun in VB.NET wie in Listing 9.6 aussehen.

Listing 9.6 Default.aspx mit Controls in VB

```
<%@ Page Language="VB" CodeFile="Default.aspx.vb" Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
   <title></title>
</head>
```

```
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Button ID="btnShowHello" runat="server" Text="Button" />
      <asp:Label ID="lblHelloWorld" runat="server" Text=""
                Visible="False"></asp:Label>
   </div>
   </form>
</body>
</html>
```

In C# sieht das Ganze bis auf einen Unterschied des `Language`-Attributs in der Page Direktiven genau gleich aus (Listing 9.7).

Listing 9.7 Default.aspx mit Controls in C#

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Button ID="btnShowHello" runat="server" Text="Button" />
      <asp:Label ID="lblHelloWorld" runat="server" Text=""
                Visible="False"></asp:Label>
   </div>
   </form>
</body>
</html>
```

Wenn Sie jetzt im Visual Studio unten links auf *Entwurf* klicken, sehen Sie eine Entwurfsansicht Ihrer Website. Hier können Sie genauso die Elemente anklicken und ihre Eigenschaften ändern. Wenn Sie auf den Button klicken, erscheinen solche weißen Quadrate, an denen Sie ziehen können, um die Größe zu verändern. Auch können Sie einfach die Anordnung der Elemente per Drag & Drop verändern.

9.3 Website ausführen

Im nächsten Schritt lassen wir die Website im Browser öffnen. Dafür starten wir das Debugging. Als Debugging bezeichnet man den Vorgang, einen Programmcode auf Fehler abzusuchen und diese zu entfernen. Wenn Sie später einmal komplexere Programme oder Seiten schreiben, wird es gelegentlich passieren, dass etwas nicht funktioniert, wie es soll. In diesem Fall sagt Ihnen, wenn alles gut läuft, Ihr Debugger, an welcher Stelle etwas nicht in Ordnung ist. Eine andere Möglichkeit ist es, den Quellcode Schritt für Schritt und Zeile für Zeile während der Ausführung zu überwachen. Wenn man im Visual Studio den Debugger

startet, wird versucht, das Projekt im Browser zu öffnen; wenn dies nicht funktioniert, wird Visual Studio im Normalfall in einer Fehlerliste aufzeigen, wo die Probleme liegen. Um ein Projekt zu debuggen, gibt es mehrere Möglichkeiten: Entweder klicken Sie oben in der Symbolleiste auf den grünen *Play*-Button oder im Menü *Debuggen* auf DEBUGGING STARTEN, oder Sie drücken einfach die F5-Taste. Wenn Sie das Debugging zum ersten Mal ausführen, wird Visual Studio Sie möglicherweise fragen, ob Sie den Debugging-Modus aktivieren wollen. Dies bestätigen Sie bitte, dadurch wird in Ihrer *web.config*-Datei das Debugging für dieses Projekt aktviert.

Wenn Ihr Projekt keine Fehler enthält, sollten Sie nun im Browser eine weiße Seite mit einem Button sehen (Bild 9.3).

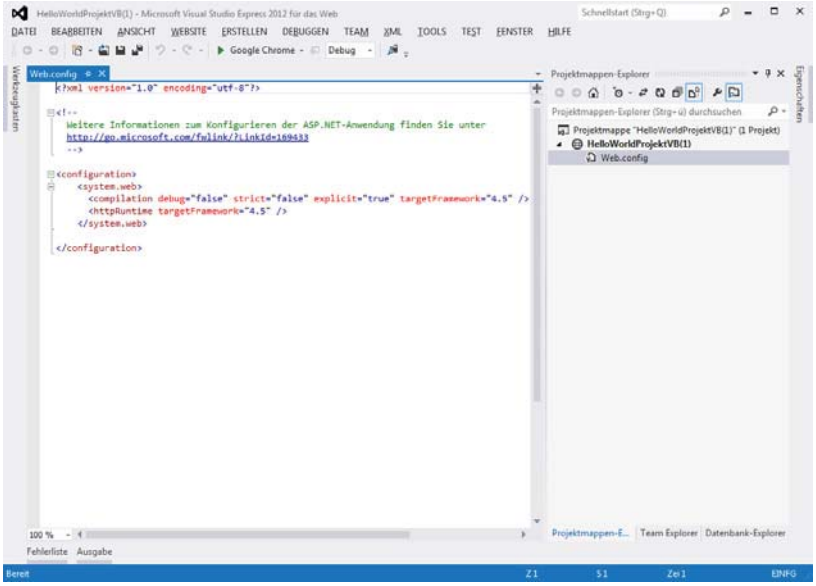

Bild 9.3 „Hello World"-Projekt im Browser

Leider kann diese Seite jetzt nicht wirklich viel, aber das wollen wir ändern.

Code-behind

Um unseren Button dazu zu bewegen, unsere „Hello World"-Nachricht anzuzeigen, müssen wir im Code-behind dazu die nötigen Anweisungen geben. Was soll passieren? Wenn man auf den Button klickt, soll das Label die Meldung „Hello World" ausgeben. Dafür müssen wir die Eigenschaft *Text* und die Eigenschaft *Visible* bei einem Klick auf den Button ändern.

Die einzelnen Elemente haben sogenannte Events (Ereignisse). Diese Events sind zum Beispiel beim Button das Click-Event, das ausgeführt wird, wenn man auf den Button klickt, oder bei das Load-Event bei Seiten, das beim Laden der Seite ausgeführt wird. In unserem Fall nutzen wir das Click-Event des Buttons, denn wir wollen ja, dass beim Klick auf den Button etwas passiert. Um in dieses Event nun Programmcode zu schreiben, doppelklicken Sie im Entwurfsmodus auf den Button, der etwas tun soll. Visual Studio erstellt dann automatisch in Ihrem Projekt den nötigen Code, um in das Event eingreifen zu können (Listing 9.8).

Listing 9.8 Click Event in VB.NET

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub btnShowHello_Click(sender As Object, e As EventArgs) Handles
btnShowHello.Click

    End Sub
End Class
```

In der ersten Zeile in Listing 9.8 wird eine Klasse erstellt, dessen Bezeichner wir auch in unserer Page Direktiven der *aspx*-Seite wiederfinden, hierdurch wird die Verknüpfung der *aspx*-Seite mit der Code-behind-Seite hergestellt. In der zweiten Zeile erbt unsere Klasse `_Default` von der Klasse `System.Web.UI.Page`, dadurch wird unsere Klasse zu einer ASP.NET-Page und erhält die Eigenschaften und Ereignisse usw. von dieser. In der dritten Zeile wird eine Prozedur erstellt, die ausgeführt wird, wenn auf den Button geklickt wird. Hier begegnen uns einige neue Dinge. Als Erstes ist dies das Schlüsselwort `Protected`. Dieses Schlüsselwort legt den Gültigkeitsbereich der Prozedur fest. Der Gültigkeitsbereich sagt aus, von wo in einem Projekt eine Variable, Klasse, Prozedur usw. aufgerufen werden kann. `Protected` bedeutet, dass diese Prozedur nur aus der eigenen Klasse oder aus Klassen, die von dieser Klasse erben, aufgerufen werden kann. Ein weiterer Gültigkeitsbereich wäre `Public`. Eine `Public Sub` kann von überall aus dem Projekt aufgerufen werden. Als Nächstes werden zwei Argumente übergeben, `sender As Object` und `e As EventArgs`. Die Variable `sender` referenziert das aufrufende Steuerelement. Die Variable `e` vom Typ `EventArgs` enthält mögliche zusätzliche Parameter, die beim Aufruf der Prozedur übergeben werden. Bei unserem Button ist die Variable leer, aber bei einem `ImageButton` würden hier die Koordinaten übergeben werden, an welcher Stelle wir auf das Bild geklickt hätten. Als Letztes kommt die `Handles`-Anweisung, hierbei wird die Prozedur an das `Click`-Event von `btnShowHello` gebunden. Zwischen dieser Zeile und der Zeile `End Sub` können wir nun den Code platzieren, der ausgeführt werden soll, wenn der Button geklickt wird.

Listing 9.9 Click Event in C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

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

    }
    protected void btnShowHello_Click(object sender, EventArgs e)
    {

    }
}
```

In Listing 9.9 sehen wir nun im Gegensatz zur *aspx*-Seite doch einige Unterschiede zum VB.NET-Code. In den ersten Zeilen befinden sich mehrere `using`-Anweisungen. Diese importieren verschiedene Namespaces, die benötigt werden, um die Seite funktionieren zu lassen. Die hier stehenden `using`-Anweisungen wurden alle automatisch eingefügt. Danach folgt die Erstellung der Klasse der Seite, das ist analog zu VB.NET, und es wird auch von `System.Web.UI.Page` geerbt. Innerhalb der Klasse wurde automatisch eine Funktion erstellt, die für das Load-Event der Seite zuständig ist. In diese Funktion schreiben wir für dieses Beispiel nichts hinein. Danach folgt die Funktion, die für das Click-Event des Buttons zuständig ist. Die Parameter und der Gültigkeitsbereich sind die Gleichen wie bei VB.NET. Doch im Gegensatz zu VB.Net fehlt die `Handles`-Anweisung, damit die Funktion an den Button gebunden wird. Diese Bindung findet bei C# an einer anderen Stelle statt. Dazu wechseln wir zurück in die *.aspx*-Datei. In der Quellenansicht sehen wir, dass dem Button ein neues Attribut hinzugefügt wurde: `OnClick="btnShowHello_Click"`. Durch dieses Attribut wird der Ereignishandler dem Ereignis zugewiesen, also die Funktion, die die Aufgabe des Ereignisses ausführt. In diesem Fall ist es die Funktion `btnShowHello_Click` aus der *.aspx.cs*-Datei.

Jetzt fehlen noch die Anweisungen innerhalb der Funktionen.

Listing 9.10 Click-Event in VB.NET

```
Protected Sub btnShowHello_Click(sender As Object, e As EventArgs) Handles
btnShowHello.Click
    'Der Text-Eigenschaft des Labels
    'den Wert "Hello World" zuweisen
    Me.lblHelloWorld.Text = "Hello World"
    'Die Visible-Eigenschaft auf true
    'setzen, damit das Label sichtbar wird
    Me.lblHelloWorld.Visible = True
End Sub
```

In der ersten Zeile in Listing 9.10 weisen wir dem Label über die Text-Eigenschaft „Hello World“ zu. Vor dem Namen des Labels steht das Schlüsselwort `Me`. Mit `Me` greift man auf die Elemente der aktuellen Klasse zu, in diesem Fall die Elemente der Page-Klasse, in der wir uns gerade befinden. Mit dem Punkt als Trenner kann man auf die Unterelemente oder die Eigenschaften eines Elements zugreifen. `lblHelloWorld` ist ein Element innerhalb von `Me`, und `Text` ist eine Eigenschaft von `lblHelloWorld`. Nach dem =-Zeichen folgt dann der neue Wert für die Eigenschaft. In der nächsten Zeile ändern wir noch die Eigenschaft `Visible` auf `true`.

Listing 9.11 Click Event C#

```
protected void btnShowHello_Click(object sender, EventArgs e)
{
    //Der Text-Eigenschaft des Labels
    //den Wert "Hello World" zuweisen
    this.lblHelloWorld.Text = "Hello World";
    //Die Visible-Eigenschaft auf true
    //setzen, damit das Label sichtbar wird
    this.lblHelloWorld.Visible = true;
}
```

In C# sieht der Code ähnlich aus wie in VB.NET, nur wird in C# anstatt `Me` das Schlüsselwort `this` verwendet (Listing 9.11). Die Objekte und Eigenschaften heißen natürlich gleich wie im VB.NET-Beispiel.

Wenn Sie nun den Debug-Modus starten, sollte sich die Seite öffnen und bei einem Klick auf den Button das Label mit dem zugewiesenen „Hello World“-Text sichtbar werden (Bild 9.4).

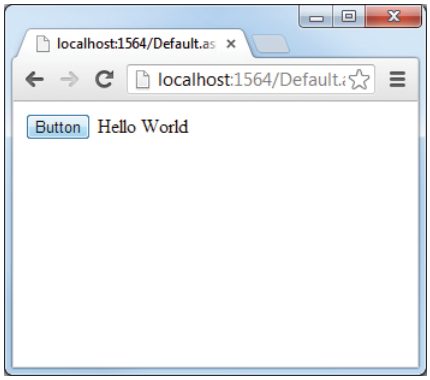

Bild 9.4
Hello World-Nachricht

ÜBUNG: Versuchen Sie nun, die Eigenschaften der Steuerelemente zu verändern, die wir bis jetzt noch nicht verändert haben. Bauen Sie mehrere Buttons und Labels ein und versuchen Sie, verschiedene Eigenschaften im Code-behind zu verändern, je nachdem auf welchen Button man klickt.