

ASP.NET 2.0: Master Pages, Navigation, Themes

Schöne neue Webbaublötze

Die erste Beta-Version von .NET 2.0, Codename „Whidbey“, rückt näher. Mit ihrem Erscheinen bietet sich erstmals die Möglichkeit, die neuen Features in einem breiten und öffentlichen Rahmen zu testen. Überproportional viel hat sich bei ASP.NET 2.0 getan. dotnetpro stellt in einer dreiteiligen Serie die Neuerungen der Webentwicklungsumgebung vor.

ASP.NET ist eine Frontend-Technologie, bei der es darum geht, durch Business-Abläufe gewonnene Daten über den Browser zu visualisieren und eine Interaktion mit Benutzern zu ermöglichen. In der Praxis geht es jedoch oft auch nur um das Darstellen von statischen Informationen und sehr überschaubaren interaktiven Elementen. Doch ob komplexe Webapplikation oder

einfache Internetpräsenz, viele Anforderungen sind bei jedem ASP.NET-Projekt wiederzufinden. An vorderster Stelle steht der Wunsch nach einer schnell umsetzbaren und doch ebenso schnell austauschbaren Benutzerschnittstelle.

Mit mehreren neuen Features unterstützt ASP.NET 2.0 den Entwickler bei der Erstellung von Benutzerschnittstellen und schließt dabei offensichtliche Lücken der ersten Version. Einen Schwerpunkt bildet die weitgehende Trennung von Seitenstruktur und Design. Nicht jeder Entwickler ist als Grafik-Ass geboren. Daher muss es möglich sein, die visuelle Gestaltung losgelöst von der Control-Struktur einer Seite zu bearbeiten und jederzeit wieder auszutauschen.

Master Pages

Die Funktionalität von Master Pages ist unter vielen Namen bekannt, unter anderem als Page Templating. Stets ist das Gleiche gemeint, nämlich das zentrale Hinterlegen eines Seiten-Layouts samt einer Navigation, die automatisch auf die einzelnen Inhaltsseiten übertragen wird. Änderungen werden nur noch an einer Stelle vorgenommen und wirken sich sofort auf alle abhängigen Seiten aus.

In den Versionen 1.0 und 1.1 von ASP.NET wurde keine Master-Page-Funktionalität angeboten. Das war Grund genug für zahlreiche Entwickler, um eigene Ansätze zu implementieren. Die Konzepte, die Uwe Baumann in [1] vorgestellt hat, reichten von der statischen Ausgabe der HTML-Tags über die Verwendung von User Controls bis hin zum Aufbohren des Page-Lifecycles. Letzterer Ansatz bot die größtmögliche Flexibilität und wurde auch von mir präferiert, so richtig ideal war aber

Listing 1

Master Pages haben eine @Master-Direktive.

```
<%@ master language="C#" %>

<html>
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form runat="server">
    <table id="Table1" cellspacing="1"
      cellpadding="1" width="100%" border="1">
      <tr>
        <td colspan="2">
          <h1>My Little Company</h1>
        </td>
      </tr>
      <tr>
        <td>Navigation?</td>
      </tr>
      <tr>
        <td>
          <asp:contentplaceholder
            id="ContentPlaceholder1" runat="server">
          </asp:contentplaceholder>
        </td>
      </tr>
      <tr>
        <td>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>
```

auch diese Lösung nicht. Mit Version 2.0 bietet ASP.NET nun erstmals eine eigene, voll integrierte Unterstützung für Master Pages an. Dabei hinterlegt der Entwickler ein oder mehrere Layouts und weist den

Auf einen Blick

Autor



Patrick A. Lorenz ist Microsoft MVP.NET sowie CEO und CTO der PGK Software & Communication GmbH. Die Entwicklung und Betreuung der dotnetpro-Website zählt zu seinem Verantwortungsbereich. Patrick ist Autor der dotnetpro sowie mehrerer Fachbücher zu ASP.NET. Er hat das weltweit erste Buch zu ASP.NET 2.0 geschrieben. Sie erreichen ihn unter www.pgk.de oder unter www.aspnet2.de.

dotnetpro.code
A0404ASPNET20

Sprachen C#

Technik ASP.NET 2.0

Voraussetzungen Visual Studio .NET Whidbey

Serie

1. Master Pages, Navigation, Themes

2. Data Controls

3. Infrastruktur-Komponenten

einzelnen Inhaltsseiten das Layout zu. In Verbindung mit Visual Web Developer können Sie die Inhalte visuell im Rahmen der Master Page gestalten und dabei auch mehrere Platzhalter individuell befüllen.

Um eine neue Master Page zu erstellen, wählen Sie den Befehl *Add New Item* aus dem *Website*-Menü von Visual Studio .NET Whidbey und erstellen ein neues Element vom Typ *MasterPage* mit dem Namen *MasterPage1.master*. Alle Master Pages haben diese spezielle Endung, die unter anderem verhindert, dass die Seite direkt im Browser aufgerufen wird.

Das Ergebnis ist eine neue Seite, die bereits ein Control vom Typ *ContentPlaceHolder* enthält. Es handelt sich um ein reguläres Control. Das heißt, Sie können dieses wie gewohnt in das sonstige HTML-Layout der Seite integrieren, wie es Abbildung 1 zeigt.

Ein Blick auf den Quelltext der Seite in Listing 1 zeigt, dass die Master Page dem Aufbau einer regulären Seite entspricht. Der *ContentPlaceHolder* ist als normales Server Control platziert. Ein Unterschied lässt sich allerdings doch erkennen: Die *@Master*-Direktive ersetzt in diesem Fall die *@Page*-Direktive.

Nachdem Sie die Vorlage erfolgreich erstellt haben, geht es ans Eingemachte. Nun erstellen Sie die eigentliche Seite mit ihrem individuellen Inhalt. Fügen Sie dazu mit dem *Add New Item*-Befehl eine neue Seite vom Typ *Content Page* mit dem Namen *ContentPage1.aspx* ein. Über den folgenden Dialog wählen Sie die eben erstellte Master Page aus. Die neue Seite wird nun in der Source-Ansicht angezeigt. Der Inhalt ist noch recht dürftig und umfasst lediglich eine einzige Zeile. Über die *@Page*-Direktive wird die gewählte Master Page referenziert.

Ein Wechsel auf die Design-Ansicht enthüllt die wahre Leistungsfähigkeit der aktualisierten Entwicklungsumgebung. Die neu erstellte Inhaltsseite wird im Kontext der Master Page angezeigt, die mittels Alpha-Blending in den Hintergrund gesetzt wurde. Das platzierte *ContentPlaceHolder*-Control ist aktiv und kann mit Inhalt gefüllt werden, wie es in Abbildung 2 zu sehen ist.

Statt des im Beispiel gezeigten Textes können Sie selbstverständlich auch andere Controls im Platzhalter hinterlegen. Sie unterliegen in dieser Hinsicht keinerlei Beschränkungen. Auch User Controls und beliebiger server- wie clientseitiger Quelltext lassen sich ohne Änderungen einsetzen.

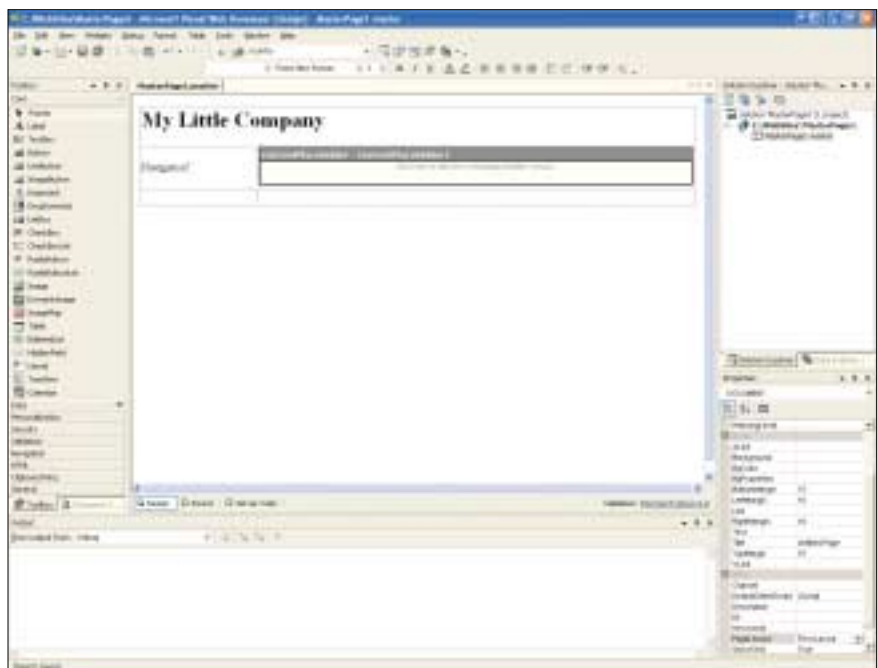


Abbildung 1 Master Pages lassen sich visuell in VS.NET gestalten.

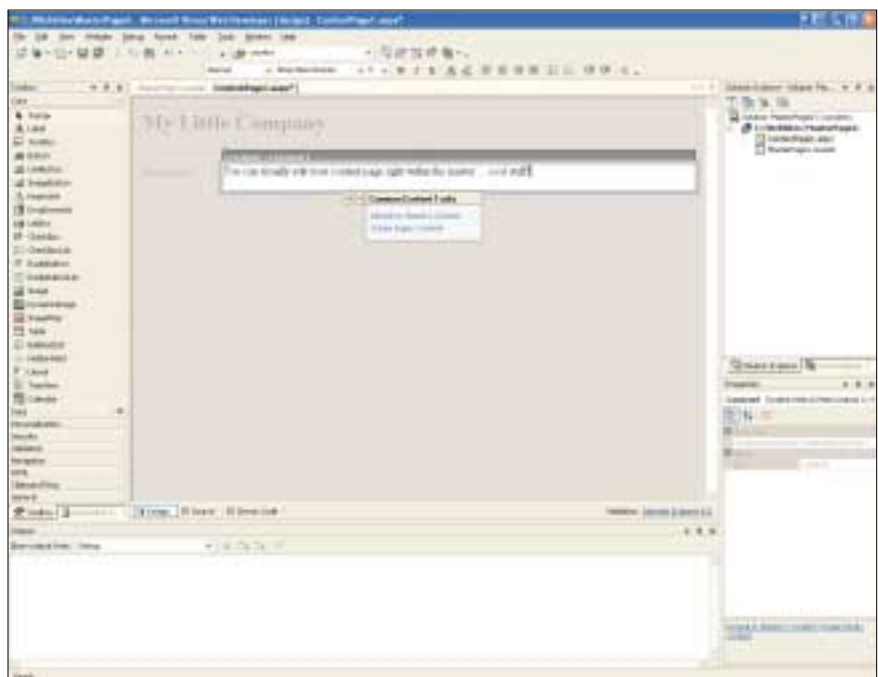


Abbildung 2 Content Pages werden im Kontext der gewählten Master Page bearbeitet.

Ein Blick auf den HTML-Quelltext der Seite verrät, was intern passiert. Der eingegebene Text wird innerhalb eines *Content*-Controls gespeichert, das über die Eigenschaft *ContentPlaceHolderId* mit dem Platzhalter in der Master Page verknüpft wird.

```
<%@ page language="C#"
master="~/MasterPage1.master" %>
<asp:content id="Content1"
```

```
contentplaceholderid="ContentPlaceHolder1"
runat="server">
```

```
You can visually edit your content page right
within the master ... cool stuff!
</asp:content>
```

Beachten Sie, dass Inhalte ausschließlich innerhalb der *Content*-Controls hinterlegt werden dürfen. Außerhalb der durch die Master Page vorgegebenen Regionen können keine Objekte platziert werden. Dies schließt auch Standard-

Listing 2

Eine Content Page kann über ihre Eigenschaft "Master" die Eigenschaften der Master Page beeinflussen.

```
// MasterPage3.master
<%@ master language="C#" %>
<script runat="server">
    public string HtmlTitle {
        get { return this.LT_HtmlTitle.Text; }
        set { this.LT_HtmlTitle.Text = value; }
    }
</script>
<html>
<head runat="server">
    <title><ASP:Literal id="LT_HtmlTitle" runat="server" /></title>
</head>
<body>
    <form runat="server">
        <asp:contentplaceholder id="ContentPlaceholder1" runat="server">
            </asp:contentplaceholder>
        </form>
    </body>
</html>
// ContentPage3.aspx
<%@ page language="C#" master="~/MasterPage3.master" %>
<script runat="server" language="C#">
    void Page_Load(object sender, System.EventArgs e)
    {
        this.Master.HtmlTitle = "Hello World!";
    }
</script>
```

HTML-Tags wie *html*, *head* und *body* ein, da diese bereits in der Master Page hinterlegt werden.

Natürlich können Sie die erste Content Page um eine beliebige Anzahl weiterer Seiten ergänzen. Änderungen an der Master Page wirken sich sofort und automatisch auf alle zugeordneten Inhalte aus. Auch ist es möglich, beliebig viele unterschiedliche Master Pages für unterschiedliche Bereiche Ihrer Website anzulegen.

Statt für jede einzelne Seite individuell eine Master Page über die *@Page*-Direktive zu bestimmen, können Sie dies auch global über ein undokumentiertes Attribut in der *web.config* festlegen:

```
<pages
    master="~/MasterPage.master"
/>
```

Je nach Anforderung an das Projekt kann es interessant sein, mehrere Platzhalter zu definieren, um eine möglichst große Flexibilität bei der Gestaltung der einzelnen Seiten zu gewährleisten. Das System ist in dieser Hinsicht nicht beschränkt, denn Sie können eine beliebige Anzahl an *ContentPlaceHolder*-Controls hinterlegen und diese später mit Inhalt füllen.

Sehr angenehm ist zudem die Möglichkeit, Standardinhalte zu hinterlegen. So können Sie beispielsweise einen Footer-Bereich mit Copyright-Hinweisen füllen. Im Regelfall wird der eingegebene Text direkt übernommen. Sie haben jedoch zudem die Möglichkeit, diesen individuell auf ausgewählten Unterseiten zu ersetzen.

```
<%@ master language="C#" %>
```

```
...
<asp:contentplaceholder id="ContentPlace
Holder1" runat="server">
    Copyright 2004 by John Doe
</asp:contentplaceholder>
```

Als Basis für individuelle Erweiterungen und Eingriffe bietet Ihnen ASP.NET die Möglichkeit, zur Laufzeit programmgesteuert auf die Seitenvorlage zuzugreifen. Hierzu bietet die abgeleitete Content Page eine als *protected* markierte Eigenschaft *Master* an. Hierüber erhalten Sie direkt eine Instanz auf die individuelle Master Page und können auf deren öffentlich angebotenen Eigenschaften und Methoden zugreifen.

Ein typisches Beispiel für den Einsatz ist die Zuweisung eines individuellen Seitentitels. Die beiden Dateien in Listing 2 zeigen dies anhand eines auf der Master Page platzierten *Literal*-Controls. Das Zuweisen eines Titels erfolgt über eine individuelle, öffentliche Eigenschaft *HtmlTitle*, die den Text an das Label weiterreicht. Falls Sie übrigens auch Meta-Tags dynamisch definieren wollen, empfehle ich Ihnen die Verwendung des von der Klasse *Page* angebotenen *Header*-Objekts, das die Schnittstelle *IPageHeader* implementiert und auf diese Weise Zugriff auf Meta-Tags, externe CSS-Dateien sowie die globalen Seiten-Stylesheets bietet. Der nachfolgend gezeigte Ansatz funktioniert gleichermaßen in Master Pages, Content Pages und regulären Seiten:

```
<%@ page language="C#" master="~/
MasterPage4.master" %>
```

```
<script runat="server" language="C#">
void Page_Load(object sender, System.EventArgs e)
{
    this.Master.HtmlTitle = "Hello World!";
    this.Header.Metadata.Add("Author", "Patrick A.
Lorenz");
}
</script>
```

Intern funktionieren Master Pages übrigens recht einfach. Wenn Sie *Trace=True* verwenden, können Sie im Control Tree der Seiten nachvollziehen, dass die Master Page als eigenständiges Control direkt unterhalb der Seite angelegt wird.

Auch die *ContentPlaceHolder* sind dort zu erkennen. Diese enthalten direkt die Objekte, die entweder in der Content Page hinterlegt oder in der Master Page als Standard angegeben wurden. Die Inhalte der auf der Seite platzierten Content Controls werden also übertragen.

Navigation

Wenn das Design-Grundgerüst der Seite steht, dann geht es im zweiten Schritt meist um die Struktur der Website, also um die einzelnen Seiten.

Abgebildet wird die Struktur über die Navigation. ASP.NET 2.0 verfügt über eine umfangreiche Unterstützung zur Implementierung von derartigen Navigationsstrukturen.

Die Basis bildet eine *siteMap*, die alle Elemente, das heißt Seiten, der Navigation in einer Hierarchie zusammenfasst. Diese *siteMap* basiert auf einer XML-Datei *app.sitemap*, die im Root-Verzeichnis

der Applikation hinterlegt wird. Der Aufbau der Datei entspricht einem vorgegebenen Schema:

```
<siteMap>
  <siteMapNode title="" description="" url="">
    <siteMapNode .../>
  </siteMap>
</siteMap>
```

Die einzelnen *siteMapNode*-Elemente in diesem Schema können beliebig verschachtelt werden, woraus sich die Hierarchie der Navigation ergibt. Ein Beispiel für eine solche Hierarchie wird in der Dokumentation mitgeliefert und ist in Listing 3 zu sehen.

Listing 3

Im siteMap-Element lässt sich die Hierarchie der gesamten Website definieren.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode title="Home" description="Home" url="default.aspx" >
    <siteMapNode title="Products" description="Our products"
      url="Products.aspx">
      <siteMapNode title="Hardware" description="Hardware choices"
        url="navigation1.aspx" />
      <siteMapNode title="Software" description="Software choices"
        url="Software.aspx" />
    </siteMapNode>
    <siteMapNode title="Services" description="Services we offer"
      url="Services.aspx">
      <siteMapNode title="Training" description="Training classes"
        url="Training.aspx" />
      <siteMapNode title="Consulting" description="Consulting services"
        url="Consulting.aspx" />
      <siteMapNode title="Support" description="Supports plans"
        url="Support.aspx" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

Unterhalb von *Home* sind hier zwei Bereiche *Products* und *Services* angeordnet, die ihrerseits wieder einige Unterpunkte enthalten. Beachten Sie, dass die hier verwendeten Strukturen nicht zwangsweise der physikalischen entsprechend müssen. Es bietet sich jedoch aus Gründen der Übersicht gerade bei größeren Projekten durchaus an, die Navigation in der Dateistruktur abzubilden.

Neben den hier gezeigten Attributen *title*, *url* und *description* können Sie in *keywords* eine mittels Kommata separierte Liste von Schlüsselwörtern für die jeweilige Seite zuweisen. Wollen Sie den Zugriff auf eine Seite einschränken, können Sie die erlaubten Rollen ebenso durch Kommata getrennt dem Attribut *roles* übergeben. Auch beliebige individuelle Attribute können hinterlegt werden.

Um Ihrer Applikation diese oder eine andere Navigation zuzuweisen, fügen Sie dem Projekt eine neue XML-Datei mit dem oben genannten Namen *app.sitemap* hinzu und kopieren den gezeigten Inhalt hinein. Anschließend steht die Navigation in jeder Seite der Website zur Verfügung.

Zur Visualisierung der Navigationshierarchie ist das neue *TreeView*-Control prädestiniert. Um eine derartige Hierarchie automatisch in alle Seiten einzubinden, empfiehlt sich zudem die Verwendung von Master Pages. So geht's:

Anzeige 1/2
hoch

1. Um eine Navigation mit dem *TreeView*-Control anzulegen, platzieren Sie zunächst ein neues Control an der gewünschten Position innerhalb Ihrer Seite oder Master Page.
2. Wählen Sie anschließend aus der Task-Liste den Befehl *Connect To DataSource ...* und legen Sie eine neue Quelle vom Typ *SiteMapDataSource* an.
3. Setzen Sie die Eigenschaft *InitialExpandDepth* des *TreeView*-Controls auf 2.
4. Starten Sie die Website mit [F5].

Das *SiteMapDataSource*-Control liest die Daten automatisch aus der hinterlegten XML-Datei und stellt sie dem *TreeView* zur Verfügung. Ohne weitere Eingriffe sollte Ihr Browser in etwa wie in Abbildung 3 gezeigt aussehen.

Der für dieses Beispiel benötigte HTML-Quelltext ist ausgesprochen kurz:

```
<asp:treeview id="TreeView1" runat="server"
font-underline="False"
datasourceid="SiteMapDataSource1"
initialexpanddepth="2">
<parentnodestyle font-underline="False">
</parentnodestyle>
<leafnodestyle font-underline="False">
</leafnodestyle>
<nodestyle font-underline="False">
</nodestyle>
<rootnodestyle font-underline="False">
</rootnodestyle>
</asp:treeview>
<asp:sitemapdatasource id="SiteMapDataSource1"
runat="server">
</asp:sitemapdatasource>
```

Eine andere Möglichkeit zur Visualisierung der Navigation zeigt das zweite Beispiel.

Die drei Hauptpunkte werden nun in einer horizontalen Navigationsleiste angezeigt. Hierzu wird ein *DataList*-Control verwendet, das über ein separates *SiteMapDataSource*-Control verfügt. Die *TreeView* ist weiterhin vorhanden, zeigt aber nun lediglich die Navigationspunkte ab der zweiten Ebene an, natürlich in Abhängigkeit von der aktuellen Position der Seite.

```
<asp:datalist id="DataList1" runat="server"
repeatdirection="Horizontal"
datasourceid="SiteMapDataSource2">
<itemtemplate>
<asp:hyperlink
id="HyperLink1"
runat="server"
navigateurl='<%# DataBinder.Eval
(Container.DataItem, "url") %>'
```

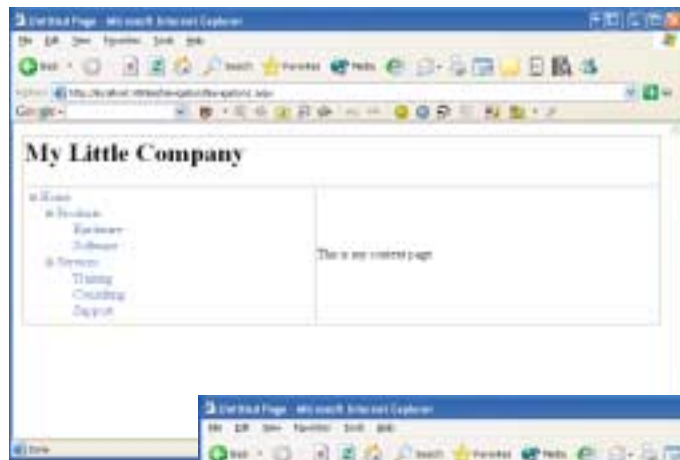


Abbildung 3 Mit dem *TreeView*-Control lässt sich die Navigationshierarchie der Website darstellen.

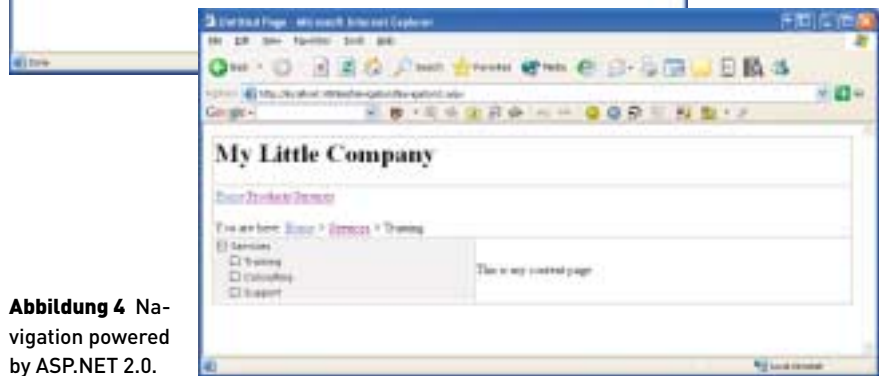


Abbildung 4 Navigation powered by ASP.NET 2.0.

```
text='<%# DataBinder.Eval
(Container.DataItem, "title") %>'>
</asp:hyperlink>
</itemtemplate>
</asp:datalist>
<asp:sitemapdatasource id="SiteMapDataSource2"
runat="server"
sitemapviewtype="Flat"
flatdepth="2">
</asp:sitemapdatasource>
```

Kennen Sie Bread Crumbs, auf Deutsch Brotkrümel? So wie Hänsel und Gretel im Märchen Brotkrümel austreuten, um mit ihrer Hilfe nach Hause zurückzufinden, bezeichnen Bread Crumbs im Zusammenhang mit der Webentwicklung die Anzeige der aktuellen Position in der Navigationshierarchie. ASP.NET 2.0 stellt ein Control zur Verfügung, das speziell für die Anzeige dieser Positionierung gedacht ist. Das Control hört auf den Namen *SiteMapPath* und muss einfach – ohne Angabe einer Datenquelle – auf der Seite platziert werden. Auch hier bietet sich der Einsatz von Master Pages an.

```
<asp:sitemappath id="SiteMapPath1"
runat="server">
</asp:sitemappath>
```

Das Control ist ein Paradebeispiel für weit reichende Anpassbarkeit. Über zahlreiche Eigenschaften können Sie festlegen, wie die Position angezeigt werden soll. So können Sie beispielsweise mittels


PathSeparator das Trennzeichen angeben und über *RenderCurrentNodeAsLink* festlegen, ob auch die aktuelle Seite per Link erreichbar sein soll. Auch können Sie einen *ToolTip* mit der Beschreibung des jeweiligen Navigationselementes anzeigen lassen. Selbstverständlich lassen sich alle Bereiche über Styles auch optisch anpassen. Falls Ihnen diese Einflussmöglichkeiten immer noch nicht ausreichen, können Sie eine Reihe von individuellen Templates hinterlegen.

In Abbildung 4 sehen Sie die drei gezeigten Beispiele im kombinierten Einsatz.

Das ASP.NET-Team plant, mit der endgültigen Version 2.0 ein umfangreiches *DHTML-Menu*-Control auszuliefern, das sich die benötigten Daten ähnlich dem *SiteMapPath*-Control aus dem aktiven *SiteMapProvider* zieht. In der aktuellen Vorabversion ist dieses Control jedoch noch nicht enthalten.

Das *SiteMapPath*-Control und voraussichtlich auch das später verfügbare *Menu*-Control benötigen keine explizite Datenquelle zur Anzeige der Navigation. Die Controls verwenden dazu ein API, das übergreifend zur Verfügung steht.

Die Basis bildet die Klasse *SiteMap* aus dem Namespace *System.Web*, die über vier statische Eigenschaften unter anderem Zugriff auf den aktuellen Knoten (*CurrentNode*) sowie den Wurzelknoten



(*RootNode*) erlaubt. Tatsächlich werden diese Eigenschaften auch vom *SiteMapDataSource*-Control verwendet, um anderen Data Controls den Zugriff auf die Navigationsdaten zu ermöglichen.

In den nachfolgenden Zeilen sehen Sie die Verwendung der Klasse *SiteMap*. Innerhalb der *Page_Load*-Methode der Master Page wird der Titel des aktiven Navigationseintrages (*SiteMapNode*) abgefragt und als HTML-Titel der Seite wiederverwendet.

```
<%@ master language="C#" %>
<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        if (SiteMap.CurrentNode != null) {
            this.LT_HtmlTitle.Text = SiteMap.CurrentNode.Title;
        }
    }
</script>
<html>
<head id="Head1" runat="server">
    <title><ASP:Literal id="LT_HtmlTitle" runat="server"/></title>
</head>
...

```

Wie vielen anderen der neuen Features von ASP.NET liegt auch der Navigation ein Provider-Modell zugrunde. Hierüber können Sie individuelle Datenquellen wie beispielsweise eine Datenbank verwenden, um Ihre Navigation aufzubauen. Nicht immer müssen die Daten also aus der *app.sitemap*-Datei kommen.

Um einen neuen Provider zu erstellen, legen Sie eine Klasse an, die die Schnittstelle *ISiteMapProvider* implementiert. Die Schnittstelle definiert drei Methoden und vier Eigenschaften, die von der Klasse *SiteMap* genutzt werden, um das benötigte Objektmodell aufzubauen. Ein Beispiel für die Implementierung eines individuellen Providers finden Sie in der Dokumentation der Schnittstelle.

Themes

Master Pages liefern das übergreifende Design und die Navigation hält die Website zusammen. Damit wird es Zeit, sich um die Seiten selbst zu kümmern. Um den sich wiederholenden Elementen einer Website wie beispielsweise Überschriften oder Eingabefeldern ein einheitliches Look-and-Feel zu geben, werden in aller Regel Cascading Stylesheets verwendet. Daran soll sich auch in Zukunft nichts ändern, jedoch geht die CSS-Technologie für den Geschmack eines ASP.NET-Entwicklers nicht weit genug. Denken Sie zum Beispiel an die komplexen UI-Eigenschaften und Styles eines Grids oder eines *Calendar*-Controls.

ASP.NET 2.0 erlaubt die Verwendung von so genannten Themes. Diese ermöglichen die übergreifende Definition von Control-Designs, so genannten Skins, zur Vereinheitlichung des Look-and-Feels. Ein Theme beinhaltet für jeden gewünschten Server-Control-Typ ein oder mehrere Skins, die wiederum eine leere Control-Definition mit den gewünschten Eigenschaften-Zuweisungen enthalten. Sie können ein Theme auf eine ganze Website, auf Teile einer Website oder auf eine einzelne Seite anwenden. Bei jeder Definition können Sie entscheiden, ob diese prinzipiell für alle Controls eines Typs oder nur für ausgewählte gelten soll.

Ihren großen Vorteil spielen Themes immer dann aus, wenn eine bestehende Website ergänzt oder modifiziert werden soll. Dann nämlich lassen sich Änderungen zentral vornehmen und wirken sich direkt auf alle entsprechenden Seiten aus. Die einzige Voraussetzung hierfür ist die konsequente Nutzung von Themes.

Anzeige 1/2
hoch

Zu den per Theme beziehungsweise Skin definierbaren Eigenschaften eines Controls gehören neben sehr vielen anderen zum Beispiel auch die Farbwerte für Hintergrund, Vordergrund und Rahmen. Zudem ist es je nach Control möglich, Bilder oder andere Ressourcen individuell zuzuweisen, beispielsweise die Node-Icons des neuen *TreeView*-Controls. Nicht per Theme definierbar sind Eigenschaften, die primär zur Laufzeit gesetzt werden, wie etwa *DataSource*, oder Eigenschaften, die direkt das Verhalten eines Controls beeinflussen.

Eines der mitgelieferten Themes für das *Label*-Control sieht beispielsweise so aus:

```
<asp:Label runat="server"
  ForeColor="#000066"
  BackColor="transparent">
</asp:Label>
```

Es geht allerdings auch wesentlich komplexer wie etwa beim *TreeView*-Control, bei dem wie erwähnt auch die angezeigten Bilder mittels des Themes festgelegt werden. Beide Design-Vorlagen wirken sich, soweit nicht anders angegeben, auf alle Controls des jeweiligen Typs aus.

```
<asp:TreeView runat="server"
  BorderColor="#EEEEEE" ParentNodeImageUrl=
  "images/basicblue_greysquare.gif" RootNode-
  ImageUrl="images/basicblue_greysquare.gif">
  <NodeStyle horizontalpadding="5"
  ForeColor="#000066" verticalpadding="1"
  BackColor="#FFFFFF"></NodeStyle>
  <LeafNodeStyle ForeColor="#000066"
  BackColor="#FFFFFF"></LeafNodeStyle>
  <LevelStyles>
  <asp:TreeNodeStyle Font-Bold="True"
  BackColor="#FFFFFF" ForeColor="#000066"
  ChildNodesPadding="5"></asp:TreeNodeStyle>
  </LevelStyles>
  ...
</asp:TreeView>
```

Auf den ersten Blick haben Themes viel mit Cascading Stylesheets gemein. Anders als CSS sind Themes aber direkt in ASP.NET integriert und auf dessen Konzepte hin optimiert. Themes können daher sehr viele Eigenschaften definieren, die mittels CSS in dieser Form gar nicht erreichbar wären. Hierzu gehören unter anderem die bereits angesprochenen Bilder des *TreeView*-Controls.

Themes sind zudem nicht kaskadierend, wie dies bei CSS der Fall ist. Eigenschaften, die in einem Theme definiert sind, überschreiben prinzipiell die korrespondierenden Werte, die direkt dem

Control zugewiesen wurden. Davon abgesehen können Themes aber auch CSS-Dateien enthalten. Sind diese im Theme-Verzeichnis abgelegt, so werden sie automatisch von den einzelnen Seiten referenziert.

Mit ASP.NET wird eine Reihe von globalen Standard-Themes ausgeliefert, die Sie direkt in Ihre Websites einbinden können. Die Themes sind im folgenden Verzeichnis abgelegt:

```
<windir>\Microsoft.NET\Framework\<version>\ASP-
.NETClientFiles\Themes\
```

Jedes verfügbare Theme erhält einen separaten Unterordner, dessen Name dem des Themes entspricht. In der aktuellen Alpha-Version werden zwei Standard-Themes mitgeliefert, *BasicBlue* und *SmokeAndGlass*. Es ist zu erwarten, dass in der Beta einige weitere Themes zur Verfügung stehen werden.

Um das Theme für die gesamte Website zu verwenden, müssen Sie dieses in der Konfigurationsdatei *web.config* angeben. Hierzu dient der folgende Eintrag im *pages*-Element:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <system.web>
```

```
<pages theme="BasicBlue" />
</system.web>
</configuration>
```

Anschließend verwenden alle Seiten, soweit nicht explizit anders angegeben, das Theme *BasicBlue*. Die Auswirkungen sind in Abbildung 5 erkennbar. Die Beispielseite enthält ein *TreeView*-, ein *SiteMapPath*- sowie ein *GridView*-Control zur Anzeige der Northwind-Kunden.

Wählt man für dieselbe Seite das alternativ angebotene Theme *SmokeAndGlass*, so erstrahlt sie mit dem in Abbildung 6 gezeigten Glanz. Obwohl das Erscheinungsbild in den beiden Abbildungen deutlich anders ist, wurden an der Seite selbst keinerlei Änderungen vorgenommen.

Einzig die Einstellung in der *web.config* wurde modifiziert, sodass ein anderes Theme verwendet wird.

Intern arbeiten Themes zu einem guten Stück mit Cascading Stylesheets – also doch! Wenn Sie den Quelltext einer generierten ASP.NET-Seite anschauen, werden Sie die entsprechenden Style-Definitionen im Kopfbereich erkennen. Sofern Ihr Theme ausgelagerte CSS-Dateien enthält, werden auch diese automatisch über das virtuelle Verzeichnis *aspnet_client* referenziert.

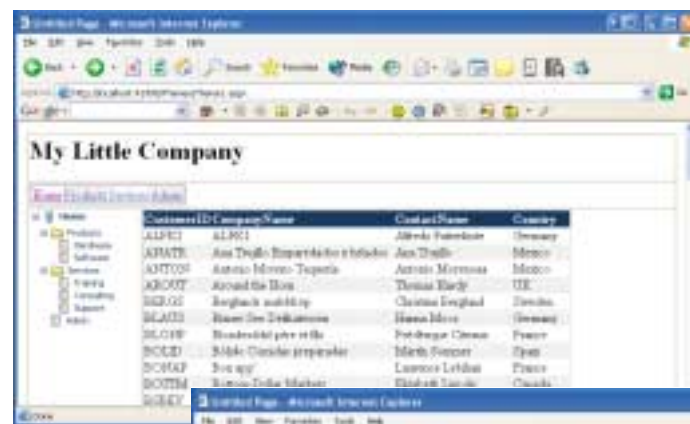


Abbildung 5
Die gleiche Seite, einmal mit dem Theme „BasicBlue“ ...

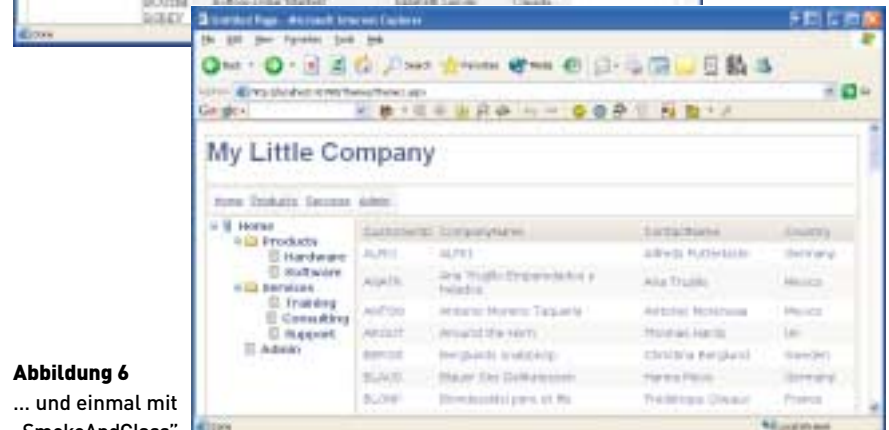


Abbildung 6
... und einmal mit „SmokeAndGlass“.

```

<html>
<head id="Head1">
<title>Untitled Page</title>
<style>
.aspxnet_s0 {
padding:1,5,1,5;color:#000066;...; }
.aspxnet_s1 { color:#000066;
text-decoration:none; }
...
.aspxnet_s13 { color:Blue;text-decoration:none; }
</style>
<link rel="stylesheet"
href="/aspnet_client/system_web/1_2_30703/
Themes/BasicBlue/whatever.css"
type="text/css" />
</head>
<body>

```

Sollten die obigen Beispiele mit einer individuell erstellten Seite nicht korrekt funktionieren, so verfügt diese möglicherweise nicht über ein serverseitiges *head*-Tag, ein neues HTML-Server-Control. Dieses ist Voraussetzung, da die gezeigten Styles anderenfalls nicht eingebunden werden können.

```

<html>
<head id="Head1" runat="server">
<title>Untitled Page</title>
</head>
<body>

```

Es wäre langweilig, wenn alle mit ASP.NET 2.0 erstellten Websites die gleichen zwei Themes verwenden würden. Selbstverständlich haben Sie die Möglichkeit, eigene Themes zu erstellen.

Um ein neues Theme anzulegen, müssen Sie zunächst im Root-Verzeichnis Ihrer Website einen Ordner mit dem Namen *themes* erstellen. Ähnlich wie die Verzeichnisse *bin* und *code* ist dieser Name fest vorgegeben, was auch durch ein spezielles Icon in VS.NET deutlich gemacht wird. Für das neu zu erstellende Theme legen Sie unterhalb des neuen Verzeichnisses einen weiteren Ordner an. Über den gewählten Namen können Sie das Theme später ansprechen. Ich habe mich für *HelloWorldTheme* entschieden.

Innerhalb des neuen Verzeichnisses legen Sie nun eine Textdatei mit dem Namen *theme.skin* an, die später die Design-Vorlagen für die Controls aufnehmen soll. Tatsächlich ist der Name der Datei irrelevant. Es kommt ausschließlich auf die Endung *skin* an. Microsoft empfiehlt sogar, für jeden Control-Typ eine separate Datei anzulegen.

Bei den globalen Vorlagen sind die Entwickler allerdings ihrer eigenen Emp-

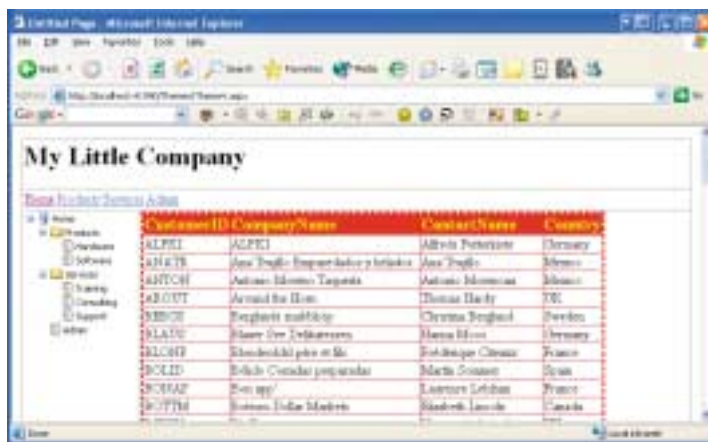


Abbildung 7
Themes lassen sich auch selbst erstellen.

fehlung nicht gefolgt. Ich schließe mich diesem Weg an und meine, dass eine einzelne Datei in aller Regel ausreicht.

Innerhalb der Datei können Sie die gewünschten Design-Vorlagen definieren. Dazu hinterlegen Sie für jeden gewünschten Control-Typ, zum Beispiel *Label*, *GridView*, *TextBox* und so weiter, eine leere *Control*-Hülle einschließlich des Attributs *runat="server"* und weisen die gewünschten Eigenschaften zu. Beachten Sie, dass keine ID zugewiesen werden sollte.

Der Inhalt der Datei könnte beispielsweise wie nachfolgend gezeigt aussehen. Hier werden die beiden Controls *TreeView* und *GridView* formatiert. Das auf die bereits zuvor verwendete Seite angewandte Theme hinterlässt deutliche Spuren, wie die Abbildung 7 zeigt.

```

<ASP:TreeView runat="server"
Font-Size="12"
/>

<ASP:GridView runat="server"
BorderColor="Red"
BorderWidth="3"
BorderStyle="Dashed"
>

<HeaderStyle
BackColor="Red"
ForeColor="Yellow"
Font-Size="15"
/>

<RowStyle
ForeColor="Black"
/>
</ASP:GridView>

```

Statt die Controls manuell in der *skin*-Datei zu erstellen und mit den gewünschten Eigenschaften zu versehen, empfiehlt sich der Einsatz der Design-An-

sicht von VS.NET. Nachdem Sie das Control dort wie gewünscht angepasst haben, können Sie es aus dem HTML-Quelltext in das Theme kopieren und anschließend nicht benötigte Eigenschaften wie die ID entfernen.

Fazit

ASP.NET 2.0 bietet jede Menge Verbesserungen, die die Entwicklung von webbasierten Applikationen deutlich vereinfachen. Die in diesem ersten Teil der Serie beschriebenen UI-Techniken beschleunigen die erstmalige Anlage einer Website und stellen sicher, dass Sie Änderungen zentral vornehmen können statt langwierige Copy-and-Paste-Aktionen vornehmen zu müssen.

Im zweiten Teil der Serie geht es um die neuen Data Controls von ASP.NET 2.0. Im dritten Teil geht es um die die Infrastrukturelemente der neuen Version, die unter anderem das Benutzermanagement, Web Parts und Personalisierung auf einfachste Weise zur Verfügung stellen.

Wenn Sie aber jetzt schon neugierig sein sollten, so finden Sie eine umfangreiche Vorstellung dieser und vieler anderer Neuerungen von ASP.NET 2.0 in [4].

[1] Uwe Baumann, Aus einem Guss, Dokumentvorlagen mit ASP.NET, dotnetpro 9/2003, Seite 20 ff.

[2] Patrick A. Lorenz, Halbe Arbeit, doppelter Lohn, PDC 2003: Neuerungen in ASP.NET 2.0, dotnetpro 11/2003, Seite 35 ff.

[3] Patrick A. Lorenz, Geballte Kraft, PDC 2003: VS.NET Whidbey für Webentwickler, dotnetpro 11/2003, Seite 44 ff.

[4] Patrick A. Lorenz, ASP.NET 2.0 Revealed, Apress Oktober 2003, ISBN 1-59059-337-5