5 Internet Explorer WebControls

Wie wir bereits gehört haben, kann man ASP.NET durch WebControls von Drittanbietern leicht und einfach erweitern. Zurzeit bietet Microsoft solche Erweiterungen von WebControls auf ihrer Homepage an. Sie werden unter dem Titel *Internet Explorer WebControls* vertrieben.

Die Internet Explorer WebControls sind eine Sammlung von ASP.NET Server-Controls, die das Oberflächendesign von Webanwendungen vereinfachen. Wenn man mit diesen Server-Controls und Objekten programmiert, werden Seiten erzeugt, die auf allen üblichen Browsern korrekt dargestellt werden. Die WebControls ermitteln automatisch den Browser und passen sich an die Zielumgebung an. Die größte Funktionalität erhält man unter dem Internet Explorer 5.5 und höher. Die Controls beinhalten Funktionen, die clientseitig in DHTML implementiert wurden.

Die folgende Tabelle zeigt die Internet Explorer WebControls, die in der aktuellen Version vorhanden sind.

WebControl	**Beschreibung**
MultiPage	Mit dem MultiPage- und dem TabStrip-WebControl kann man Registerkarten auf einer ASP.NET Seite implementieren, die keine Wünsche offen lassen.
TabStrip	Mit dem MultiPage-und dem TabStrip-WebControl kann man Registerkarten auf einer ASP.NET-Seite implementieren, die keine Wünsche offen lassen.
ToolBar	Mit dem ToolBar-WebControl kann man auf einer ASP.NET-Seite eine Toolbar implementieren, so wie man es von Windows-Anwendung gewohnt ist.
TreeView	Mit dem *TreeView*-WebControl kann man auf einer ASP.NET-Seite einen *TreeView* implementieren, der keine Wünsche mehr offen lässt.

Tabelle 5.1: Internet Explorer WebControls

5.1 Programmieren mit den WebControls

Wenn man mit den Internet Explorer WebControls ASP.NET-Seiten erstellen möchte, ist es notwendig, dass man einige Page-Direktiven verwendet. Diese Direktiven sind notwendig, so dass man auf die WebControls und auf die Objekte im serverseitigen Code zugreifen kann. Insgesamt sind zwei Direktiven notwendig, so dass die Internet Explorer WebControls auf der ASP.NET-Seite für die Programmierung zur Verfügung stehen.

Die Import-Direktive

Mit der *Import*-Direktive muss man den Namespace, in dem die WebControls definiert sind, auf die ASP.NET-Seite importieren. Die Internet Explorer WebControls befinden sich im Namespace *Microsoft.Web.UI.WebControls*. Daher sieht die *Import*-Direktive wie folgt aus:

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
```

Diese Zeile importiert den Namespace *Microsoft.Web.UI.WebControls*, der in der Assembly *Microsoft.Web.UI.WebControls.dll* definiert ist. Dieses Assembly ist ein *Custom Control*, das die WebControls implementiert.

Wenn diese Anweisung auf der ASP.NET-Seite nicht vorhanden ist, ist es auch nicht möglich, dass man die Internet Explorer WebControls verwenden kann.

Die Register-Direktive

Die *Register*-Direktive kann verwendet werden, damit man einer Assembly und deren Namespace einen benutzerdefinierten Namen zuweisen kann. Über diesen benutzerdefinierten Namen können dann die Assembly und der darin definierte Namespace angesprochen werden. Da die meisten Namespaces sehr lange Namen haben, wird ihnen über die *Register*-Direktive ein kurzer Name zugewiesen, über den sie angesprochen werden können. Die *Register*-Direktive für die Internet Explorer WebControls könnte z. B. wie folgt aussehen:

```
<%@
   Register TagPrefix="MyCtrl"
   Namespace="Microsoft.Web.UI.WebControls"
   Assembly="Microsoft.Web.UI.WebControls"
%>
```

Mit dieser Anwendung bekommt der Namespace *Microsoft.Web.UI.WebControls* den benutzerdefinierten Namen *MyCtrl* zugewiesen. Die Server-Controls, die in diesem Namespace implementiert wurden, könnten dann wie folgt angesprochen werden.

```
<MyCtrl:TreeView>...</MyCtrl:TreeView>
```

oder

```
<MyCtrl:TabStrip>…</MyCtrl:TabStrip>
```

Nachdem die *Import*-Direktive und die *Register*-Direktive in der ASP.NET-Seite hinzugefügt wurden, kann man mit den Server-Controls programmieren und diese im serverseitigen Code ansprechen.

Das Runat-Attribut

Um dem Webserver klar zu machen, dass es sich bei den Internet Explorer WebControls um serverseitige Controls handelt, muss man bei jeder Definition eines Controls auf der ASP.NET das Attribut *runat="server"* hinzufügen. Dadurch wissen der Webserver und die ASP.NET-Ausführungsschicht, dass es sich bei dem definierten Element um ein serverseitiges Control handelt. Außerdem müssen die WebControl wie auch alle anderen Controls innerhalb eines *<form runat="server"*>-Attributs definiert werden. Das nächste Bespiel zeigt, wie man einen einfachen *TreeView* auf einer ASP.NET-Seite erstellen könnte.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyCtrl"
    Namespace="Microsoft.Web.UI.WebControls"
    Assembly="Microsoft.Web.UI.WebControls" %>

<html>
<head>
</head>

<body>
   <form runat="server">
      <MyCtrl:TreeView runat="server">
         <MyCtrl:TreeNode Text="Eintrag 1" />
         <MyCtrl:TreeNode Text="Eintrag 2" />
      </MyCtrl:TreeView>
   </form>
</body>
</html>
```

Listing 5.1: Verwendung des TreeView-Controls

Das AutoPostback Attribut

Über das *AutoPostback*-Attribut kann gesteuert werden, wie der Client Aktionen des Benutzers an den Server übermittelt. Wenn das Attribut auf *true* gesetzt ist, wird für jede Aktion des Benutzers ein *Server-Roundtrip* ausgelöst, der das Ereignis dem Server meldet. Dadurch kann der Server die entsprechende Ereignisbehandlungsfunktion aufrufen, und die ASP.NET-Seite für den Benutzer aktualisieren.

Wenn das *AutoPostback*-Attribut auf *False* gesetzt ist, werden keine Ereignisse des Benutzers an den Server gemeldet. Dadurch ist es möglich, dass man Ereignisse clientseitig abfängt und diese dann durch DHTML an den Server meldet, der dann wieder entsprechend reagieren muss.

Als eine generelle Regel ist zu sagen, dass man in *Page_Load* immer überprüfen sollte, ob die Seite durch einen Server-Roundtrip oder im Zuge des Erstaufrufs angezeigt wird. Dies kann man durch das Property *IsPostback* der Page-Klasse überprüfen. Die Internet Explorer WebControls und auch alle anderen Server-Controls müssen zur im Zuge des Erstaufrufs der ASP.NET-Seite initialisiert werden, da durch das *State-Management* von ASP.NET der Status der Controls von Server-Roundtrip zu Server-Roundtrip mitgeführt wird. Dies würde im Code wie folgt aussehen.

```
<script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e)
   {
      if (Page.IsPostBack)
      {
         InitializeControls();
      }
   }
</script>
```

Im folgenden Beispiel möchte ich eine ASP.NET-Seite zeigen, die ein *TreeView*-Control implementiert. Dieses *TreeView* zeigt mehrere Länder mit ihren Hauptstädten an.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyTree"'
    Namespace="Microsoft.Web.UI.WebControls"
    Assembly="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
</head>

<body>
   <form runat="server">
    <h3><font face="Verdana">Verwendung des TreeView Internet Explorer
WebControls</font></h3>
```

```
        <MyTree:TreeView runat="server">
           <MyTree:TreeNode Text="Österreich">
              <MyTree:TreeNode Text="Wien" />
           <MyTree:TreeNode>
           <MyTree:TreeNode Text="Deutschland">
              <MyTree:TreeNode Text="Berlin" />
           </MyTree:TreeNode>
           <MyTree:TreeNode Text="Frankreich">
              <MyTree:TreeNode Text="Paris" />
           </MyTree:TreeNode>
        </MyTree:TreeView>
     </form>
  </body>
  </html>
```

Listing 5.2: Verwendung des TreeView-Controls

Das Beispiel erzeugt die folgende Ausgabe:

Abbildung 5.1: Verwendung des TreeView-Controls

5.2 Das MultiPage-WebControl

Mit dem *MultiPage*-WebControl kann man innerhalb einer ASP.NET-Seite mehrere Seiten anzeigen. Das *MultiPage*-WebControl erlaubt es dem Programmierer, mehrere *PageView*-Elemente zu definieren, die anschließend auf der Seite angezeigt werden. Außerdem wird durch das Steuerelement ein client- und ein serverseitiges Programmiermodell offengelegt, durch das das Control programmiert werden kann.

Das *Multipage*-WebControl wird durch eine Kombination der folgenden Elemente programmiert:

- *MultiPage*: Definiert einen Container für die PageView-Elemente.
- *PageView*: Erzeugt ein PageView-Element innerhalb des MultiPage-Elementes.

Das folgende Beispiel zeigt, wie man das MultiPage-WebControl in einer realen Webanwendung verwenden könnte. Dabei wird ein Assistent erstellt, der vom Benutzer mehrere Informationen über eine fiktive Bestellung einholt. Der Assistent besteht insgesamt aus vier Schritten, die über zwei Buttons am unteren Rand erreichbar sind.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyCtrl"
    Namespace="Microsoft.Web.UI.WebControls
    Assembly="Microsoft.Web.UI.WebControls"
%>
<html>
<head>
<script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e)
   {
      if (!Page.IsPostBack)
      {
         SetupButtons();
      }
   }

   void SetupButtons()
   {
      if (Wizard.SelectedIndex == 0)
      {
         BackBtn.Visible = false;
         NextBtn.Text = "Start >";
      }
      else if (Wizard.SelectedIndex == 1)
      {
         BackBtn.Visible = true;
         NextBtn.Text = "Vorwärts >";
      }
      else if (Wizard.SelectedIndex == (Wizard.Controls.Count - 3))
      {
         NextBtn.Text = "Vorwärts >";
      }
      else if (Wizard.SelectedIndex == (Wizard.Controls.Count - 2))
      {
         NextBtn.Text = "Schluss";
      }
      else if (Wizard.SelectedIndex == (Wizard.Controls.Count - 1))
      {
```

```
            BackBtn.Visible = false;
            NextBtn.Visible = false;
         }
      }

      void BackClick(Object sender, EventArgs e)
      {
         if (Wizard.SelectedIndex > 0)
         {
            Wizard.SelectedIndex--;
            SetupButtons();
         }
      }

      void NextClick(Object sender, EventArgs e)
      {
         if (Wizard.SelectedIndex < (Wizard.Controls.Count - 1))
         {
            Wizard.SelectedIndex++;
            SetupButtons();

            if (Wizard.SelectedIndex == (Wizard.Controls.Count - 1))
            {
               CreateReceipt();
            }
         }
      }

      void CreateReceipt()
      {
     String html = "<p>Auftragsnummer: <b>123456789</b><p>Empfänger:<br>";
       html += "<pre style=\'font-size:small\'>" + FirstName.Text + " " +
LastName.Text + "\n" + Address.Text + "</pre>";
         html += "<p>Rechnungsadresse:<br>";

         if (BillShipping.Checked == true)
         {
            html += "<i>(Gleich wie Empfangsadresse)</i>";
         }
         else
         {
       html += "<pre style=\'font-size:small\'>" + BillFirstName.Text + " " +
BillLastName.Text + "\n" + BillAddress.Text + "</pre>";
         }

         ReceiptPane.InnerHtml = html;
      }
</script>
</head>

<body>
```

```
   <form runat="server">
      <h3><font face="Verdana">Internet-Bestellung</font></h3>
      <MyCtrl:MultiPage id="Wizard" runat="server" BorderWidth="1"
BorderStyle="Solid">
         <MyCtrl:PageView id="Cart">
            <h2>Einkaufswagen</h2>
       <p>Das ist ein Beispiels-Assistent, der das MultiPage WebControl
demonstriert.</p>

            <p>Web-Anwendungen mit Visual C#
            <br>ATS 950,00
            <br>Menge: 1
            </p>
         </MyCtrl:PageView>

         <MyCtrl:PageView id="Shipping">
            <p>Bitte geben Sie die Versandinformationen ein:
            <table>
               <tr>
                  <td>Vorname:</td>
                 <td><asp:TextBox id="FirstName" runat="server" /></td>
               </tr>
               <tr>
                  <td>Nachname:</td>
                  <td><asp:TextBox id="LastName" runat="server" /></td>
               </tr>
               <tr>
                  <td>Adresse:</td>
                  <td><asp:TextBox id="Address" runat="server"
TextMode="MultiLine" Rows="3" /></td>
               </tr>
            </table>
         </MyCtrl:PageView>

         <MyCtrl:PageView id="Billing>
            <h2>Rechnung</h2>
            <p>Bitte geben Sie die Rechnungsinformationen ein:
            <table>
               <tr>
                  <td>Kreditkarte:</td>
                <td><asp:TextBox id="CreditCart" runat="server" /></td>
               </tr>
               <tr>
                  <td>Rechnungsadresse:</td>
                 <td><asp:CheckBox id="BillShipping" runat="server"
Checked="True" /></td>
               </tr>
               <tr>
                  <td>Vorname:</td>
              <td><asp:TextBox id="BillFirstName" runat="server" /></td>
               </tr>
```

```
                <tr>
                   <td>Nachname:</td>
               <td><asp:TextBox id="BillLastName" runat="server" /></td>
                </tr>
                <tr>
                   <td>Adresse:</td>
                   <td><asp:TextBox id="BillAddress" runat="server"
 TextMode="MultiLine" Rows="3" /></td>
                </tr>
             </table>
          </MyCtrl:PageView>

          <MyCtrl:PageView id="Receipt">
             <h2>Bestätigung</h2>
             <p>Web-Anwendungen mit Visual C#
             <br>ATS 950,00
             <br>Menge: 1
             <br>
             <br>Versandkosten: ATS 50,00
             <br>Gesamt: ATS 1000,00
             <br>
             </p>
             <div id="ReceiptPane" runat="server"></div>
          </MyCtrl:PageView>
       </MyCtrl:MultiPage>

 <asp:Buttonid="BackBtn"runat="server"Text="<Zurück"OnClick="BackClick"
 />
       <asp:Button id="NextBtn" runat="server" Text="Vorwärts >"
 OnClick="NextClick" />
    </form>
 </body>
 </html>
```

Listing 5.3: Verwendung des MultiPage-WebControls

Das Beispiel erzeugt die in Abbildung 5.2 gezeigte Ausgabe.

Ich möchte jetzt nochmals den Quelltext Schritt für Schritt durchgehen, damit man sieht, wie das *MultiPage*-WebControl funktioniert. In der Funktion *SetupButtons* werden die beiden Schaltflächen für die Navigation initialisiert, je nachdem auf welcher Seite man sich befindet.

Die Ereignisbehandlungsfunktion *BackClick* wird aufgerufen, wenn der Button mit der Aufschrift < *Zurück* geklickt wird. In ihr wird der *SelectedIndex* des MultiPage-WebControls um eins vermindert, was bedeutet, dass die vorherige Seite angezeigt wird.

Abbildung 5.2: Verwendung des MultiPage-WebControls

```
void BackClick(Object sender, EventArgs e)
{
   if (Wizard.SelectedIndex > 0)
   {
      Wizard.SelectedIndex--;
      SetupButtons();
   }
}
```

Die Ereignisbehandlungsfunktion *NextClick* macht das gleiche wie ihr Vorgänger, nur dass die nächste Seite des *MultiPage*-WebControls angezeigt wird.

Die benutzerdefinierte Funktion *CreateReceipt* wird aufgerufen, wenn der letzte Schritt des Assistenten angezeigt wird. In dieser Funktion wird dann die Rechnung für die Bestellung erstellt und angezeigt. In dieser Funktion sieht man, wie man auf alle Steuerelemente innerhalb aller *PageViews* zugreifen kann.

Im Html-Teil der ASP.NET-Seite wird als erster Schritt das *MultiPage*-Control definiert.

```
<MyCtrl:MultiPage id="Wizard" runat="server" BorderWidth="1" BorderStyle="Solid">
</MyCtrl:MultiPage>
```

Nachdem das *MultiPage*-WebControl definiert ist, werden die verschiedenen *PageView*-Steuerelemente mit ihrem Inhalt definiert, wobei jede *PageView* einen Schritt des Assistenten darstellt.

```
<MyCtrl:PageView id="Cart">
...
</MyCtrl:PageView>
```

Das nächste Beispiel zeigt, wie man dynamisch im serverseitigen Code *PageView*-Steuerelemente zu einem *MultiPage-Control* hinzufügen kann.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyCtrl"
    Namespace="Microsoft.Web.UI.WebControls"
    Assembly="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
<script language="C#" runat="server">
   void cmdAdd_Click(Object sender, EventArgs e)
   {
      PageView myView = new PageView();
      MyMultiPage.Controls.Add(myView);

      myView.Controls.Add(new LiteralControl(txtPageView.Text));
   }
</script>
</head>

<body>
   <form runat="server">
  <h3><font face="Verdana">Dynamische PageView-WebControls</font></h3>
     Bitte geben Sie den Namen des neuen PageView-WebControls ein:<br>
      <asp:TextBox id="txtPageView" runat="server" />
    <asp:Button id="cmdAdd" Text="Hinzufügen" OnClick="cmdAdd_Click"
runat="server" />

      <MyCtrl:MultiPage id="MyMultiPage" runat="server" />
   </form>
</body>
</html>
```

Listing 5.4: Dynamische Erzeugung von PageView-Steuerelementen

5.3 Das TabStrip WebControl

Mit dem *TabStrip* WebControl ist es möglich, leistungsfähige Registerkarten auf einer ASP.NET-Seite anzuzeigen. Meistens wird das TabStrip-WebControl in Verbindung mit dem *MultiPage-Control* verwendet, wobei das *TabStrip*-Steuerelement die Registerkarten definiert und das MultiPage-Steuerelement den Inhalt der verschiedenen Registerkarten definiert und dementsprechend anzeigt.

Es gibt die folgenden *TabStrip*-Objekte, mit denen im serverseitigen Code programmiert werden kann:

Objekt	Beschreibung
TabStrip	Dient als Container für alle anderen Objekte.
Tab	Definiert eine Registerkarte innerhalb des TabStrip-Containers.
TabSeparator	Definiert ein Trennelement, dass zwischen zwei Tab-Elementen angezeigt wird.

Tabelle 5.2: Die verschiedenen TabStrip-Objekte

Features

Das *TabStrip*-WebControl führt eine Vielzahl von Features ein, die in der folgenden Tabelle näher beschrieben werden.

Feature	Beschreibung
Textbasierte Registerkarten	Erstellt Registerkarten, die einen Text enthalten.
Bildbasierte Registerkarten	Erstellt Registerkarten, die Bilder enthalten.
Ausrichtung	Die Registerkarten können entweder horizontal oder vertikal ausgerichtet werden.

Tabelle 5.3: Die verschiedenen Features des TabStrip-Controls

Wie auch all den anderen WebControls, gibt es eine Möglichkeit, dass man dem Steuerelement verschiedene Stile zuweisen kann. Dabei unterstützt das *TabStrip*-Control zur Zeit die folgenden Stile:

Stil	Beschreibung
Default	Definiert das Aussehen, wenn die Registerkarte nicht ausgewählt ist und wenn sich die Maus nicht über der Registerkarte befindet.
Selected	Definiert das Aussehen, wenn die Registerkarte durch den Benutzer ausgewählt wurde.
Hover	Definiert das Aussehen, wenn der Benutzer die Maus über die Registerkarte bewegt.

Tabelle 5.4: Die verschiedenen Stile des TabStrip-Controls

Das Aussehen der Registerkarten kann man bequem über CSS-Eigenschaften steuern. Das folgende Codefragment verwendet verschiedene CSS-Eigenschaften für die Definition des *TabStrip*-WebControls.

```
   <MyTab:TabStrip id="MyTabStrip" runat="server"
TabDefaultStyle="background-color:#000000;font-family:verdana;font-weight:bold"
      TabHoverStyle="background-color:#777777"
      TabSelectedStyle="background-color:#ffffff;color:#000000">

      <MyTab:Tab Text="Posteingang" />
      <MyTab:Separator />
      <MyTab:Tab Text="Postausgang" />
      <MyTab:Separator />
      <MyTab:Tab Text="Addressen" />
   </MyTab:TabStrip
```

Natürlich ist es auch möglich, dass man die verschiedenen Stile im serverseitigen Code ansprechen und somit ändern kann. Der folgende Code demonstriert es.

```
   <script language="C#" runat="server">
      void Page_Load(Object sender, EventArgs e)
      {
         MyTabStrip.BorderColor = Color.Blue;
         MyTabStrip.DefaultStyle.Add("background", "green");
      }
   </script>
```

Anstatt dass man das Aussehen mithilfe von CSS-Eigenschaften ändert, kann man den Registerkarten auch Grafiken zuweisen, mit denen man das Aussehen steuern kann. Im folgenden Code wird das Aussehen der Registerkarten mithilfe von Grafiken festgelegt.

```
   <MyTab:TabStrip id="MyTabStrip" runat="server"
   SepDefaultImageUrl="sep_default.gif" SepSelectedImageUrl="sep_selected.gif">
<MyTab:TabSeparator DefaultImageUrl="image1.gif" SelectedImageUrl="Image2.gif"
   />
      <MyTab:Tab SelectedImageUrl="Posteingang_selected.gif"
   DefaultImageUrl="Posteingang.gif" />
      <MyTab:TabSeparator />
      <MyTab:Tab SelectedImageUrl="Postausgang_selected.gif"
   DefaultImageUrl="Postausgang.gif" />
      <MyTab:TabSeparator />
      <MyTab:Tab SelectedImageUrl="Adressen_selected.gif"
   DefaultImageUrl="Adressen.gif" />
   </MyTab:TabStrip>
```

Das folgende Beispiel zeigt, wie man eine E-Mail-Anwendung in ASP.NET erstellen kann. Dabei wird das *MultiPage*-Control in Verbindung mit dem *TabStrip*-Control verwendet. Insgesamt werden zwei Registerkarten mit den Aufschriften *Posteingang* und *Postausgang* angelegt. Der Inhalt dieser Registerkarten wird über die *MultiPage*-Controls definiert. Dabei steht der Inhalt des ersten *MultiPage*-Controls für die erste Registerkarte, der Inhalt des zweiten *MultiPage*-Controls für die zweite Registerkarte usw.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyTab"
    Namespace="Microsoft.Web.UI.WebControls"
    Assembly="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
</head>

<body>
   <form runat="server">
  <MyTab:TabStrip id="tsHoriz" runat="server" Style="font-weight:bold"
 TabDefaultStyle="Border:solid1pxblack;background:lightgreen;padding-
left:5px;padding-right:5px;"
      TabHoverStyle="color:red"
      TabSelectedStyle="border:solid 1px black;border-
bottom:none;background:white;padding-left:5px;padding-right:5px"
      SepDefaultStyle="border-bottom:solid 1px #000000;"
      TargetID="mpHoriz">

         <MyTab:Tab Text="Posteingang" />
         <MyTab:TabSeparator />
         <MyTab:Tab Text="Postausgang" />
         <MyTab:TabSeparator DefaultStyle="width:100%" />
      </MyTab:TabStrip>

  <MyTab:MultiPage id="mpHoriz" runat="server" Style="border:solid 1px
#000000;border-top:none;padding:5px;height:330">
         <MyTab:PageView>
     <table style="width:95%;font-size:x-small;backbround:lightgreen"
BorderColor="#ffffff" border="1" CellSpacing="0" CellPadding="1">
               <tr style="font-weight:bold">
               <td> </td>
               <td>Von</td>
               <td>Betreff</td>
               <td>Datum</td>
               </tr>

               <tr>
               <td><asp:CheckBox runat="server" /></td>
               <td>Microsoft Corp.</td>
               <td>Einführung in Microsoft .NET</td>
```

```
                <td>Sonntag, 31. Dezember 2001, 23:45</td>
                </tr>

                <tr>
                <td><asp:CheckBox runat="server" /></td>
                <td>Aschenbrenner Klaus</td>
                <td>Web-Anwendungen mit Visual C#</td>
                <td>Montag, 01. Januar 2002, 14:23</td>
                </tr>

                <tr>
                <td><asp:CheckBox runat="server" /></td>
                <td>DataDialog.NET</td>
                <td>Kostenangebot für Softwareprojekt</td>
                <td>Mittwoch, 3. Januar 2002, 10:56</td>
                </tr>
             </table>
          </MyTab:PageView>

          <MyTab:PageView>
     <table style="width:70%;font-size:x-small;background:lightgreen"
BorderColor="#ffffff" Border="1" CellSpacing="0" CellPadding="1">
                <tr>
                <td>Empfänger:</td>
       <td><asp:TextBox id="textRecipient" runat="server" /><asp:Button
id="cmdSearchRecipient" Text="..." runat="server" /></td>
                </tr>

                <tr>
                <td>CC:</td>
             <td><asp:TextBox id="txtCC" runat="server" /><asp:Button
id="cmdSearchCC" Text="..." runat="server" /></td>
                </tr>

                <tr>
                <td>BC:</td>
             <td><asp:TextBox id="txtBC" runat="server" /><asp:Button
id="cmdSearchBC" Text="..." runat="server" /></td>
                </tr>

                <tr>
          <td style="background-color:#ffffff" colspan="2"> </td>
                </tr>

                <tr>
     <td colspan="2"><asp:TextBox id="txtMessage" TextMode="MultiLine"
Cols="82" Rows="10" runat="server" /></td>A
                </tr>
             </table>
          </MyTab:PageView>
       </MyTab:MultiPage>
```

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

Listing 5.5: Verwendung des TabStrip-Controls in Kombination mit dem MultiPage-Control

Das Beispiel erzeugt die folgende Ausgabe:

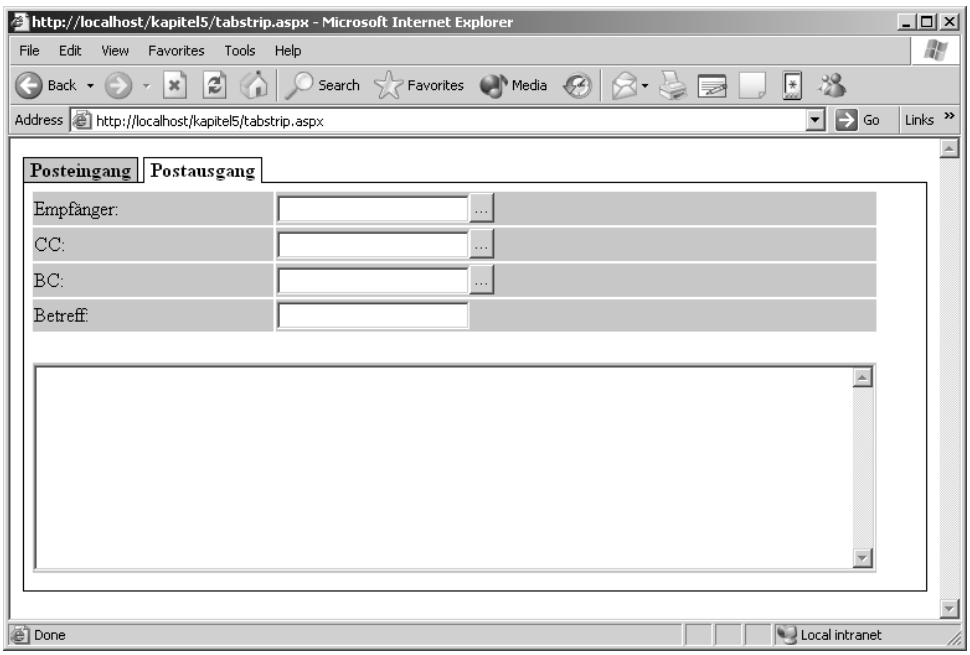

Abbildung 5.3: Verwendung des TabStrip-Controls in Kombination mit dem MultiPage-Control

5.4 Das Toolbar-WebControl

Die Internet Explorer *Toolbar*-WebControls implementieren Toolbars, so wie wir sie von den Windows-Anwendungen her gewohnt sind. Das *Toolbar*-WebControl kann überall auf der Oberfläche hinzugezogen werden und überall angedockt werden. Wenn die vertikale Ausrichtung des Controls während des Ziehens verändert wurde, passt es sich automatisch den neuen Umständen an.

Eine Toolbar kann mit der Kombination aus den folgenden Objekten erstellt werden:

Objekt	Beschreibung
Toolbar	Dient als Container für die anderen Toolbar-Objekte
ToolbarButton	Definiert einen Toolbar-Button
ToolbarCheckButton	Definiert einen Toolbar-CheckButton
ToolbarCheckGroup	Definiert einen Toolbar-CheckGroup
ToolbarDropDownList	Definiert ein DropDown-Listenelement
ToolbarLabel	Definiert einen Label auf einer Toolbar
ToolbarSeparator	Definiert ein Teilungselement auf einer Toolbar
ToolbarTextBox	Definiert eine TextBox auf einer Toolbar

Tabelle 5.5: Objekte einer Toolbar

Das *Toolbar-Element* selbst dient als Container für alle anderen Elemente. Das Toolbar-WebControl hat drei Stile, die in der folgenden Tabellen näher erklärt werden:

Stil	Beschreibung
Default	Das ist der standardmäßige Stil, der verwendet wird, wenn die Toolbar nicht ausgewählt ist und wenn sich der Mauscursor nicht über der Toolbar befindet.
Selected	Dieser Stil wird angewendet, wenn die Toolbar ausgewählt wird.
Hover	Dieser Stil wird verwendet, wenn sich die Maus über der Toolbar befindet.

Tabelle 5.6: Die verschiedenen Zustände des Toolbar-WebControls

Im nächsten Beispiel wird gezeigt, wie man mithilfe von CSS-Eigenschaften das Layout einer Toolbar anpassen kann.

```
<MyToolbar:Toolbar id="MyToolbar" runat="server"
 BorderWidth="5" BorderColor="Gray" Font-Name="Tahoma" Font-Size="8pt"
BackColor="#cccccc" Width="75%"
DefaultStyle="color:black;border:solid3px#cccccc;background:#cccccc"
   HoverStyle="border:solid 3px #ffffff;background:#dddddd"
   SelectedStyle="border:solid 3px #ffffff; background:#eeeeee">

   <MyToolbar:ToolbarButton Text="Mein Button" ImageUrl="mmc.gif" />
   <MyToolbar:ToolbarSeparator />
   <MyToolbar:ToolbarButton Tex="<i>HTML</i> <b>Inhalt</b>" />
   <MyToolbar:ToolbarSeparator />

   <MyToolbar:ToolbarCheckButton Text="CheckButton" />
   <MyToolbar:ToolbarSeparator />

   <MyToolbar:ToolbarCheckGroup>
      <MyToolbar:ToolbarCheckButton Text="CheckButton 1" />
      <MyToolbar:ToolbarCheckButton Text="CheckButton 2" />
```

```
        <MyToolbar:ToolbarCheckButton Text="CheckButton 3" />
    </MyToolbar:ToolbarCheckGroup>
</MyToolbar:Toolbar>
```

Listing 5.6: Verwendung von CSS-Eigenschaften für die Toolbar

Das Beispiel erzeugt die folgende Ausgabe:

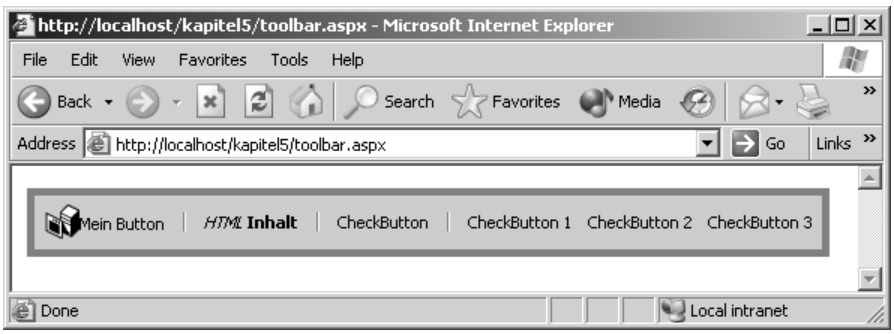

Abbildung 5.4: Verwendung von CSS-Eigenschaften für die Toolbar

Außerdem können auch die Stile einer Toolbar im serverseitigen Code angesprochen werden. Dies wird im folgenden Beispiel demonstriert.

```
<%@ Import Namespace="System.Drawing" %>
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyToolbar"
    Namespace="Microsoft.Web.UI.WebControls"
    Assembly="Microsoft.Web.UI.WebControls"

<html>
<head>
<script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e)
   {
      MyToolbar.BackColor = Color.White;
      MyToolbar.BorderColor = Color.Black;
      MyToolbar.DefaultStyle.Add("background"x, "white");
   }
</script>
</head>

<body>
   <form runat="server">
 <MyToolbar:Toolbarid="MyToolbar"runat="server"AutoPostBack="false">
        <MyToolbar:ToolbarSeparator />
        <MyToolbar:ToolbarTextBox Text="Text" id="txtText" />
        <MyToolbar:ToolbarDropDownList>
```

```
            <asp:ListItem>Text 1</asp:ListItem>
            <asp:ListItem>Text 2</asp:ListItem>
            <asp:ListItem>Text 3</asp:Listitem>
         </MyToolbar:ToolbarDropDownList>
         <MyToolbar:ToolbarCheckGroup>
            <MyToolbar:ToolbarCheckButton Text="Button 1" />
            <MyToolbar:ToolbarCheckButton Text="Button 2" />
         </MyToolbar:ToolbarCheckGroup>
      </MyToolbar:Toolbar>
   </form>
</body>
</html>
```

Listing 5.7: Dynamische Definition von Stilen

Das Beispiel erzeugt die folgende Ausgabe:

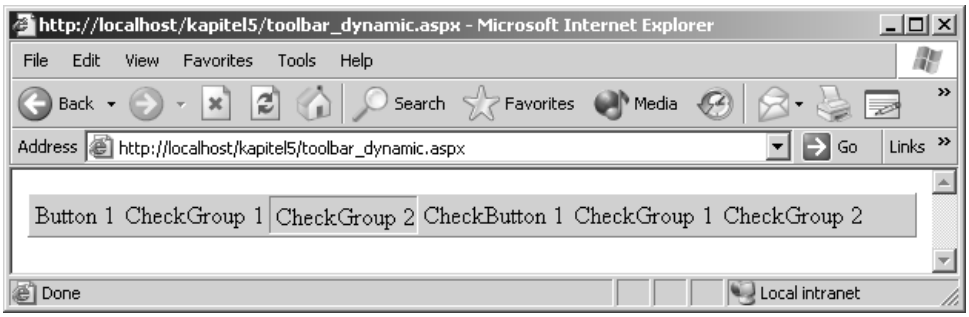

Abbildung 5.5: Dynamische Definition von Stilen

Das folgende Beispiel zeigt, wie man in *Page_Load* eine Toolbar dynamisch erstellen kann.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyToolbar"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
<script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e)
   {
      if (!Page.IsPostBack)
      {
         ToolbarButton tb = new ToolbarButton();
         tb.Text = "Button 1";
```

```
            MyToolbar.Items.Add(tb);
            ToolbarCheckGroup ckg = new ToolbarCheckGroup();
            MyToolbar.Items.Add(ckg);
            ToolbarCheckButton ckb = new ToolbarCheckButton();
            ckb.Text = "CheckButton 1";
            MyToolbar.Items.Add(ckb);

            ckb = new ToolbarCheckButton();
            ckg.Items.Add(ckb);
            ckb.Text = "CheckGroup 1";
            ckb = new ToolbarCheckButton();
            ckg.Items.Add(ckb);
            ckb.Text = "CheckGroup 2";

            ckg = new ToolbarCheckGroup();
            MyToolbar.Items.Add(ckg);
            ckb = new ToolbarCheckButton();
            ckg.Items.Add(ckb);
            ckb.Text = "CheckGroup 1";
            ckb = new ToolbarCheckButton();
            ckg.Items.Add(ckb);
            ckb.Text = "CheckGroup 2";
        }
    }
</script>
</head>

<body>
    <form runat="server">
        <MyToolbar:Toolbar id="MyToolbar" runat="server" />
    </form>
</body>
</html>
```

Listing 5.8: Dynamische Erstellung einer Toolbar

Das Beispiel erzeugt die folgende Ausgabe:

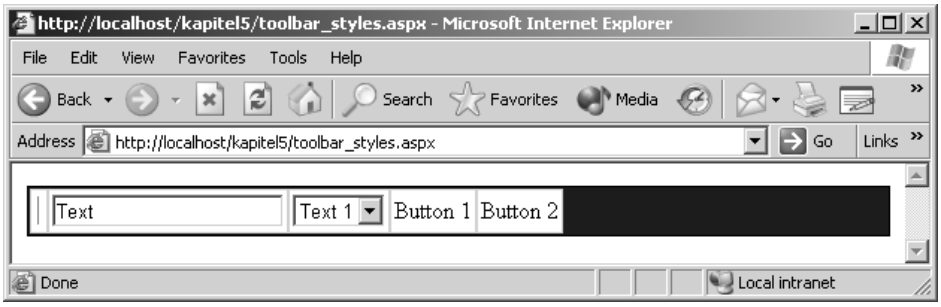

Abbildung 5.6: Dynamische Erstellung einer Toolbar

Im nächsten Beispiel wird gezeigt, wie der serverseitige Code auf Ereignisse reagiert, die der Benutzer clientseitig auslöst.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyToolbar"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebContorls"
%>

<html>
<head>
<script language="C#" runat="server">
   void cmdShow_Click(Object sender, EventArgs e)
   {
      UserInput.InnerHtml = "Sie haben auf das Objekt mit der ID " +
sender.ToString() + " geklickt";
   }
</script>
</head>

<body>
   <form runat="server">
      <MyToolbar:Toolbar id="MyToolbar" runat="server"
OnButtonClick="cmdShow_Click" AutoPostBack="true">
 <MyToolbar:ToolbarCheckButtonid="cmdModeChange"ToolTip="Modusändern"
ImageUrl="help.gif" Text="Kalendermodus" />
         <MyToolbar:ToolbarSeparator />
         <MyToolbar:ToolbarCheckGroup>
            <MyToolbar:ToolbarCheckButton Text="Gruppe A1" />
            <MyToolbar:ToolbarCheckButton Text="Gruppe A2" />
         </MyToolbar:ToolbarCheckGroup />
         <MyToolbar:ToolbarSeparator />
         <MyToolbar:ToolbarCheckGroup>
      <MyToolbar:ToolbarCheckButton Text="Gruppe B1" Selected="True" />
            <MyToolbar:ToolbarCheckButton Text="Gruppe B2" />
            <MyToolbar:ToolbarCheckButton Text="Gruppe B3" />
         </MyToolbar:ToolbarCheckGroup>
      </MyToolbar:Toolbar>

      <p>Benutzerinformationen:<br>
      <div id="UserInput" runat="server" style="border:1px solid
black;background:beige;width:500px;height:50px"></div>
   </form>
</body>
</html>
```

Listing 5.9: Abfangen von clientseitigen Ereignissen

Das Beispiel erzeugt die folgende Ausgabe:

Abbildung 5.7: Abfangen von clientseitigen Ereignissen

Als Abschluss möchte ich im folgenden Beispiel noch zeigen, wie man eine Collection von Toolbar-Objekten durchlaufen kann.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyToolbar"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
<script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e)
   {
      myDiv.InnerHtml = MyToolbar.Controls.Count.ToString() + <br>;

      for (int i=0;i<MyToolbar.Controls.Count;i++)
      {
         myDiv.InnerHtml += MyToolbar.Controls[i].ID + " ";
      }
   }
</script>
</head>

<body>
   <form runat="server">
  <MyToolbar:Toolbar id="MyToolbar" runat="server" AutoPostBack="true">
        <MyToolbar:ToolbarSeparator id="Separator" />
        <MyToolbar:ToolbarButton id="cmdButton1" Text="Button 1" />
        <MyToolbar:ToolbarButton id="cmdButton2" Text="Button 2" />
```

```
            <MyToolbar:ToolbarButton id="cmdButton3" Text="Button 3" />
        </MyToolbar:Toolbar>

        <br>
        <div id="myDiv" runat="server"></div>
    </form>
</body>
</html>
```

Listing 5.10: Ansprechen von Toolbar-Objekten

5.5 Das TreeView-WebControl

Das *TreeView*-WebControl ist ein ASP.NET Server-Control, mit dem man hierarchische Daten auf einer ASP.NET-Seite anzeigen kann. Das *TreeView*-Control geht sogar so weit, dass man eine XML-Datenquelle an das Control binden kann, die definiert, welche Einträge im *TreeView* angezeigt werden.

Das *TreeView*-WebControl kann mit den folgenden Objekten programmiert werden:

Objekt	Beschreibung
TreeView	Definiert einen *TreeView*.
TreeNodeType	Definiert einen Knotentyp, der zu einem oder zu mehreren Knoten gehören kann.
TreeNode	Erzeugt einen Knoten im *TreeView*.

Tabelle 5.7: Die verschiedenen Objekte des TreeView-WebControls

Das nächste Beispiel zeigt, wie ein ganz einfaches *TreeView* erstellt wird.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyTree"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebControls"

<html>
<head>
</head>

<body>
   <form runat="server">
   <h3><font face="Verdana">Ein einfacher TreeView</font></h3>
   <MyTree:TreeView runat="server">
      <MyTree:TreeNode Text="Mein erster Knoten">
         <MyTree:TreeNode Text="Mein zweiter Knoten" />
      <MyTree:TreeNode>
      <MyTree:TreeNode Text="Mein dritter Knoten" />
   </MyTree:TreeView>
```

```
    </form>
  </bod>
  </html>
```

Listing 5.11: Ein einfaches TreeView

Das Beispiel erzeugt die folgende Ausgabe:

Abbildung 5.8: Ein einfaches TreeView

Im Html-Teil der ASP.NET-Seite wird als erster Schritt das *TreeView*-Objekt definiert:

```
<MyTree:TreeView runat="server">
...
</MyTree:TreeView>
```

Nachdem das *TreeView*-Objekt erzeugt wurde, werden die einzelnen Knoten des Controls erstellt. Dabei können die Knoten untereinander verschachtelt werden, damit eine hierarchische Struktur entsteht:

```
<MyTree:TreeNode Text="Mein erster Knoten">
   <MyTree:TreeNode Text="Mein zweiter Knoten" />
</MyTree:TreeNode>
<MyTree:TreeNode Text="Mein dritter Knoten" />
```

Formatierung

Natürlich kann man, wie auch bei den anderen Server-Controls, das *TreeView*-WebControl komplett an seine Bedürfnisse anpassen und dementsprechend formatieren. Über einen *TreeNodeType* kann man das Aussehen der Einträge näher definieren.

Das folgende Beispiel zeigt das Vorgehen näher.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyTree"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
</head>

<body>
   <form runat="server">
   <h3><font face="Verdana">Verwendung eines TreeNodeTypes</font></h3>
      <MyTree:TreeView runat="server" ChildType="Folder">
         <MyTree:TreeNodeType Type="Folder" ExpandedImageUrl="images/
folderopen.gif" ImageUrl="images/folder.gif" />
         <MyTree:TreeNode Text="Österreich">
            <MyTree:TreeNode Text="Graz" />
            <MyTree:TreeNode Text="Innsbruck" />
            <MyTree:TreeNode Text="Salzburg" />
         </MyTree:TreeNode>
         <MyTree:TreeNode Text="Deutschland">
            <MyTree:TreeNode Text="Berlin" />
            <MyTree:TreeNode Text="Hamburg" />
            <MyTree:TreeNode Text="München" />
         </MyTree:TreeNode>
      </MyTree:TreeView>
   </form>
</body>
</html>
```

Listing 5.12: Verwendung eines TreeNodeTypes

Das Beispiel erzeugt die in Abbildung 5.9 gezeigte Ausgabe.

Mit der folgenden Anweisung wird ein *TreeNodeType* mit dem Namen *Folder* erzeugt:

```
<MyTree:TreeNodeType Type="Folder" ExpandedImageUrl="images/folderopen.gif"
ImageUrl="images/folder.gif" />
```

Dieser *TreeNodeType* wird dann dem gesamten *TreeView* zugeordnet. Dadurch werden alle Einträge mit der definierten Grafik versehen. Wenn der Eintrag geschlossen ist, wird die *ImageUrl* verwendet, und wenn der Eintrag geöffnet ist, wird die *ExpandedImageUrl* als Grafikdatei verwendet.

```
<MyTree:TreeView runat="server" ChildType="Folder">
...
</MyTree:TreeView>
```

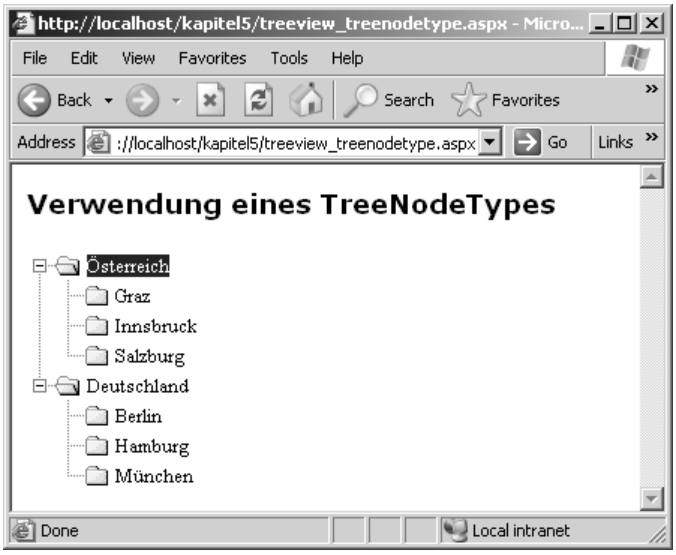

Abbildung 5.9: Verwendung eines TreeNodeTypes

Natürlich gibt es auch die Möglichkeit, dass man einem *TreeView* CSS-Eigenschaften zuweisen kann. Im folgenden Codeauschnitt werden einem *TreeNode* CSS-Eigenschaften zugewiesen.

```
<MyTree:TreeNode Text="Mein Knoten"
DefaultStyle="background:#cccccc;border:1px solid black;font-size:8pt"
HoverStyle="color:blue;font-name:Arial"
SelectedStyle="color:red;font-name:Arial;font-weight:bold-italic" />
```

Mithilfe von Html-Tags können auch die TreeNodes formatiert werden:

```
<MyTree:TreeNode Text="<i>Kursiver Text</i> und <b>fetter Text</b>
```

Datenbindung

Das leistungsfähigste Feature des *TreeView*-Controls ist die Datenbindung. Dadurch kann man einen dynamischen Inhalt mit dem *TreeView* darstellen. Diese Daten müssen als XML-Daten bereitgestellt werden. Im nächsten Beispiel wird die folgende XML-Datei erstelt:

```
<TREENODES>
   <TreeNode Text="Österreich">
      <TreeNode Text="Graz" />
      <TreeNode Text="Innsbruck" />
      <TreeNode Text="Salzburg" />
   </TreeNode>
```

```
    <TreeNode Text="Deutschland">
       <TreeNode Text="Berlin" />
       <TreeNode Text="Hamburg" />
       <TreeNode Text="München" />
    </TreeNode>
</TREENODES>
```

Listing 5.13: city.xml

Bei der Erstellung der XML-Datei ist immer zu beachten, dass das *Root-Element* mit dem Namen <*TREENODES*> immer großgeschrieben wird, da sonst das *TreeView*-Control die XML-Datei nicht richtig interpretieren kann!

In der folgenden ASP.NET-Datei wird anschließend die erstellte XML-Datei mit dem *TreeView*-Control verbunden.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyTree"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebControls"
%>

<html>
<head>
</head>

<body>
   <form runat="server">
<h3><fontface="Verdana">DatenbindungmitdemTreeViewControl</font></h3>
      <MyTree:TreeView runat="server">
         <MyTree:TreeNode Text="Europa" ImageUrl="images/root.gif"
TreeNodeSrc=http://localhost/city.xml />
      </MyTree:TreeView>
   </form>
<body>
</html>
```

Listing 5.14: Datenbindung mit dem TreeView-Control

Das Beispiel erzeugt die in Abbildung 5.10 gezeigte Ausgabe.

Beim *TreeView*-Control gibt es außerdem das Attribut *TreeNodeXslSrc*. Mit diesem kann man eine Xsl-Datei angeben, die die XML-Datei, die bei *TreeNodeSrc* festgelegt wurde, in das richtige Format konvertiert. Im folgenden Code wird z. B. ein *TreeView*-Control erstellt, dass eine Xsl- und eine XML-Datei für die Datenbindung verwendet.

```
<MyTreeCtrl:TreeViewid="MyTree"runat="server"TreeNodeSrc="nodes.xml"
TreeNodeXslSrc="transform.xsl" />
```

Abbildung 5.10: Datenbindung mit dem TreeView-Control

Das nächste Listing zeigt den Inhalt der Datei *nodes.xml*.

```
<?xml version="1.0" ?>
<HelpToc>
   <HelpTocNode Title="Einige meiner Bücher">
      <HelpTocNode Title="Web-Anwendungen mit Visual C#" />
      <HelpTocNode Title="Windows-Anwendungen mit Visual C#" />
   </HelpTocNode>
</HelpToc>
```

Listing 5.15: nodes.xml

Die Datei *transform.xsl* hat folgenden Inhalt:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
   <xsl:template match="/">
      <TREENODES>
         <xsl:for-each select="HelpToc/HelpTocNode">
            <xsl:element name="TreeNode">
         <xsl:attribute name="Text"><xsl:value-of select="@Title"/></
xsl:attribute>

                  <xsl:for-each select="HelpTocNode">
                  <xsl:element name="TreeNode">
         <xsl:attribute name="Text"><xsl:value-of select="@Title"/></
xsl:attribute>
                  </xsl:element>
```

```
                </xsl:for-each>
            </xsl:element>
        </xsl:for-each>
    </TREENODES>
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.16: transform.xsl

Wenn das *TreeView* immer größer und größer wird, wird natürlich auch das Laden der XML-Datei immer länger dauern. Daher gibt es die Möglichkeit, dass man jeden Knoten mit einer XML-Datei versehen kann. Erst wenn der Knoten geöffnet wird, wird auch die XML-Datei geladen. Eine gute Demonstration dieser Vorgehensweise kann man unter *http://msdn.microsoft.com/libary* ansehen, wobei aber zu beachten ist, dass dieses *TreeView* nicht mit dem Internet Explorer *TreeView*-Control implementiert wurde.

Mit dem SQL Server 2000 wurden sehr viele XML-Features eingeführt. Es ist jetzt sogar möglich, dass man Abfragen als XML-Anfragen stellen kann und das Ergebnis ebenfalls als Xml zurückbekommt. Für den SQL Server 7.0 gibt es dafür ein Zusatz-AddOn.

Das nächste Codebeispiel zeigt, wie man den Inhalt eines *TreeView* Controls aus einem SQL Server ermitteln könnte.

```
<MyTree:TreeView Text="Root" Expanded="true" TreeNodeSrc=http://localhost/
MySQLServer?sql=execute+sp_getTreeView />
```

Ereignisse

Wie bei den WebForm-Controls gibt es auch bei den Internet Explorer WebControls die Möglichkeit, dass man auf Ereignisse reagiert, die der Benutzer clientseitig ausgelöst hat. Das *TreeView*-WebControl unterstützt die folgenden Ereignisse:

Ereignis	**Beschreibung**
OnEpand	Wird ausgelöst, wenn ein Knoten geöffnet wird.
OnCollapse	Wird ausgelöst, wenn ein Knoten geschlossen wird.
OnSelectedIndexChanged	Wird ausgelöst, wenn ein Eintrag im *TreeView* Control ausgewählt wird.

Tabelle 5.8: Ereignisse des TreeView-Controls

Das nächste Beispiel zeigt, wie man auf diese Ereignisse reagieren und wie man sie im serverseitigen Code abfangen und behandeln kann.

```
<%@ Import Namespace="Microsoft.Web.UI.WebControls" %>
<%@ Register TagPrefix="MyTree"
    Assembly="Microsoft.Web.UI.WebControls"
    Namespace="Microsoft.Web.UI.WebControls"

<html>
<head>
<script language="C#" runat="server">
   void MyTree_Expand(Object sender, TreeViewClickEventArgs e)
   {
    lblOutput.Text = "Expanded (Node Index = " + e.Node.ToString()+ ")";
   }

   void MyTree_Collapse(Object sender, TreeViewClickEventArgs e)
   {
    lblOutput.Text = "Collapsed (Node Index= " + e.Node.ToString()+ ")";
   }

voidMyTree_SelectedIndexChanged(Objectsender,TreeViewSelectEventArgse)
   {
  lblOutput.Text="Selected:"+e.NewNode.ToString()+"(oldNodeIndex="
+ e.OldNode.ToString() +  ")";
   }
</script>
</head>

<body>
   <form runat="server">
  <h3><font face="Verdana">Ereignisbehandlung mit dem TreeView Control</
font></h3>
  <MyTree:TreeView runat="server" id="MyTree" OnExpand="MyTree_Expand"
OnCollapse="MyTree_Collapse"OnSelectedIndexChanged="MyTree_SelectedIndexChanged"
AutoPostBack="true" TreeNodeSrc="city.xml" />

      <p>
      <asp:Label id="lblOutput" runat="server" />
   </form>
</body>
</html>
```

Listing 5.17: Ereignisbehandlung mit dem TreeView-Control

Das Beispiel erzeugt die folgende Ausgabe:

Abbildung 5.11: Ereignisbehandlung mit dem TreeView Control

5.6 Zusammenfassung

In diesem Kapitel hat man gesehen, wie einfach es ist, ASP.NET mit WebForm-Controls zu erweitern, die von einem Drittanbieter stammen. Zurzeit bietet Microsoft solch eine Erweiterung an, nämlich die *Internet Explorer WebControls*.

Diese Sammlung beinhaltet die folgenden Controls:

- *MultiPage*
- *TabStrip*
- *Toolbar*
- *TreeView*

Das *MultiPage*- und das *TabStrip*-Control werden meistens zusammen verwendet. Dabei definiert das *TabStrip*-Control verschiedene Registerkarten und das Aussehen dieser Registerkarten wird mit dem *MultiPage*-Controls festgelegt. Dabei wird für jede Registerkarte ein eigenes *MultiPage*-Control angelegt.

Sehr interessant ist auch das *Toolbar*-Control, mit dem man leistungsfähige Toolbars erstellen kann, so wie man es von den Windows Anwendungen her kennt.

Das leistungsfähigste Control in der Sammlung ist sicher das *TreeView*-Control, mit dem man mit ein paar Zeilen Code ein umfangreiches *TreeView* erzeugen kann. Außerdem kann das *TreeView*-Control an eine XML-Datenquelle gebunden werden. Diese Quelle kann sogar vom SQL Server stammen.

Im letzten Beispiel dieses Kapitels habe ich gezeigt, wie einfach es ist, aus einer Datenbank ein *TreeView* zu erzeugen. Dabei wurde die XML-Datei für das *TreeView* von einer Komponente erstellt, die die Daten aus der Datenbank ausliest und anschließend in die XML-Datei schreibt.