

Hannes Preishuber

# **ASP .NET Crashkurs**

**2. Auflage**



Hannes Preishuber

# ASP .NET Crashkurs, 2. Auflage

**Microsoft**<sup>®</sup>  
Press

Hannes Preishuber: ASP .NET Crashkurs, 2. Auflage  
Microsoft Press Deutschland, Konrad-Zuse-Str. 1, 85716 Unterschleißheim  
Copyright © 2006 by Microsoft Press Deutschland

Das in diesem Buch enthaltene Programmmaterial ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor, Übersetzer und der Verlag übernehmen folglich keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programmmaterials oder Teilen davon entsteht.

Das Werk einschließlich aller Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die in den Beispielen verwendeten Namen von Firmen, Organisationen, Produkten, Domänen, Personen, Orten, Ereignissen sowie E-Mail-Adressen und Logos sind frei erfunden, soweit nichts anderes angegeben ist. Jede Ähnlichkeit mit tatsächlichen Firmen, Organisationen, Produkten, Domänen, Personen, Orten, Ereignissen, E-Mail-Adressen und Logos ist rein zufällig.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1  
08 07 06

ISBN 3-86063-988-9

© Microsoft Press Deutschland  
(ein Unternehmensbereich der Microsoft Deutschland GmbH)  
Konrad-Zuse-Str. 1, D-85716 Unterschleißheim  
Alle Rechte vorbehalten

Fachlektorat: Frank Eller  
Korrektorat: Jutta Alfes, Siegen  
Layout und Satz: Gerhard Alfes, mediaService, Siegen ([www.media-service.tv](http://www.media-service.tv))  
Umschlaggestaltung: Tom Becker, Inscale GmbH & Co. KG ([www.inscale.de](http://www.inscale.de))  
und Hommer Design GmbH, Haar ([www.HommerDesign.com](http://www.HommerDesign.com))  
Gesamtherstellung: Kösel, Krugzell ([www.KoeselBuch.de](http://www.KoeselBuch.de))

# Inhaltsverzeichnis

Vorwort .....	11
Willkommen in der Welt von ASP.NET .....	12
Deklarative Programmierung .....	12
Inhalt .....	13
Support .....	13
Danksagung .....	13
<b>1 Einführung in ASP.NET 2.0 .....</b>	<b>15</b>
Webanwendungen .....	16
ASPX-Seite .....	16
Client/Server-Beziehung .....	18
Cookies .....	19
HTTPPost .....	19
HiddenField .....	19
QueryString .....	19
JScript .....	19
Serverobjekte .....	21
Application-Variablen .....	21
Session-Variablen .....	22
Cache-Variablen .....	22
Viewstate-Variablen .....	22
Global.asax .....	23
Webserver-Steuerelemente .....	24
Ereignisbehandlung .....	24
Seiten-Lebenszyklus .....	25
Systemvoraussetzungen .....	26
Was ist neu? .....	26
Visual Studio aka Visual Web Developer .....	26
Meine erste Webanwendung .....	27
Verzeichnisse und Dateien .....	27
Code und Design .....	29
Steuerelemente .....	30
Konfiguration .....	32
Kompilierung .....	32
Cross Page Posting .....	33
<b>2 Layout und Design .....</b>	<b>35</b>
Entwurfsregeln .....	36
Masterseite .....	38
Masterseiten erstellen .....	38

Inhaltsseite erstellen .....	41
Navigation .....	45
Seitenübersicht erstellen .....	46
Menu-Steuerelement .....	49
TreeView .....	53
Mehrsprachige Websites .....	55
Designvorlagen für Seiten .....	61
WebParts .....	63
WebPart-Manager .....	64
WebPartZone .....	65
Editieren von WebParts .....	70
Design-Modus .....	71
EditorZone .....	72
Catalog-Zone .....	73
Export und Import von WebParts .....	74
Connection-Modus .....	76
<b>3 Caching .....</b>	<b>77</b>
Grundlagen .....	78
Output-Caching .....	78
Daten-Caching .....	84
Cache API .....	85
SQL-Abhängigkeit .....	86
SQL-Abhängigkeit konfigurieren .....	87
<b>4 Architektur .....</b>	<b>91</b>
Sicherheit im Design .....	92
Infrastruktur .....	92
Architektur .....	92
Betrieb .....	93
Verschlüsseln von Verbindungszeichenfolgen .....	95
Codezugriffssicherheit (Code Access Security) .....	96
Schichten .....	96
Statusinformationen .....	98
Entwickeln im Team .....	101
<b>5 Webserver-Steuerelemente .....</b>	<b>103</b>
Generelle Änderungen .....	104
Accessibility .....	104
Eingabe .....	106
Eingabeprüfung .....	109
ValidationGroup .....	110
Datenbindung .....	112
Änderungen an bestehenden Steuerelementen .....	113
Button .....	113
LinkButton .....	114
ImageButton .....	114
CheckBox .....	114

Image-Steuer-element	115
Label-Steuer-element	115
TextBox-Steuer-element	115
AdRotator	115
Calendar-Steuer-element	117
Literal-Steuer-element	118
Panel-Steuer-element	118
Table-Steuer-element	119
XML-Steuer-element	121
Änderungen am HTML-Server-Steuer-element	121
ValidationGroup	121
DefaultFocus	121
SubmitDisabledControl	121
HtmlSelect	122
Neue Webserver-Steuer-elemente	122
FileUpload	122
HiddenField-Steuer-element	124
ImageMap-Steuer-element	124
MultiView-Steuer-element	127
Wizard-Steuer-element	129
CheckBoxList-Steuer-element	133
RadioButtonList-Steuer-element	134
BulletedList-Steuer-element	135
Neue HTML-Server-Steuer-elemente	137
HTML-Head-Steuer-element	137
HtmlMeta-Element	138
HtmlLink	139
HtmlTitle	139
HTMLInputPassword	139
HTMLInputSubmit	140
HTMLInputReset	140
<b>6 Sicherheit</b>	<b>141</b>
Sicherheit unter ASP.NET	142
Cookieless Forms Authentication	143
Security-Konzept	145
Security-Grundlagen	146
SQL Server-Datenbanken vorbereiten	147
Membership-Objekt	148
Role Manager	156
Rollen aktivieren	157
Rollen zuweisen und entfernen	158
Rollen auslesen	159
Rollen erstellen und löschen	160
Programmsteuerung mit Rollen	161
Anmelden-Steuer-elemente	161
CreateUserWizard	161
Login-Steuer-element	164
LoginView-Steuer-element	166

PasswordRecovery-Steuererelement .....	167
LoginStatus-Steuererelement .....	170
LoginName-Steuererelement .....	171
ChangePassword-Steuererelement .....	171
Personalisierung mit Profilen .....	173
Konfiguration .....	174
Profiles .....	175
<b>7 Datenzugriff .....</b>	<b>179</b>
Crashkurs Datenzugriff .....	180
Einfaches datengebundenes Formular .....	180
Einfaches Datenzugriffsobjekt .....	182
Datenbindung ohne Code .....	188
Daten anzeigen .....	188
Sortieren der Ansicht .....	190
Blättern .....	191
Data Caching .....	191
ControlParameter .....	192
Daten ändern .....	193
Datenquellen-Steuererelemente .....	195
SQLDataSource .....	196
AccessDataSource .....	202
ObjectDataSource .....	204
XMLDataSource .....	208
SiteMapDataSource .....	209
Neues in ADO.NET 2.0 .....	209
Datenbindung .....	210
<b>8 DataView-Steuererelemente .....</b>	<b>213</b>
GridView .....	214
GridView-Attribute .....	214
RowStyle-Element .....	216
Blättern .....	218
Spalten .....	220
Template-Spalten .....	227
GridView Ereignis Methoden .....	228
DetailsView .....	232
DetailsView-Attribute .....	233
Daten einfügen .....	234
Master & Detail .....	236
FormView .....	239
XML binden .....	241
DataGrid .....	241
DataList .....	242
Repeater .....	244
Verschachtelte Steuererelemente .....	246



<b>9</b>	<b>Trace, Debug und Config</b>	<b>249</b>
	Ablaufverfolgung	250
	Page Tracing	250
	Ablaufverfolgung auf Anwendungsebene	251
	Trace aus Objekten	252
	Trace-Nachrichten routen	253
	Debug	254
	Ausnahmen	256
	Fehler aufzeichnen	256
	IIS-Konfiguration	257
	Websiteverwaltungs-Tool	258
	Konfigurations-API	261
	Kompilierung	263
	Health Monitoring	265
<b>10</b>	<b>Die Client-Seite</b>	<b>267</b>
	Script Jobs	268
	Fokus setzen	268
	Default Button (Standardschaltfläche)	269
	Position wieder finden	270
	Client Click	270
	Registrieren von Client Scripts	272
	RegisterClientScriptBlock	272
	RegisterStartupScript	272
	RegisterClientScriptInclude	273
	RegisterClientScriptResource	273
	RegisterHiddenField	273
	RegisterOnSubmitStatement	274
	RegisterArrayDeclaration	274
	Prüffunktionen	275
	Rückgaben mit Get	276
	Client Callbacks	276
	Chat	277
	Server-Code	277
	Die Chat-Seite	278
	Client-Code am Server erzeugen	279
	CallBackReference am Server	280
	CallBack-Logik	281
	Timer	281
	Der fertige Chat	282
<b>11</b>	<b>Konfiguration mit web.config</b>	<b>285</b>
	Allgemeines	286
	Änderungen	286
	ClientTarget	286
	Globalization	286
	HttpHandlers	287
	HttpModules	287

HTTPRuntime .....	288
Pages .....	289
WebRequestModules .....	290
DefaultProxy .....	290
Neue Bereiche .....	291
Compilation .....	291
Caching .....	291
ConnectionStrings .....	292
ExpressionBuilders .....	292
httpCookies .....	293
HostingEnvironment .....	293
MailSettings .....	293
System.Data .....	294
System.CodeDOM .....	295
<b>12 Allerlei Neues .....</b>	<b>297</b>
E-Mail-Versand .....	298
Asynchrone Seitenausführung .....	301
Cross Page Posting .....	302
URLMappings .....	303
ASP.NET Expressions .....	304
Benutzer-Steuerelement .....	305
SQL Server 2005 Express .....	305
Verwaltung .....	306
SQL Express-Datenbanken im Projekt verwenden .....	308
QuickStart Tutorials .....	310
Migration .....	311
ATLAS .....	312
AccessMembership-Provider .....	313
Stichwortverzeichnis .....	315

# Vorwort

**In diesem Vorwort:**

Willkommen in der Welt von ASP.NET	12
Inhalt	13
Support	13
Danksagung	13

# Willkommen in der Welt von ASP.NET

Beim Thema Softwareentwicklung scheiden sich noch immer die Geister. Web oder Windows? Was eine zeitlang beinahe so ausgesehen hat, als ob das Pendel Richtung Windowsanwendung ausschlägt, ist nun doch wieder ganz anders.

Vor allem unerfüllbare Wünsche an die Benutzerschnittstelle lassen eine Webanwendung als Lösung nicht sinnvoll erscheinen. Nun kommen aber mit Client Callback's, AJAX und ATLAS Technologien in Reichweite, die dieses Manko ausgleichen. Kommunikation mit dem Webserver wird damit ohne die aufwändigen Postbacks realisiert. Die Vorteile des einfachen Rollouts, des einfachen Erlernens der Benutzeroberfläche und der hohen Skalierbarkeit bleiben. So kann man fast schon von einer Renaissance der Webanwendung sprechen. Führende Firmen wie Google oder Microsoft machen es vor, welche Möglichkeiten in Webanwendungen stecken. Die Marketing-Leute haben schon einen Namen für diese neue Epoche: Web 2.0.

ASP.NET 2.0 ist zwar nicht unbedingt Web 2.0, aber es bringt erhebliche Produktivitätsfortschritte. An der häufig zitierten Aussage, 70% weniger Code impliziere auch 70% weniger Fehler, ist sicher etwas Wahres dran. Aber, und das muss ich mit aller Deutlichkeit sagen, nur wenn man sich auch darauf einlässt.

## Deklarative Programmierung

Es gibt zwei Kern-Komponenten von ASP.Net 2.0. einmal das Providermodell, das alles und jedes konfigurierbar und austauschbar macht, und die deklarative Programmierung. Gerade das zweite wirft bei vielen gestandenen OO-Entwicklern Akzeptanzprobleme auf.

Mit Hilfe des Vorgängerbuches, das noch auf der Beta 2 von ASP.NET basierte, habe ich deutlich mehr als zweihundert Entwickler auf ASP.NET 2.0 geschult. Dabei habe ich je nach Wissensgrad ein bis vier Tage pro Kurs aufgewandt. Ich habe dabei gelernt, dass der Lernaufwand für ASP.NET 2.0 steigt, da es derartig viele Funktionen gibt, die man erst einmal kennen muss, um von ihnen zu profitieren. Die zweite Erfahrung war, dass die Deklaration von Objekten statt deren Codierung erheblich auf Ablehnung stoßen kann.

Auszüge aus den gebrachten Argumenten:

»Ein Objekt existiert und kann per Code verwendet werden, obwohl es nicht per New instanziiert wurde. Schließlich hat man aus jahrelanger Erfahrung für die üblichen Probleme Lösungskonzepte im Gehirn abgelegt, die man wiederverwenden möchte. Und dann ist da noch scheinbar die Trennung von Code und Design aufgegeben worden, weil SQL Kommandos sich plötzlich in der ASPX-Seite befinden.«

Nach vielen Diskussionen mit den Entwicklern, in denen ich engagiert und voller Emotionen für diese Art der Softwareentwicklung plädiert habe, bin ich nun an dem Punkt wo ich jedem empfehle: »Machen Sie es wie Sie es für gut befinden«. Definitiv kann ich von mir sagen, dass ich mit der deklarativen Programmierung von ASP.NET 2.0 viel produktiver bin und es mir auch mehr Freude bereitet. Wenn ich heute eine Winforms-Anwendung mit .NET 2.0 entwickle, frage ich mich oft, warum gibt es das hier nicht, das geht doch auch in ASP.NET 2.0?

Also geben Sie mir und sich die Chance, und lassen sich völlig unvoreingenommen in die Welt von ASP.Net 2.0 einführen.

Am Ende muss das Ergebnis stimmen und die Software, die wir geschrieben haben, Geld verdienen.

# Inhalt

Dieses Buch ist so geschrieben, das mit der Visual Studio Web Developer 2005 Express Edition alle Beispiele nachvollzogen werden können. Auch die Abbildungen sind Screenshots dieses Produktes. Es kommt ausschließlich VB 2005 als Programmiersprache zum Einsatz, wobei die Beispiele so kurz gehalten sind, dass dies kaum eine Rolle spielt. Entsprechende Hinweise zu Unterschieden zwischen VB und C# sind vorhanden.

Für den sinnvollen Gebrauch dieses Buches sind gute Kenntnisse von C# oder VB.NET und der SQL-Abfragesprache notwendig. Ideal ist es, wenn Sie bereits Kenntnisse in ASP.NET haben. Allerdings werden durch detaillierte Ausführungen auch ASP- und Visual-Basic-Entwickler angesprochen.

Das Buch ist absichtlich kurz und knapp gehalten. Es liefert in kompakter Form einen Einstieg in die Lösung von Aufgaben mit ASP.NET 2.0. Es wird dabei keine große Anwendung gebaut, sondern es werden mit Codebeispielen jeweils die einzelnen Funktionen gezeigt. Insofern werden die MSDN-Dokumentation und auch die Quickstart-Beispiele zum unabdingbaren Werkzeug des Entwicklungsalltags gehören. Dort, wo sich Stolpersteine auftun oder die Hilfe nicht ausreicht, gibt es Praxistipps des Autors, die auch in die Tiefe gehen können.

Die Gliederung der Kapitel wurde an den Aufgaben vor, während, und nach dem Projekt ausgerichtet. Dies soll ein wenig die Sensibilität wecken, nicht unmittelbar drauflos zu programmieren, sondern mehr Zeit in die Planung und Evaluierung zu investieren.

Anhand der eigenen Erfahrungen des Autors kann dieses Buch auch hervorragend als Schulungsunterlage empfohlen werden.

# Support

Die Codebeispiele stehen auf der Webseite [www.preishuber.net](http://www.preishuber.net) zum Test und zum Download zur Verfügung. Diese Beispiele werden auch laufend aktualisiert. Der Download stellt allerdings kein lauffähiges Webprojekt zur Verfügung, sondern ist nur eine Sammlung der Beispiele.

Wer technische Fragen hat, soll diese in den Foren von [www.devtrain.de](http://www.devtrain.de) stellen. Dort werden sich der Autor und die Mitarbeiter der ppedv AG persönlich um die Antworten kümmern. Für alle anderen Fragen oder Feedback senden Sie einfach eine E-Mail an [hannesp@ppedv.de](mailto:hannesp@ppedv.de).

## Beispiele bei Microsoft

Sie können die Beispiele des Buchs auch unter

<http://www.edv-buchversand.de/mspress/support.asp>

herunterladen. Tragen Sie im Eingabefeld für die ISBN-Nummer die Zahl 532 ein, und klicken Sie dann auf *Suchen*.

# Danksagung

Wer mein Vorgängerbuch in Händen gehalten hat, wird gesehen haben, dass ich Mitglied einer kleinen Familie mit zwei Kindern, eines fünf Jahre, eines acht Monate alt, bin. Für diese beiden, falls sie jemals in ferne Zukunft dieses Buch in Händen halten sollten:

Fabian und Alina, ich liebe euch über alles und wünsche mir nichts sehnlicher als eine glückliche Zukunft für euch.

An diesem Buch haben außerdem mitgearbeitet:

*Thomas Braun-Wiesholler*, Lektor bei Microsoft Press als »Geburtshelfer«.

Das Fachlektorat übernahm *Frank Eller* selbst Buchautor und ausgewiesener .NET-Experte.

Aus dem Team der ASP .NET professional half *Nadine Schmidberger*, die das Manuskript vorab korrigierte.

Herzlichen Dank an alle erwähnten und nicht erwähnten Personen. Besonders danken möchte ich aber Ihnen, dem Leser, denn ohne Sie hätte meine Arbeit wenig Sinn. Ich hoffe, dass Ihnen dieses Buch eine Hilfe ist.

*Hannes Preishuber, im Januar 2006*

## Kapitel 1

# Einführung in ASP.NET 2.0

### **In diesem Kapitel:**

Webanwendungen	16
Client/Server-Beziehung	18
JScript	19
Serverobjekte	21
Global.asax	23
Webserver-Steuerelemente	24
Seiten-Lebenszyklus	25
Systemvoraussetzungen	26
Was ist neu?	26

Mit Active Server Pages (ASP) begann bei Microsoft die Zeit der dynamischen Webseiten. Damit war der Weg frei zur einfachen Webanwendung. Entwickler, die mit ASP versuchten, Ihr Wissen und Vorgehen aus der Windows-Entwicklung 1:1 in die Web-Welt umzumünzen, mussten damit scheitern – genauso wie es zwangsläufig schief gehen muss, ein DOS-Programm ohne Änderung des Konzeptes auf Windows zu portieren. Jede Plattform weist ihre spezifischen Eigenheiten auf. Bildlich gesprochen kann man dabei das Internet als eine Art Betriebssystem für Webanwendungen bezeichnen. Es gibt verschiedene beteiligte Komponenten, die an unterschiedlichen Orten zum Einsatz kommen. Die ASP-Seiten werden auf einem Webserver ausgeführt und nur der erzeugte HTML-Code wird an den Browser übertragen. Der Browser erzeugt dann aus dem HTML-Code ein für den Benutzer lesbares Dokument. Bei Microsoft heißt dieser Webserver *IIS* für *Internet Information Services*. Das erste Kapitel gibt Windows-Entwicklern in kompakter Form einen Einstieg in die Welt der Webanwendungen.

## Webanwendungen

In der Regel wird eine Webanwendung vom Benutzer ohne eine spezielle Schulung benutzt. Jedenfalls haben Millionen *Amazon*-Benutzer keinen speziellen Kurs absolviert und verwenden trotzdem erfolgreich die gleichnamige Webseite. Dementsprechend muss die Benutzerführung selbsterklärend sein. Ein »Handbuch zu Amazon« wird niemand zur Hand haben, geschweige denn lesen, und die unter Windows so beliebte **F1**-Hilfe findet sich im Web eigentlich nie.

Die Problematik der Benutzerführung stellt sich also im Web noch mehr als unter Windows. Die Navigation einer Website muss so ausgelegt sein, dass der Benutzer mit wenigen Mausklicks alle relevanten Stellen erreicht. Der Programmierer muss weiterhin sicherstellen, dass die Anwendung in keinem Fall in einen instabilen Modus versetzt wird. ASP.NET verbindet mehrere Webseiten zu einer virtuellen Einheit, der so genannten Webapplikation. Der Webserver (IIS) verwaltet diese Anwendung.

## ASPX-Seite

Die kleinste Einheit einer Webanwendung ist eine einzelne Seite, die in einer Datei mit der Endung ASPX abgespeichert wird. In der Regel beinhalten diese Seiten eine Mischung aus HTML-Code und ASP.NET-Steuerelementen. Ruft ein Besucher die Website auf, wird der enthaltene Code ausgeführt und der resultierende HTML-Code an den Anforderer gesandt.

## Die Sprache HTML

Tim Berners-Lee erfand die Sprache HTML mit dem Ziel einfach und für alle verständlich zu sein. HTML (*Hyper Text Markup Language*) ist eine Sprache zur Strukturierung von Texten. Es besteht auch die Möglichkeit, Grafiken und multimediale Inhalte einzubinden bzw. in den Text zu integrieren. Es können Verweise auf beliebige andere Webseiten oder Datenquellen im Internet erzeugt werden, die dadurch für den Benutzer durch einen einzigen Klick erreichbar werden. Mit HTML können Formate wie Überschriften, Absätze, Listen und Tabellen erzeugt werden. Reine HTML-Seiten bezeichnet man auch als *statische Seiten*, da sich der Inhalt nicht ändert. In der zweiten Phase des Internets sind die Entwickler dazu übergegangen, die Seiten dynamisch (d.h. mit wechselndem Inhalt, i.d.R. aus einer Datenbank) zu erstellen.

Das wichtigste Element einer dynamischen Webseite ist die Möglichkeit, Formulare in den Text zu integrieren. Zusätzlich bietet HTML Schnittstellen für Erweiterungssprachen wie beispielsweise JavaScript an, mit



dessen Hilfe HTML-Elemente nach Wunsch verändert oder die Interaktion mit dem Benutzer realisiert werden kann.

Es gibt verschiedene HTML-Versionen, wobei man die Version 3.2 als Mindestmaß bezeichnen kann, das jeder Browser einwandfrei verarbeiten kann. Die aktuellste Version ist XHTML 4.1, die sich an den wesentlich strikteren XML-Richtlinien orientiert.

Eine HTML-Seite besitzt zwingend ein Stammelement `<HTML>`. Darin enthalten ist ein Kopfbereich, eingeschlossen durch das Element `<HEAD>`, der nicht im Browser angezeigt wird, und der eigentliche Dokumentkörper (Bereich `<BODY>`), in dem der anzuzeigende HTML-Code enthalten ist. HTML-Elemente bestimmen das Ausgabeformat. Text, der zwischen dem öffnenden H1-Tag (`<H1>`) und dem schließenden Tag (`</H1>`) steht, wird beispielsweise als Überschrift erster Ordnung (und entsprechend groß) dargestellt. Ein `<br>`- oder auch `<br />`-Element bewirkt einen Zeilenumbruch. Letzteres entspricht der XHTML-Schreibweise.

```
<html>
<head >
</head>
<body>
<h1>Überschrift</h1>
das ist Text<br />
nächste <b>Zeile</b><br />
</body>
</html>
```

**Listing 1.1** HTML-Beispielseite

Jeder übliche Browser bietet auch die Möglichkeit, den HTML-Quelltext einer Seite zu betrachten.

## ASP und ASP.NET

Um die Erstellung der Seite dynamisch nach einer Codelogik zu gestalten, kann die Seite durch ASP-Elemente ergänzt werden. Eines der ältesten Elemente ersetzt dabei einen Platzhalter im HTML-Code durch einen Wert. Der ASP-Abschnitt wird durch `<%` eingeleitet und durch `%>` beendet. Das Gleichheitszeichen dient dazu, die Rückgabe der dargestellten Funktion (in diesem Fall die Ermittlung der aktuellen Zeit) in den HTML-Code zu schreiben.

```
<body>
<%=Time()%>
</body>
```

**Listing 1.2** Der ASP-Code ruft die Methode `Time()` auf

An den Browser wird dann allerdings der erzeugte HTML-Code gesandt. Man nennt diesen Vorgang auch *Rendering*.

```
<body>
23.12.2006
</body>
```

**Listing 1.3** Der erzeugte HTML-Code zeigt die Uhrzeit an

Solche Codeblöcke dürfen auch mehrfach in einem HTML-Dokument vorkommen. Auch andere Web orientierte Entwicklungsframeworks bedienen sich einer ähnlichen Vorgehensweise. Im Ergebnis erhält man einen durchmischten HTML- und Programmcode, der schwer les- und wartbar ist. Deshalb ging Microsoft mit ASP.NET einen anderen Weg und trennt hier Programmlogik und HTML-Code weitestgehend.

Ein Hilfsmittel dazu ist das `<SCRIPT>` Element. Dies wurde ursprünglich eigentlich dafür erfunden, Skripte (z.B. Javascript) auf Clientseite auszuführen. Mit ASP.NET und dem Zusatz `runat=server` wird der Inhalt zum Server-Code.

```
<script runat="server">
    Public Sub test()
        Dim i As Integer
        i = 1
    End Sub
</script>
```

**Listing 1.4** ASP.NET-Code wird in die HTML-Seite eingebettet

Die verwendete Programmiersprache wird in der *Page*-Deklaration der Seite oder in der Datei *web.config* festgelegt. Details dazu folgen etwas später.

#### HINWEIS

Manchmal sieht man die Attributwerte in Anführungszeichen eingeschlossen und manchmal ohne. Beides ist zulässig und funktioniert. Im Zuge einer konsistenten Schreibweise und in Anbetracht des XHTML-Standards, der dies zwingend erfordert, empfehle ich generell die Verwendung von Anführungszeichen.

## Client/Server-Beziehung

Wie stehen eigentlich Webbrowser und Webserver zueinander? Persönlich würde ich sagen: Sie mögen sich nicht. Im Gegensatz zu einer üblichen Client-/Server-Anwendung pflegen die beiden keine dauerhafte Beziehung zueinander. Eigentlich sind die Abrufe von Webseiten – salopp gesagt – eine Folge von andauernden One Night Stands. Technisch richtig bezeichnet man das als *statuslose Verbindung*. Der Server empfängt eine Seitenanfrage und wird diese mit einem Datenstrom beantworten. Wenn der Benutzer den Computer ausschaltet, wird das der Server nicht bemerken.

Es gibt damit zwei Programme, die in getrennten Adressräumen laufen, den Server und den Browser. Für den Austausch von Daten zwischen Server und Browser gibt es grundlegend nur wenige Methoden.

- HTTP POST
- HiddenField
- Querystring
- Cookie

Dies ist gerade für Windows-Entwickler sehr ungewohnt. Speziell in der Client-Server-Entwicklung wurde intensiv mit dauerhaften Verbindungen, wie z.B. einem Datenbank-Cursor gearbeitet. Dieser Architekturan-satz ist für heutige Anforderungen nicht zu gebrauchen. Statuslose Anwendungen skalieren wesentlich besser und können auch mit hohen Anwenderzahlen arbeiten.

Der Softwareentwickler benötigt aber das detaillierte Wissen um den Benutzer. Andernfalls könnte kein Bestellvorgang der Welt abgeschlossen werden. Dazu werden auf dem Webserver Statusinformationen

gespeichert. Mit einem Sitzungsschlüssel – *Session Key* genannt – wird die eindeutige Zuordnung zwischen Anwender und Sitzungsdaten am Server hergestellt. Der Session Key muss daher bei jeder Anfrage zwischen Server und Client übermittelt werden. Dies geschieht in der Regel mithilfe von Cookies.

## Cookies

Cookies (auf deutsch: *Kekse*) sind kleine Dateien, die beim Anwender lokal gespeichert werden. Es gibt zwei Typen. Das *temporäre Cookie* existiert nur im Arbeitsspeicher des Webbrowsers. Das *permanente Cookie* enthält ein Attribut, das die Lebenszeit (*Lifetime*) definiert und wird entsprechend lange auf der lokalen Festplatte gespeichert. Damit bleibt diese Information auch nach dem Abschalten des Computers erhalten. Cookies werden im http-Header unsichtbar übertragen und sind in der Regel auf 4 kB Daten beschränkt. Session-Cookies sind temporäre Cookies.

## HTTPPost

Normalerweise ruft ein Browser eine Webseite mit dem http-Befehl *GET* ab. Anders ist es beim Absenden eines Formulars. Dieses wird per *POST*-Befehl übermittelt und damit auch alle Werte, die der Benutzer in Eingabefelder eingegeben hat. Eine ASPX-Seite wird beim ersten Aufruf per *GET* angefordert. Alle folgenden Aufrufe werden per *POST* durchgeführt. ASP.NET verwendet zusätzliche Statusinformationen, die der Server benötigt.

## HiddenField

Auch ein *HiddenField* (verstecktes Feld) ist ein Eingabefeld, allerdings eben mit dem Zusatz *Hidden*. Deshalb sind der Inhalt und das Element selbst für den Benutzer im Browser nicht sichtbar. ASP.NET verwendet ein solches Feld mit dem Namen `__ViewState`, in dem verschiedene Statuswerte gespeichert werden, die auch vom Benutzer beeinflusst werden können.

## QueryString

Mit dem *QueryString* werden die Informationen in der URL übergeben. Dabei werden Schlüssel-/Wertepaare übergeben, beginnend mit einem Fragezeichen nach der eigentlichen abgerufenen Seite. Mehrere Werte können mittels `&` verknüpft werden.

```
Index.aspx?id=1&name=Hannes
```

So kann zwischen Seiten gewechselt werden und gleichzeitig Information von der vorangegangenen Seite mitgenommen werden. Das könnte z.B. die ID eines ausgewählten Kunden sein, dessen Bestelldaten auf der folgenden Seite angezeigt werden sollen.

## JScript

Der Webserver generiert HTML-Code und sendet diesen an den Browser. Der Browser interpretiert den Code und bereitet die Seite auf. Eine weitere Besonderheit ist, dass im Browser aber auch richtiger Programmcode ablaufen kann. Üblicherweise verwendet man als Sprache dafür *JScript*. Obwohl auf den ersten

Blick Ähnlichkeiten mit Java vorhanden sind, wie z.B. der Strichpunkt am Ende der Zeile, sind beide nicht einmal verwandt.

Die eigentlich richtige Bezeichnung, die aber selten verwendet wird, ist ECMA-Script. Dieser Code wird weder von ASP noch von ASP.NET so richtig berücksichtigt. Als Webentwickler müssen Sie neben einwandfreien HTML-Kenntnissen auch über gute ECMA-Script-Kenntnisse verfügen. JScript ist übrigens *case sensitiv* – zwischen Groß- und Kleinschreibung wird also unterschieden. Der Codeblock wird dabei durch öffnende und schließende `<SCRIPT>` Tags begrenzt. Durch das Attribut *language* wird die verwendete Sprache definiert.

```
<script language=jscript>
alert("popup");
</script>
```

**Listing 1.5** Ein PopUp-Dialog wird im Browser angezeigt

Ein häufiger Denkfehler passiert, wenn versucht wird, ein Meldungsfenster am Browser anzuzeigen. Dazu wird dann die Funktion *MessageBox* eingesetzt die, wenn überhaupt, ein Fenster am Server öffnen würde, da es sich ja um Server-Code handelt. Im Browser kann nur per JScript eine entsprechende Meldung erzeugt werden. *Alert* zeigt ein einfaches Meldungsfenster an, *Confirm* ein Fenster, das eine Bestätigung (Ja/Nein) erwartet.

Weitere wichtige Befehle erlauben es auf INPUT-Elemente zur Laufzeit am Client zuzugreifen und diese zu verändern. Mit *getElementById* kann ein vorhandenes INPUT-Element referenziert und der Wert gesetzt werden. Der Anwender tut dies per Klick auf einen Hyperlink, in dem die JScript-Funktion *setText* ausgelöst wird.

```
<input id="PLZ" type="text" />
<a onclick="jscript:setText('84489');">PLZ setzen</a>
<script type="text/javascript">
    function setText (wert) {
        document.getElementById("PLZ").value = wert;
    }
</script>
```

**Listing 1.6** JScript Beispiel aus einer ASPX-Seite

Ein weiterer Einsatzbereich für JScript ist das Öffnen eines zusätzlichen Browserfensters um z.B. einen Auswahldialog anzuzeigen. Dazu wird die Funktion *window.open* verwendet. Dabei können mithilfe von Parametern die Größe, Position und das Aussehen bestimmt werden.

```
auswahl = window.open("details.aspx?id=12", "Neues Fenster", "width=200,height=200,left=100,top=200");
auswahl.focus();
```

**Listing 1.7** Per JScript PopUp-Fenster erzeugen

Für den ASP.NET Entwickler führt auf Dauer kein Weg an JScript und HTML vorbei. Das Intellisense-Feature von Visual Studio hilft zwar beim Schreiben des Codes, ersetzt aber nicht das Grundlagenwissen.

Im nächsten Schritt werden die grundlegenden Objekte von ASP.NET erläutert.

# Serverobjekte

Mit Hilfe von Bibliotheken werden Standard-Programmieraufgaben zusammengefasst. ASP bietet speziell für die spezifischen Anforderungen von Webanwendungen seit Anbeginn die so genannten *Serverobjekte*, die diese Funktion erfüllen. Dafür gibt es sechs Objekte, die ständig zur Verfügung stehen. Diese sind sowohl in klassischen ASP als auch in ASP.NET vorhanden.

Objekt	Beschreibung
Response	Erlaubt den Zugriff auf den Ausgabestrom vom Webserver zum Browser.
Request	Erlaubt den Zugriff auf den Datenstrom vom Browser zum Server.
Context	Für den Zugriff auf den aktuellen Kontext; auch seitenübergreifend.
Server	Bietet allgemeine Hilfsfunktionen.
Application	Zugriff auf Events und Methoden die anwendungsspezifisch sind.
Session	Zugriff auf Events und Methoden die sitzungsspezifisch sind.
Trace	Neue Funktion für das Tracing (Nachverfolgung) der Anwendung.
Page	Neues Objekt für den Zugriff auf relevante Einstellungen der Seite.

**Tabelle 1.1** Server-Objekte aus ASP

Um einen Text im Browser auszugeben, kann die Methode *Response.Write* verwendet werden. Die wichtigste Funktion wird aber *Response.Redirect* sein, mit der man eine Umleitung auf eine andere Seite vom Server aus erzwingen kann.

```
<% response.redirect("neueseite.aspx") %>
```

Dies geschieht natürlich üblicherweise eingebettet in eine Programmlogik. Wenn man dann in der neuen Seite Daten aus der alten Seite weiter benötigt, braucht man die Hilfe weiterer Serverobjekte. Es gibt drei davon, die das Speichern von Daten dauerhaft erlauben und dabei jeweils eigene Szenarien abdecken.

## Application-Variablen

Wenn eine Information dauerhaft und einmalig für die gesamte Webanwendung gespeichert werden soll, bieten sich dafür *Application*-Variablen an. Der *ApplicationState* existiert einmal für jede Webanwendung, die *Application*-Variablen stehen daher allen Anwendern zur Verfügung. Die Speicherung erfolgt untypisiert.

```
Application("variablename") = "Wert"
```

Wenn der Server oder der entsprechende IIS-Prozess neu gestartet wird, verlieren die *Application*-Variablen ihre Gültigkeit. Beachten Sie hier auch das *Cache*-Objekt, das ein ähnliches Verhalten zeigt.

Wenn Sie dann ein Objekt aus der Application-Variablen zurückgewinnen möchten, muss noch eine Typumwandlung in den Ursprungsdatentyp erfolgen. Dies geschieht in VB 2005 mit dem Operator *CType* und in C# durch voranstellen des Datentyps in Klammern.

```
Dim myObj As OriginalObjekt  
myObj = CType(Application("objVariable"), OriginalObjekt)
```

**Listing 1.8** Lesen und Umwandeln einer *Application*-Variablen

Der Einsatz von Application-Variablen ist dann am Ende, wenn die Daten pro Benutzer gespeichert werden sollen. Dann müssen Session-Variablen in Betracht gezogen werden.

## Session-Variablen

In einer Session-Variablen werden Daten für jeden einzelnen Benutzer am Server gespeichert. So kann man in einer Session-Variablen z.B. den vollen Namen des Benutzers ablegen. Jede ASPX-Seite innerhalb der Webanwendung kann auf diese Variable zugreifen. Die Variable ist solange gültig wie die Session aktiv ist. Dies endet in der Regel zwanzig Minuten nach der letzten Nutzung der Session durch den Benutzer. Alternativ kann eine Session auch durch *Session.Abandon* abgebrochen werden. Die Zuweisung erfolgt ähnlich wie bei Application-Variablen untypisiert.

```
Session("variable") = TextBox1.Text
```

Bei exzessiver Nutzung von Session-Variablen können unter Umständen erhebliche Datenmengen anfallen, die den Arbeitsspeicher des Servers belasten. Dennoch sind Session-Variablen die erste Wahl zum Speichern von Daten über den Lebenszyklus einer ASPX-Seite hinaus. Jede Webseite stellt ein eigenes Datenobjekt dar und darin deklarierte Variablen verlieren nach Ausführung des Codes am Server ihre Gültigkeit.

Wenn Daten dauerhaft abgelegt werden sollen, geschieht dies am besten in einer Datenbank.

## Cache-Variablen

Daten im Cache abzulegen, funktioniert erst seit ASP.NET. Grundsätzlich ist diese Funktionalität ähnlich zur Vorgehensweise bei der Verwendung von *Application*-Variablen. Allerdings lässt sich wesentlich mehr »Intelligenz« implementieren. Funktionelle Details beim Caching und speziell die Neuerungen in ASP.NET 2.0 werden in Kapitel 3 besprochen.

## Viewstate-Variablen

*Viewstate*-Variablen sind eine weitere Neuerung in ASP.NET. Wie der Name vermuten lässt, werden darin primär Statusinformationen zur Darstellung gespeichert. So wird beispielsweise bei der Tabellendarstellung mit dem *DataGrid* sehr umfangreich von Viewstates Gebrauch gemacht. Darin enthalten sind – um bei diesem Beispiel zu bleiben – die kompletten Daten und, wo es zutrifft, die Sortierreihenfolge. Die Webserver-Steuerelemente machen von dieser Technik in großem Umfang Gebrauch. Dabei wird leider auch eine große Menge an Daten erzeugt, die vom Server zum Client und wieder zurück transportiert werden müssen. Nicht

in jedem Fall ist das notwendig (beispielsweise bei der Anzeige statischer Texte durch Label-Steuer-elemente). Deshalb lässt sich der Viewstate auch weitestgehend, aber nicht vollständig, abschalten.

## Global.asax

Ein Erbe aus ASP-Zeiten ist die Datei *global.asax*, die früher *global.asa* hieß. Notwendiger Programmcode, der z.B. für den Start der Anwendung benötigt wird, kann in die Datei *global.asax* geschrieben werden. Diese beinhaltet die passenden Ereignisse für die Standardanforderungen. Dies können z.B. der Start einer Session, der erste Start der Anwendung oder auch das Ende einer Session sein. Die folgende Tabelle liefert einen Überblick über die wichtigsten Ereignisse.

Ereignis	Beschreibung
<i>Application_Start</i>	Wird beim ersten Aufruf einer Seite durch den ersten Benutzer aufgerufen
<i>Application_End</i>	Die Funktion wird beim Ende der Anwendung aufgerufen, üblicherweise beim <i>Shutdown</i>
<i>Application_Error</i>	Eine Funktion für den Fehlerfall
<i>Session_Start</i>	Für jeden neuen Benutzer wird diese Funktion gestartet
<i>Session_End</i>	Wenn die Sitzung abgelaufen ist (üblicherweise 20 Minuten nach Inaktivität) oder per <i>Abandon</i> beendet wird

**Tabelle 1.2** Funktionen in der *global.asax*

Neben diesen Ereignissen gibt es noch ein Vielzahl weiterer, die aber sehr selten benötigt werden.

Wenn z.B. die Anzahl der aktuell angemeldeten Benutzer angezeigt werden soll, wird diese am besten in einer Application-Variablen gespeichert. Den Wert initialisiert man in der Funktion *Application\_Start*. Diese Prozedur wird einmalig ausgeführt. Meldet sich ein neuer Benutzer an, kann in der Funktion *Session\_Start* der Wert um 1 erhöht bzw. wenn die Session endet um 1 reduziert werden.

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    Application("anzahluser") = 0
End Sub
Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    Application("anzahluser") = Int(Application("anzahluser")) + 1
End Sub
Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    Application("anzahluser") = Int(Application("anzahluser")) - 1
End Sub
```

**Listing 1.9** Auszug aus *global.asax*

Generell ist die Bedeutung der *global.asax* eher gering. Es gibt einige alternative Ansätze, die auch besser in objektorientierte Konzepte passen, wie z.B. eine abstrakte Basisklasse für Webseiten.

Im nächsten Schritt wird eine ASPX-Seite um echte Funktionalität erweitert.

## Webserver-Steuerelemente

Das Verarbeiten von Benutzereingaben ist eine der häufigsten Aufgaben des Entwicklers. Also sollte dies auch so einfach wie möglich vonstatten gehen. In der Windows-Welt helfen dabei Steuerelemente seit langer Zeit. ASP-Entwickler kennen diese eigentlich in dieser Form nicht.

Mit ASP.NET gibt es nun *Webserver-Steuerelemente* die sehr ähnlich angewendet werden, wie man das z.B. aus Visual Basic seit langem kennt. Dabei wird für jedes Steuerelement ein komplettes Objektmodell erzeugt, das es erlaubt, Werte von Eigenschaften zu lesen, zu setzen und auf Ereignisse zu reagieren.

Um die Unterscheidung von HTML-Code und serverseitigen Steuerelementen zu treffen, steht Letzteren das Präfix *ASP*: voran und sie tragen immer das zusätzliche Attribut *runat="server"*. Serverseitige Steuerelemente müssen sich immer innerhalb eines *<FORM>*-Elements befinden, das ebenfalls den Zusatz *runat="server"* tragen muss. Durch zusätzliche Attribute wird das Verhalten der Webserver-Steuerelemente beeinflusst, wie z.B. die Beschriftung eines Buttons mit dem Attribut *Text*. Außerdem muss jedes Webserver-Steuerelement mit einer eindeutigen ID gekennzeichnet werden.

Typische Steuerelemente sind der *Button* als Aktionselement und das *Label* als Darstellungselement.

```
<form id="form1" runat="server">
  <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  <asp:Button ID="Button1" Runat="server" Text="Button" />
  <br />
  <asp:Label ID="Label1" Runat="server" Text="Label"></asp:Label>
</form>
```

**Listing 1.10** Auszug aus einer ASPX-Seite

Webserver-Steuerelemente werden zur Laufzeit einer ASP.NET-Seite in HTML-Elemente übersetzt. So wird dann beispielsweise eine *TextBox* zu einem *INPUT*-Element. Dabei berücksichtigt ASP.NET allerdings auch die Eigenheiten der verschiedenen Browser-Typen und erzeugt entsprechend unterschiedlichen HTML-Code, der dann im jeweiligen Browser auch funktionieren kann.

Ausgaben auf einer Website werden am besten per Label oder mit dem Literal-Steuerelement erledigt. Wenn ein Label verwendet wird, erzeugt ASP.NET ein *<SPAN>*-Element.

```
<span id="Label1">Label</span>
```

## Ereignisbehandlung

Ein Highlight aus Webentwicklersicht ist aber sicherlich die Ereignisbehandlung. Wenn Sie in der Entwicklungsumgebung einen Doppelklick auf das Button-Steuerelement ausführen, wird sogleich das zugehörige *Click*-Ereignis als Codeblock erzeugt.

Die Webserver-Steuerelemente besitzen eine Vielzahl an Ereignissen, Methoden und Eigenschaften. So kann beispielsweise mit der *Text*-Eigenschaft der Inhalt einer *Textbox* ausgelesen und mit dem *Label-Steuerelement* wieder angezeigt werden.



```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Label1.Text = TextBox1.Text
End Sub
...
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

**Listing 1.11** Ein Button mit Ereignisbehandlung

**HINWEIS** Der Zusatz *Handles* von ASP.NET 1.x entfällt, wenn sich Code- und Design-Deklaration auf der gleichen Seite befinden bzw. C# verwendet wird.

## Seiten-Lebenszyklus

Ereignisse treten auch in der ASPX-Seite auf. Das wichtigste Ereignis ist dabei die Methode *load*, die bei jeder Ausführung der Seite durchlaufen wird. Dieses wird vor eventuellen anderen Ereignismethoden von Steuerelementen ausgeführt. In der Regel wird die dazugehörige Methode mit *page\_load* benannt. Sie können durch einen einfachen Doppelklick in der Entwurfsansicht der Seite den Funktions-Prototypen erzeugen lassen.

```
<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
End Sub
```

**Listing 1.12** Leere *Page\_Load*-Funktion

Wo werden jetzt die eigentlichen Methodennamen den Steuerelementen und die Ereignisbehandlungen zugewiesen? Dies geschieht eben automatisch mithilfe der Attribute bzw. vordefinierter Attribute. Dabei ist das Attribut *Autoeventwireup* von Bedeutung, das in der ASPX-Seite in der Standard-Einstellung mit dem Wert *true* belegt ist. Wenn dieses in der Page-Deklaration geändert wird, werden auch keine Ereignisse mehr ausgeführt und müssen demzufolge manuell per Code deklariert werden. Diese Situation tritt manchmal auf, wenn man 1.x Code portieren möchte.

**HINWEIS** Falls per *AddHandler* (VB) oder *new EventHandler* (C#) die Ereignisse deklariert wurden und *Autoeventwireup* auf *True* steht, werden die Ereignismethoden doppelt ausgeführt.

ASP.Net 2.0 enthält eine Reihe zusätzlicher Ereignisse, die eine feinere Steuerung des Ablaufes erlauben. Folgende Tabelle listet einige davon in der auftretenden Reihenfolge auf.

Ereignis	Verwendung
<i>PreInit</i>	Neu in 2.0. In diesem Event ist noch keine Theme- und Personalisierungsinformation verfügbar.
<i>Init</i>	Hier ist noch kein Viewstate verfügbar.
<i>PreLoad</i>	Neu in 2.0. Wird ausgeführt nachdem alle Postback-Daten verarbeitet sind.
<i>Load</i>	Hier sind alle Steuerelemente verfügbar und können z.B. an Daten gebunden werden.
<i>PreRender</i>	Neu in 2.0. Hier kann z.B. der Viewstate noch beeinflusst und auch gespeichert werden.

**Tabelle 1.3** Ereignisse im Lebenszyklus einer Webseite

Visual Studio zeigt im Editor die Ereignismethodennamen in der Regel im DropDown-Menü oben an oder im Eigenschaftsfenster (Klicken Sie hierzu auf den Button mit dem Blitz-Symbol). Leider scheinen die C#-Entwickler hier benachteiligt zu werden. Fall die ASPX-Seite als Sprache C# ausgewählt hat, sucht man die Methodennamen vergeblich und muss diese per manueller Codierung implementieren. Manche Methodennamen, die z.B. explizit überladen werden müssen, stehen auch dem VB 2005 Entwickler nicht automatisch zur Verfügung.

## Systemvoraussetzungen

Auf dem Webserver muss das .NET Framework zwingend installiert sein. Dabei können verschiedene Versionsstände von .NET auf einem Server parallel betrieben werden. Es lässt sich mit dem Befehlszeilen-Tool *aspnet\_regiis* für jedes Web einzeln die bevorzugte .NET-Version einstellen. Der Computer des Benutzers benötigt einen modernen Browser wie den Internet Explorer oder Firefox. Client-seitig ist die Installation des Frameworks allerdings nicht erforderlich.

Für das Arbeiten mit diesem Buch wird Visual Web Developer 2005 Express Edition Deutsch vorausgesetzt. Einige der hier beschriebenen Funktionen fehlen eventuell und sind nur in höherwertigen Editionen wie der *Professional* verfügbar. Bei den entsprechenden Passagen wird jeweils ein Hinweis dazu angebracht.

---

**HINWEIS**

Web Matrix oder Visual Studio .NET 2003 sind keine geeigneten Entwicklungsumgebungen für ASP.NET 2.0. Allerdings ist es kein Problem, verschiedene Versionen von Visual Studio parallel zu betreiben.

---

Als Entwicklungsrechner sollte ein handelsüblicher PC mit einem modernen Betriebssystem ab Windows 2000 mit Service Pack 4 eingesetzt werden. Ein IIS als Webserver ist nicht nötig, da Visual Studio 2005 seinen eigenen Webserver mitbringt.

---

**HINWEIS**

Damit Sie gleich loslegen können, finden sie auf der Buch-CD den Visual Web Developer Express in der finalen Version.

---

## Was ist neu?

Die gute Nachricht vorab: Wer sich an ASP.NET 1.0 bereits gewöhnt hat und dort seine Codebibliotheken verwendet, kann diese nahezu unverändert weiterverwenden. Die noch bessere Nachricht ist: Wer sich auf Version 2.0 einlässt, kann mit weniger Aufwand deutlich mehr erreichen. Für viele der lästigen Standardaufgaben ist entweder gar keine Zeile Code mehr nötig, oder es steht zumindest ein Assistent bereit. Viele neue Webserver-Steuerelemente beschleunigen den Entwicklungsprozess genauso wie die wirklich gut gelungene Entwicklungsumgebung.

## Visual Studio aka Visual Web Developer

Damit wären wir auch schon beim ersten wichtigen Punkt der Entwicklungsumgebung. Das preislich vergleichsweise hoch angesiedelte Visual Studio 2005 wird durch ein sehr günstiges Einstiegsprodukt mit dem Namenszusatz *Express* ergänzt. Die Express-Editionen existieren immer nur für eine spezielle Aufgabe – wie für VB, C# oder die Webentwicklung. Der Name lautet im letzteren Fall *Visual Web Developer Express*. Dieser

Edition fehlen naturgemäß einige Funktionen gegenüber der Professional oder Enterprise Edition. Das sind dann beispielsweise erweiterte Tools für die Datenbankverwaltung.

Visual Web Developer ist im Vergleich zu seinem »Vorläufer« WebMatrix allerdings ein vollwertiges Entwicklungswerkzeug. Überlebensnotwendige Funktionalitäten wie z.B. Intellisense oder Debugging sind darin enthalten.

Allerdings haben die Microsoft-Entwickler viel vom pragmatischen Ansatz von WebMatrix gelernt. Einige Steuerelemente, wie beispielsweise das für die Benutzeranmeldung, wurden für WebMatrix entwickelt und finden sich in ähnlicher Form im Visual Studio wieder. Eine der auffälligsten Änderungen ist, dass es keine proprietären Projektmappen-Dateiformate im Web-Umfeld mehr gibt. Dieser Lösungsansatz war nämlich speziell bei Teamprojekten mit erheblichen Problemen behaftet. Genau wie in WebMatrix reicht eine einfache Dateistruktur als Projektverzeichnis aus. Wenn eine Webanwendung gestartet wird, startet ein eigener Webserver. Dieser hat nichts mit dem IIS zu tun und muss auch in keiner Weise konfiguriert werden.

**HINWEIS**

Offenbar hat Microsoft erkannt, dass es auch Anhänger der alten Projekt-Typen gibt und bietet deshalb einen Zusatz mit dem Namen »Web Application Project« zum kostenfreien Download an.

## Meine erste Webanwendung

Für Ihre erste Webanwendung starten Sie Visual Web Developer Express oder Visual Studio 2005. Wählen Sie im Menü *Datei/Neu/WebSite* aus. Es wird eine Verzeichnis-Struktur und eine Datei *Default.aspx* erzeugt. Mit **F5** startet der Debugger mit der aktuell angezeigten Seite. Alternativ lässt sich die Seite ohne Debugger im Browser betrachten, indem man einen Rechtsklick auf die ASPX-Datei oder in den Editor macht und den Kontextmenüpunkt *in Browser anzeigen* auswählt. Während der Debugger läuft sind keine Änderungen an der Seite möglich.

Da es keine Solution-Dateien mehr gibt, werden alle vom Typ her passenden Dateien innerhalb des Webanwendungs-Verzeichnisses automatisch mitkompiliert. Wenn man das nicht möchte, kann man einzelne Dateien durch simples Ergänzen um die Erweiterung *exclude* von der Kompilierung ausschließen. Visual Studio hilft dabei ein wenig, indem es im Kontextmenü einer Datei die Option *aus Projekt ausschließen* bzw. *zu Projekt hinzufügen* anbietet.

## Verzeichnisse und Dateien

Die Hauptarbeit des Entwicklers geschieht in den ASPX-Seiten. Diese sind ähnlich wie HTML-Seiten aufgebaut und beinhalten Elemente wie HTML, HEAD oder BODY. In der ersten Zeile einer Seite befindet sich die *Page*-Deklaration, in der per Attribut die Einstellungen für die jeweilige Seite vorgenommen werden.

```
<%@ Page Language="VB" MasterPageFile="~/all.master" Codefile="seperateCode.aspx.vb"
Inherits="seperateCode_aspx" title="Code Seperration" %>
```

**Listing 1.13** Page-Deklaration bei Code-Behind-Methoden

Neben den ASPX-Dateien für die Deklaration der Webseiten gibt es noch einige andere wichtige Dateitypen.

Dateityp	Beschreibung
.aspx	Die Webseite, in der HTML-Code, Steuerelemente, JScript und Designelemente vorhanden sind.
.ascx	Ähnlich wie die aspx-Seite, ist allerdings zum <i>Einbetten</i> in eine solche gedacht.
.vb, .cs	Der eigentliche Quellcode; kann in separaten Dateien ausgelagert werden.
.master	Seitenvorlage.
.resx	Ressourcen für Webseiten.
.sitemap	Eine XML-basierte Datei mit einer Seitenübersicht.
.skin	Designvorlage für Steuerelemente.
Web.config	Zentrale Konfigurationsdatei einer Webanwendung. Kann auch in Unterverzeichnissen existieren.
Global.asax	Datei für gemeinsamen Code der Webanwendung.

**Tabelle 1.4** Übliche Dateitypen und Dateien im Anwendungsverzeichnis

ASP.NET 2.0 weist bestimmten Verzeichnissen definierte Aufgaben zu. Diese sind in der folgenden Tabelle aufgeführt. Die Namen dieser Verzeichnisse sind nicht veränderbar.

Verzeichnis	Status	Beschreibung
App_Data	Neu	Speziell geschütztes Verzeichnis für MDB und MDF Datenbanken und andere Daten.
App_Code	Neu	Code-Bibliotheken im Quellcode. Werden automatisch kompiliert.
App_Themes	Neu	Verzeichnis für Designvorlagen.
App_LocalResources	Neu	Ressourcen-Verzeichnis für mehrsprachige Webseiten.
Bin		Als Assemblies vorkompilierte Bibliotheken. Entspricht der Verwendung in ASP.NET 1.x.
App_GlobalResources	Neu	Verzeichnis für kompilierte Ressourcen.
App_Browsers	Neu	Browser-Deklarationen können in .browser Dateien ergänzt werden.
App_WebReferences	Neu	Speicherort für Web-Proxy-Klassen, die per Web-Referenz hinzugefügt werden.

**Tabelle 1.5** Die wichtigsten Verzeichnisse für ASP.NET-Anwendungen

In Visual Studio können diese Verzeichnisse per Rechtsklick auf die Website im Projektmappen-Explorer im Kontextmenü angelegt werden.

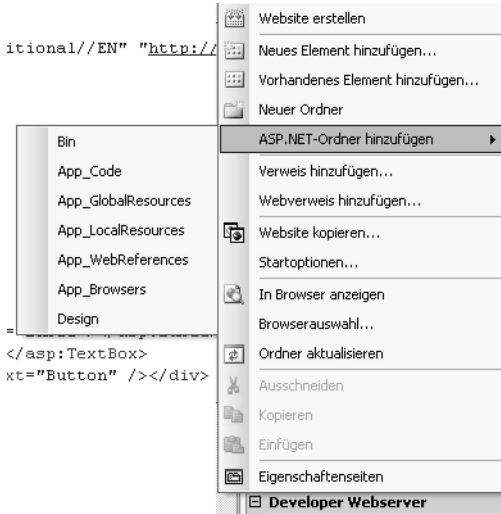


Abbildung 1.1 ASP.NET-Verzeichnis erstellen

## Code und Design

In diesem Buch wird der Code einer Seite innerhalb der ASPX-Seite angeordnet, um die Lesbarkeit zu vereinfachen. Dies wird als *Inline-Code* bezeichnet. Inline-Code wird zu Beginn der ASPX-Seite innerhalb von Script-Tags eingebettet. Durch den Zusatz `runat="server"` wird dieser Code von der ASP.NET-Engine auf dem Server ausgeführt.

```
<script runat="server">
Sub Button1Click(ByVal sender As Object, ByVal e As System.EventArgs)
End Sub
</script>
```

Listing 1.14 Ein Button-Click-Ereignis am Server verarbeiten

Von den Entwicklern wird oft die Trennung von Programmcode und Design-Code gefordert. Dies ist mit Inline-Code zwar praktisch möglich, oftmals aber nicht ausreichend deutlich. Alternativ lässt sich Programmcode in eine eigene Datei auslagern. Diese bekommt den gleichen Namen wie die ASPX-Datei und – je nach verwendeter Programmiersprache – die Erweiterung `.vb` oder `.cs`. Beim Anlegen einer neuen Datei in Visual Web Developer mit *Neues Element hinzufügen* können Sie dies per Option *Code in eigener Datei platzieren* automatisch veranlassen.

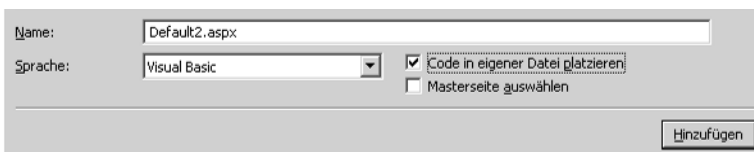


Abbildung 1.2 Zwei Dateien statt einer

**HINWEIS** Das Code-Behind-Verfahren aus ASP.NET 1.0 mit dem Attribut `Codebehind` wird von Visual Studio automatisch nach ASP.Net 2.0 portiert.

Die Verbindung der ASPX-Seite zur passenden Code-Seite wird in der *Page*-Deklaration definiert. Die Attribute *codefile* und *inherits* legen dazu die Quelldatei und die Vererbung fest.

```
<%@ Page Language="VB" codefile="seperateCode.aspx.vb" inherits="seperateCode_aspx" %>
```

Der Inhalt der eigentlichen Code-Datei ist knapp und überschaubar. ASP.NET 1.x Entwickler werden die Designer-spezifischen Code-Bereiche vermissen. Diese sind nicht mehr nötig und nicht mehr vorhanden.

```
Partial Class seperateCode_aspx
    Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    End Sub
End Class
```

**Listing 1.15** Partial Classes sorgen für Übersichtlichkeit in Code-Dateien

Das wird erst durch ein neues .NET-Feature möglich, den so genannten *partiellen Klassen*. Mit Hilfe des Schlüsselwortes *Partial* kann man Klassen über mehrere Dateien verteilen. Auf diese Weise kann die Deklaration und Instanziierung der Steuerelemente in einer ASPX-Seite geschehen und in einer Quelltext-Datei (.vb oder .cs) können die Steuerelemente dann verwendet werden. Die eigentliche Deklaration und Instanziierung ist in diesen Dateien nicht sichtbar und geschieht sozusagen »im Hintergrund«.

Wird eine Klasse (z.B. eine Seitenklasse) in einer Codebehind-Datei von einer anderen Klasse abgeleitet, so geschieht dies in VB 2005 mithilfe des Schlüsselwortes *Inherits*; in C# gibt es hierfür den *:-*Operator. Im Inline-Code wird das Attribut *Inherits* verwendet.

## Ereignisbehandlung

In einer Anwendung passiert einiges. So sendet der Benutzer seine Bestellung per Kopfdruck ab. Der Code, der dann ausgeführt werden soll, wird in die Ereignisbehandlung des Button Webserver-Steuerelements geschrieben. Im Inline Code geschieht die Zuordnung der Funktion zu dem Button über das Attribut *OnClick* des Steuerelements. Wenn Sie Visual Basic und Codebehind verwenden, fällt dieses Attribut weg und wird über den Zusatz *Handles* bei der Funktion ersetzt.

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
    ...'code
End Sub
```

**Listing 1.16** Ereignisbehandlung in ASP.NET

Die Ereignisfunktionen werden nun als *Protected* oder *Public* deklariert und erwarten zwei Parameter.

## Steuerelemente

Visual-Basic-Entwickler waren es von Anbeginn an gewohnt, ein *Steuerelement* auf ein Formular zu ziehen und mit diesem dann zu programmieren. Dieses Konzept hat in ASP.NET mit den Webserver-Steuerelementen seinen siegreichen Einzug gehalten. Man kann diese Steuerelemente in vier Gruppen unterteilen.

## HTML-Serversteuerelemente

HTML Serversteuerelemente basieren auf den standardisierten HTML-Elementen, verfügen aber über ein am Server programmierbares Objektmodell. Aus den Steuerelementen wird bei der dynamischen Generierung der Seite zur Laufzeit nahezu HTML Code erzeugt.

```
<input id="Text1" type="text" runat="server" />
```

## Webserver-Steuerelemente

Diese bieten deutlich mehr Funktionalität und erzeugen browserspezifischen HTML-Text. Aus einem Label-Steuerelement wird so z.B. ein *<DIV>*-Element. Es gibt für verschiedenste Problemstellungen Steuerelemente, wie beispielsweise für die Darstellung eines Kalenders. Seit ASP.Net 1.0 wurde das Repertoire der Webserver-Steuerelemente stark erweitert. So gibt es nun Steuerelemente für Menüs, eine Baumansicht (*TreeView*) oder auch ein *ImageMap*-Steuerelement.

## Validierungs-Steuerelemente

Diese Gruppe der Steuerelemente dient zur Prüfung von Benutzereingaben, und sie werden ausschließlich mit anderen Eingabe-Steuerelementen zusammen eingesetzt.

## Web-Benutzersteuerelement

Wenn mehrere Steuerelemente zu einem Steuerelement zusammengefasst werden, spricht man von einem Web-Benutzersteuerelement. Dieses erfüllt dann eine komplexere Funktion und kann dennoch leicht wieder verwendet werden. Praktischer Einsatzzweck ist z.B. die Darstellung von Menüs.

Ursprünglich war geplant, dass die Webserver-Steuerelemente auch den passenden Code für mobile Geräte (*Smart Devices*) erzeugen. Dieser Plan wurde aber während des Entwicklungsprozesses von Visual Studio fallengelassen, so dass nun nach wie vor ein eigenes Steuerelement-Set für mobile Endgeräte verwendet werden muss.

Viele werden sich freuen zu hören, dass es jetzt auch möglich ist, Validierungs-Steuerelemente zu gruppieren. Nehmen wir als Beispiel eine Webseite, auf der eine Zahlung vorgenommen werden soll. Der Kunde hat die Wahl zwischen Kreditkarte oder Bankverbindung als Zahlungsmethode. Die Validierungs-Steuerelemente haben bisher gnadenlos alle Prüfroutinen innerhalb einer Seite ausgeführt. Das bedeutet eine Prüfung für ein Muss-Feld erzwang die Eingabe von Bank und Kreditkarte. Zusammengehörige Steuerelemente wie *TextBox*, Validierungs-Steuerelement und der zugehörige *Absenden*-Button werden nunmehr mit dem Attribut *validationgroup* zusammengefasst. Für das genannte Beispiel gilt nun, beim Klicken auf eine Schaltfläche innerhalb der Gruppe *Kreditkarte* werden auch nur alle Steuerelemente in der Gruppe *Kreditkarte* geprüft.

## Namensräume (Namespaces)

Der Umfang der Basisklassen ist gigantisch gewachsen. Diese Klassen werden in Namensräumen zusammengefasst. Um einen Namensraum im Code einzubinden und so die darin enthaltenen Klassen nutzbar zu machen, müssen sie unterschiedliche Ansätze verwenden. Bei Inline-Code werden nach der Page-Deklaration eine oder mehrere Zeilen mit der *Import*-Anweisung gesetzt, etwa:

```
<%@ Page %>  
<%@ Import Namespace="System.Threading" %>  
<%@ Import Namespace="System.Globalization" %>
```

Bei Code-Trennung wird in der Code-Datei bei C# das *Using*-Schlüsselwort und bei Visual Basic eine *Imports*-Anweisung verwendet, etwa:

```
Imports System.IO  
Partial Class separateCode_aspx...
```

## Konfiguration

Zu den ASP-Anfangszeiten war es erforderlich, sämtliche Einstellungen einer Webanwendung im IIS-Server direkt vorzunehmen. Speziell im Hosting-Umfeld war das ein schlicht nicht zu bewältigendes Unterfangen. Auch mit ASP.NET 1.x gibt es durchaus noch Wünsche nach Verbesserungen seitens der Hoster. Häufiger Stein des Anstoßes: Die grundlegende Konfiguration wird unter ASP 1.x in der Datei *machine.config* vorgenommen. Da sich diese unterhalb des Windows Verzeichnisses befindet, verbieten manche Umgebungen den Zugriff darauf.

In ASP.NET 2.0 wurde die Konfiguration jetzt in die Dateien *machine.config* und *web.config* aufgeteilt. Die Konfiguration der Webanwendung mit der Datei *Web.Config* wird direkt im Anwendungsverzeichnis vorgenommen. Diese Datei basiert auf einem XML-Datenschema und kann direkt während des Betriebes der Anwendung geändert werden. Sogar der programmatische Schreibzugriff darauf ist möglich. Darüber hinaus besitzt ASP.NET noch eine Web basierte Verwaltungsoberfläche, um die wichtigsten Bearbeitungsaufgaben vorzunehmen. Beispielfhaft möchte ich hier das Anlegen eines Benutzers und die Zuweisung der nötigen Rechte nennen.

## Kompilierung

Bisher erforderte es ASP.NET, alle Dateien in eine große DLL zu kompilieren. Dieser Vorgang dauerte seine Zeit und ist für große Anwendungen nicht sehr sinnvoll. Auch hier hat Microsoft aus WebMatrix gelernt und betrachtet jetzt jede ASPX-Datei als Quelle zur Erzeugung einer eigenen Assembly. Der Compiler erzeugt diese einmalig völlig automatisch bei Aufruf durch den Benutzer. Wenn nun eine Datei geändert wird, muss auch nur diese neu kompiliert werden. Die so erzeugten Assemblies landen übrigens nicht im *bin*-Verzeichnis.

Durch diese Konzeptumstellung können Sie übrigens auch Programmiersprachen innerhalb einer Anwendung mischen.

Auch der Code, der als *.vb*- oder *.cs*-Datei im Code-Verzeichnis liegt, wird dynamisch kompiliert. Hier ist die Verwendung verschiedener Sprachen etwas aufwändiger. Sie müssen in der Datei *web.config* den Compiler anweisen, für Unterverzeichnisse eigene Assemblies zu erzeugen.

```
<compilation debug="true">  
  <codeSubDirectories>  
    <add directoryName="vb"/>  
    <add directoryName="cs"/>  
  </codeSubDirectories>  
</compilation>
```

**Listing 1.17** Zwei Unterverzeichnisse – *vb* und *cs* – unterhalb des Code-Verzeichnisses. Damit wird Gemischtsprachenprogrammierung möglich.



Je nach Szenario und Anforderung lassen sich trotzdem die unterschiedlichen Vorgehensweisen definieren. Neben dem dynamischen Kompilieren gibt es auch noch das Precompilation-Tool *aspnet\_compiler.exe*. Damit lässt sich selbst der HTML-Inhalt von ASPX-Dateien mitkompilieren, so dass dieser nicht mit ausgeliefert werden muss.

**HINWEIS** Auch hier liefert Microsoft einen Zusatz zu Visual Studio 2005 nachträglich aus. Mit der als Web Deployment Project bezeichneten Erweiterung lässt sich der Build Prozess umfangreich mit visueller Hilfe beeinflussen. Dies ist nicht in der Express-Version möglich.

## Cross Page Posting

Auch wenn ich persönlich noch nie das Bedürfnis verspürt habe, die Ereignisbehandlung einer Seite im Code einer anderen Seite durchzuführen, war es doch ein viel gewünschtes Feature der Entwicklergemeinde. Stellvertretend für die unzähligen kleinen Änderungen im Detail wird hier Cross Page Posting vorgestellt. Dazu besitzen die Steuerelemente *Button*, *Image* und *Linkbutton* nun ein zusätzliches Attribut *PostBackUrl*.

```
<asp:Button ID="Button1" runat="server" PostBackUrl="kapitel1postback2.aspx" Text="Button" />
```

In einer zweiten Seite kann dann auf die Steuerelemente der ersten Seite zugegriffen werden. Dazu wird die Eigenschaft *PreviousPage* verwendet. Je nach Struktur wird dann direkt mit der Funktion *FindControl* z.B. eine *TextBox* angesprochen. Zu guter Letzt muss das gefundene Objekt noch in den passenden Typ – hier in eine *TextBox* – mit *CType* gecastet werden. Erst dann kann die Eigenschaft *Text* verwendet werden.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)  
    Label1.Text = CType(PreviousPage.FindControl("Textbox1"), TextBox).Text  
End Sub
```

**Listing 1.18** Ein Steuerelement per Cross Page Posting auslesen

Weitere Details folgen im Kapitel 5.

Diese Einführung hat zunächst einmal einen Überblick, ein Gefühl dafür erzeugt, was Sie auf den nächsten Seiten erwarten wird.



## Kapitel 2

# Layout und Design

### **In diesem Kapitel:**

Entwurfsregeln	36
Masterseite	38
Navigation	45
Mehrsprachige Websites	55
Designvorlagen für Seiten	61
WebParts	63

# Entwurfsregeln

Um bei der Programmierung von Webanwendungen bestens gerüstet zu sein, muss man zu Beginn ein paar Regeln definieren. Vorab etwas Grundsätzliches zur Gestaltung der Seitenstruktur. Frames und Server Side Includes (SSI) haben seit langem ausgedient. Speziell Frames sind schwierig zu programmieren und unhandlich zu layouten. Als Ersatz stehen HTML-Tabellen oder auch Cascading Style Sheets für die Formatierung der Seiten zur Verfügung. Häufig vorkommende Elemente wie Menüs werden in Benutzer-Steuerelementen zusammengefasst und so in die Seite eingebaut. Seitenvorlagen lassen sich besonders elegant mit den neuen Master Pages von ASP.NET 2.0 erstellen. Damit steht Ihnen ein einfaches Templatesystem für Ihre Webseiten zur Verfügung.

Mit Visual Studio .NET 2002 hat Microsoft das Grid Layout und damit die pixelgenaue Positionierung als Layout-Standard eingeführt. Damit sollten Tabellen als übliches Layout-Instrument abgelöst werden. Die Positionierungsinformation wird über das *Style*-Attribut mitgeliefert. Die Entwurfsumgebung zeigt ein *Positionierungsgitter* an, um die Elemente einheitlich zu platzieren.

Tabellen sind eigentlich ursprünglich zur Darstellung von Daten gedacht. Deshalb sollen auch für Kopfzeilen von Datentabellen *<TH>*-Elemente (*Table Header*) verwendet werden und nicht wie üblich *<TD>*-Elemente (*Table Data*). Das ist auch für behinderte Personen wichtig, die Lesegeräte verwenden, die sich auf diese Daten stützen. Der Gesetzgeber schreibt dies für bestimmte Bereiche vor.

Nun werden die Bildschirmauflösungen immer größer, und die Benutzer ändern die Schriftgrößen, damit sie überhaupt noch etwas lesen können. Dadurch zerstören sie oft unabsichtlich auch noch so ausgeklügelte Designs. Hinzu kommt die mehrsprachige Gestaltung von Websites. Die Beschriftungen sind naturgemäß unterschiedlich lang. Ein bisher wenig beachtetes Phänomen sind alternative Geräte. Ein Tablet PC mit gedrehtem, im Hochformat stehendem Display hat oft Probleme mit pixelgenauen Websites. Buttons sind dann z.B. unerreichbar bei Pixel-X-Position 800 platziert.

Die beschriebene Problematik lässt sich vergleichsweise einfach mit *<table width=100%>* lösen. Schon wird der ganze Bildschirm verwendet, egal in welcher Position sich das Display am Tablet PC befindet. Eine Tabelle hilft also beim Seitenlayout.

Wenn Sie Eingabeformulare verwenden, platzieren Sie am besten jedes Eingabe-Steuerelement in einer eigenen Zelle. Auch der Text gehört in eine eigene Zelle. So kann über die Option *wrap=true* der Text vom Browser auch automatisch umgebrochen werden. Dann darf allerdings keine absolute Höhe der Zelle angegeben sein. Verwenden Sie hier am besten keine Höhenangaben und so oft wie möglich relative statt absoluter Werte.

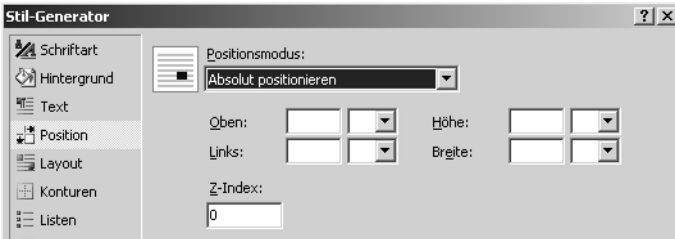
Die schwierigsten Situationen treten für uns westliche Entwickler dann auf, wenn die Webanwendung für fremde Kulturkreise verfügbar gemacht werden muss. Dann sind die bekannten Datumsproblematiken noch als harmlos zu bezeichnen. Wenn die Schrift von rechts nach links geschrieben wird, kann ein *align=left* oder *align=right* fatale Folgen haben. Verzichteten Sie deshalb innerhalb von Steuerelementen auf diese Attribute. Das Verhalten lässt sich auch per HTML-Direktive im *Body*- oder *Head*-Element steuern.

```
<HTML dir="rtl">
```

So wird die Richtung *RightToLeft* vorgegeben. Auch im unterordneten Steuerelement wie z.B. dem *<Table>*-Steuerelement ist dieses Attribut dann gültig.

All dem trägt Visual Studio 2005 Rechnung und bietet als Standardlayoutmethode wieder *Fließtext* an. Das heißt, dass ein Steuerelement nicht mehr frei im Editor hin und her bewegt werden kann.

Um ein Web-Steuerelement frei zu positionieren, muss der Umweg über die *Stil*-Eigenschaft (Style) gewählt werden: Wählen Sie dazu im Kontextmenü des Web-Steuerelements *Stil* aus. Im Dialogfeld *Stil Generator* wählen Sie in der Liste den Eintrag *Position* und setzen *Positionmodus* auf *Absolut positionieren*. Ab jetzt können Sie das Steuerelement mit der Maus auf der Webseite wieder frei positionieren.



**Abbildung 2.1** Erst durch Aktivieren im *Stil Generator* lassen sich Elemente frei positionieren

Diese Methode ist sehr aufwändig und muss für jedes Steuerelement einzeln angewendet werden. Im Browser wird ein `<SPAN>`-Element mit einem *Style*-Attribut erzeugt.

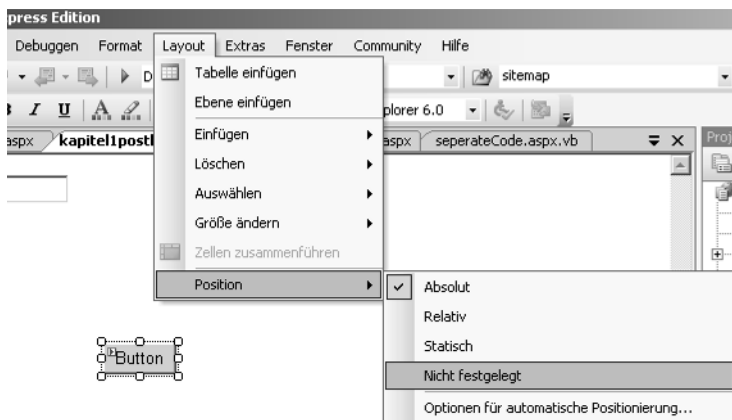
```
<span id="Label1" style="left: 112px; position: absolute; top: 128px">Label</span>
```

Schneller geht es mit dem Menüpunkt *Layout* und *Position*. Dort kann man direkt aus den Untermenüeinträgen *Absolut*, *Statisch*, *Relativ* und *nicht festgelegt* auswählen. Dazu muss das betreffende Steuerelement in der Entwurfsansicht natürlich ausgewählt sein.

Es gibt allerdings auch die Möglichkeit das Verhalten der kompletten Entwicklungsumgebung zu ändern.

Dazu muss im Menü *Layout Position/Optionen für automatische Positionierung* der Dialog geöffnet werden. Im Baum-Menü gibt es den Punkt *HMTL-Designer* und dort als Zweig *CSS Positionierung*. Dies sollte bereits vor-eingestellt sein. Im rechten Dialog kann dann über die Option *Position* aus dem Drop-Down-Menü *Absolut* ausgewählt werden. Jedes neu hinzugefügte Steuerelement erhält dann das entsprechende *Style*-Attribut.

Um dies für ein einzelnes Steuerelement wieder zu ändern, wählen Sie im *Layout*-Menü *Position* und *nicht festgelegt* aus.



**Abbildung 2.2** Ändern des Entwurfsmodus in Absolut-Positionierung

Es gibt auch zusätzliche Hilfsfunktionen, um die Steuerelemente auszurichten. Über das *Layout*-Menü lassen sich z.B. Ebenen und Tabellen erstellen.

# Masterseite

Jeder ASP- und ASP.NET-Entwickler wünscht sich ein Framework, um in Webseiten eine Layout-Struktur zu bekommen. Bisher wurden dazu meist Frames oder häufiger aufwändige selbstgebaute Template-Systeme verwendet. Dies wird durch Master Pages (Masterseite) nun wesentlich einfacher.

Mit einer Masterseite lässt sich eine Vorlagenseite für eine komplette Webanwendung erstellen. Diese Vorlage gilt dann für alle weiteren Seiten, die auch als *Inhaltsseiten* bezeichnet werden. Elemente von der Vorlage werden automatisch in die Darstellung übernommen. Dabei spielt es keine Rolle, ob man die Seiten in der Entwicklungsumgebung oder das Ergebnis im Browser betrachtet. Man sieht immer das Gesamtergebnis. Dabei findet die Zusammenführung von Master-Datei und Inhaltsdateien erst bei der Kompilierung statt. Auch spätere Änderungen an der Master Page wirken sich unverzüglich auf alle eingebundenen Seiten aus. Üblicherweise arbeitet man mit einer oder wenigen Masterseiten und vielen Inhaltsseiten, die diese Masterseiten referenzieren.

Die Masterseite wird am besten mithilfe einer HTML-Tabelle in Bereiche unterteilt. Dies könnte eine Tabelle mit zwei Zeilen und drei Spalten sein. Im oberen Bereich und damit in der ersten Zeile kann man beispielsweise ein Benutzer-Steuerelement platzieren, das zur Seitennavigation dient. Menüs mit Baumstruktur machen dagegen eher am linken Rand Sinn. Das entspräche dann der zweiten Zeile und ersten Spalte. Der Inhalt findet sich dann im mittleren Bereich – in der Tabelle entspräche dies Zeile 2, Spalte 2.

## Masterseiten erstellen

Zum Anlegen einer Masterseite wählen Sie im Menü *Website* den Punkt *Neues Element hinzufügen*. Im gleichnamigen Dialog wählen Sie aus den vorhandenen Vorlagen das *Masterseite*-Symbol aus. Damit erzeugen Sie eine Datei mit der Erweiterung *.master*. Der Name der Datei spielt keine Rolle, sollte aber funktionsbezogen gewählt sein, da es auch mehrere Master-Dateien in einem Projekt geben kann.

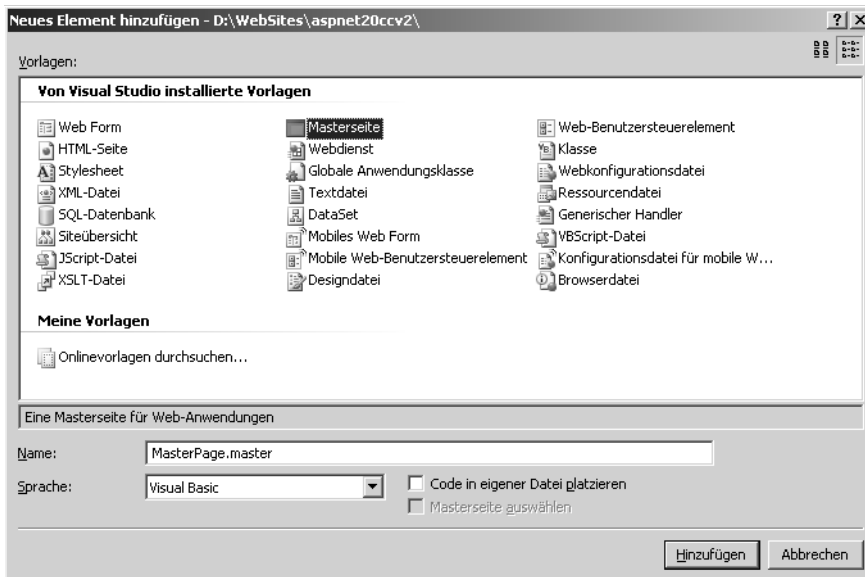


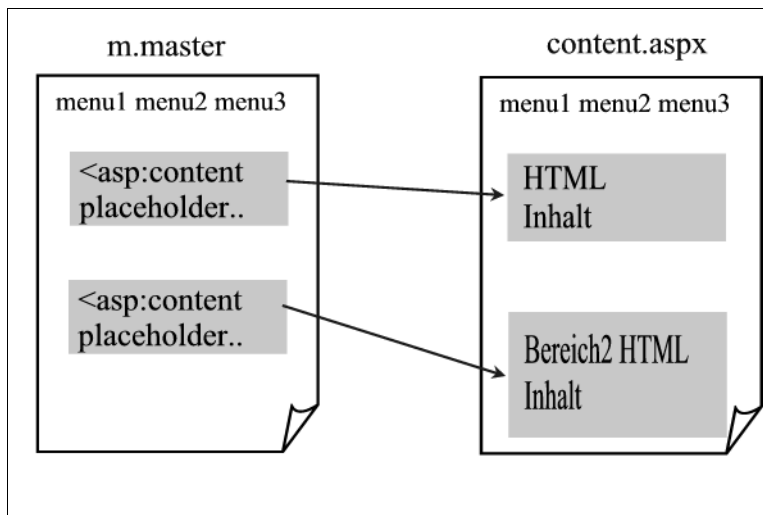
Abbildung 2.3 Anlegen einer Masterseite

Die Master-Datei kann in der Entwicklungsumgebung visuell in einer WYSIWYG Darstellung bearbeitet werden. Die Seite sieht zunächst wie eine normale ASPX-Seite aus mit einem Webserver-Steuerelement *Placeholder*. Der Inhalt der eigentlichen ASPX-Seite wird von ASP.NET zur Laufzeit an die Stelle dieses Webserver-Steuerelements gesetzt. In der Masterseite bleibt dieses Steuerelement leer. Die Designarbeiten finden in den verbleibenden Bereichen der Seite statt.

Die Master-Datei kann auch Programmlogik und damit Code enthalten, der wie in ASPX-Seiten entweder inline existiert oder in einer separaten Datei liegt. Der wesentliche Unterschied zu einer ASPX-Seite ist die Deklaration in der ersten Zeile:

```
<%@ Master Language="VB" %>
```

Das Placeholder-Steuerelement kann aus der Werkzeugleiste (Toolbox) auch mehrfach auf die Masterseite gezogen werden. So kann man eine Webseite in mehrere Bereiche unterteilen.



**Abbildung 2.4** Die Platzhalter werden in der ASPX-Seite überschrieben

Für die erste Webanwendung starten wir nun mit einer Masterseite. Diese wird mittels einer HTML-Tabelle in Bereiche für Menü und Inhalt unterteilt. Das *ContentPlaceHolder*-Steuerelement wird in eine der Tabellenzellen gezogen. Die obere Zeile wird später das Menü beinhalten. Um dieses optisch abzuheben wird ein Hintergrundbild in die Zelle eingebunden. Das kann mittels des Attributs *Background* geschehen oder über ein Stylesheet.

**HINWEIS** Ein Problem ergibt sich mit Unterverzeichnissen und Masterseiten. Es stimmen dann oft die Pfade zu verlinkten Ressourcen nicht mehr. Abhilfe verschafft die Pfadangabe mit `~` und dem Zusatz `runat=server`. ASP.NET erzeugt dann in den generierten HTML-Elementen die passenden Pfade. Das funktioniert nicht mit dem Hintergrundbild einer Tabellenzelle. Hier hilft nur die Referenzierung eines externen Styles.

Das Attribut *cssclass* des `<td>`-Elements verweist so auf eine externe Stilklasse, die in einer CSS-Datei definiert ist. Die CSS-Datei wird dann in die Masterseite eingebunden und gilt so für die gesamte Webanwendung. Dies geschieht innerhalb des Head Elements.

```
<link href="style2.css" rel="stylesheet" type="text/css" />
```

Im Kopfbereich simulieren wir mit Hyperlinks ein kleines Menü. Selbstverständlich lässt sich das Menü zu jeder Zeit wieder ändern, ohne in die einzelnen Seiten einzugreifen.

Im seitlichen Bereich wird einfach die Hintergrundfarbe mittels des Attributs *bgcolor* der Zelle gesetzt und die Breite auf *100px* (100 Pixel) festgelegt.

**HINWEIS** Falls das Attribut nicht vorhanden ist, ist wahrscheinlich das falsche Zielschema für die Validierung ausgewählt. Ändern Sie dieses in der Menüleiste dann auf Internet Explorer 6.0.

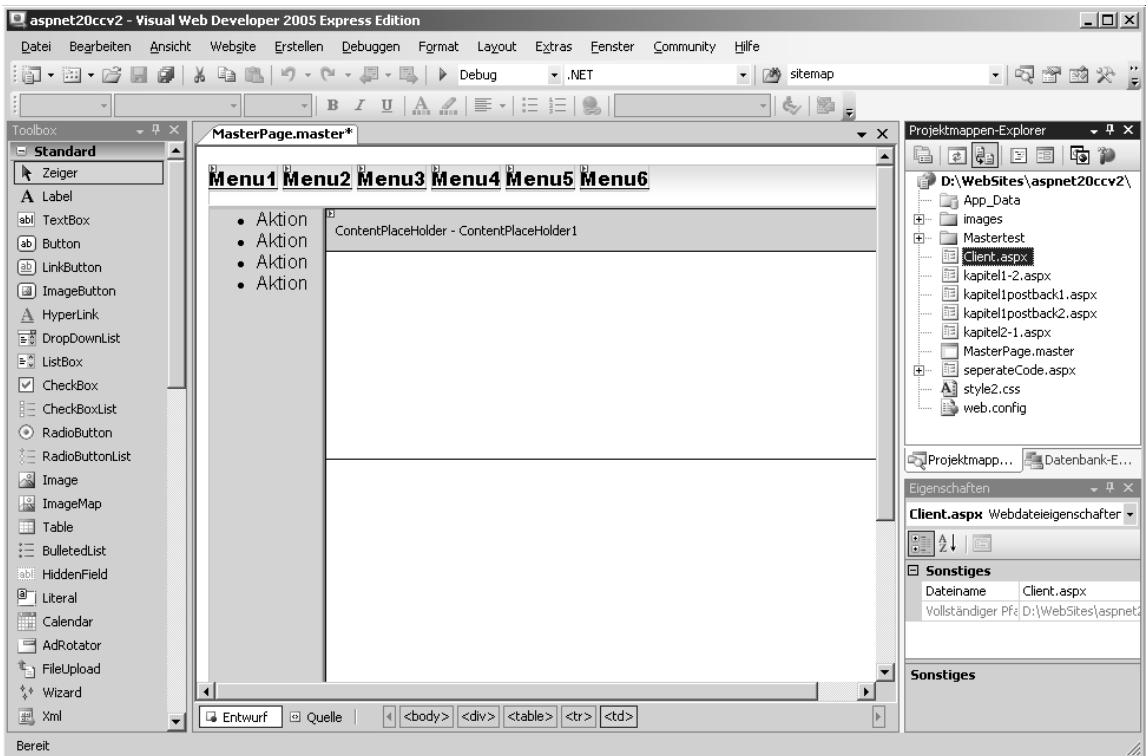


Abbildung 2.5 Benutzerführung in der Vorlagenseite

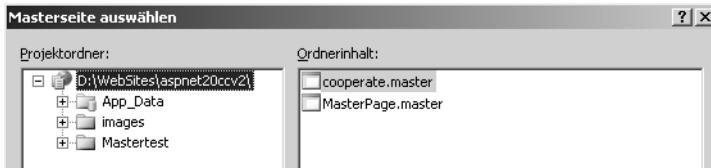
Es ist auch möglich mehrere Master Pages zu verschachteln. Stellen Sie sich einfach eine Intranetseite eines großen, weltweit agierenden Unternehmens vor. In der Corporate-Master-Vorlage wird definiert, was für alle Seiten gilt und alle Benutzer im Browser immer angezeigt bekommen. Das könnte beispielsweise der Börsenkurs sein oder ein Top-Level-Menü. Jedes Land oder jeder Standort haben dann wiederum die Möglichkeit, eigene Vorlagen zu erstellen. Diese Vorlage hat die Corporate-Vorlage als Master. Der weitere Vorgang wird hier nicht mehr näher erläutert.

**HINWEIS** Die Entwicklungsumgebung unterstützt aktuell kein visuelles Editieren von verschachtelten Master Pages



## Inhaltsseite erstellen

Nach dem Anfertigen der Masterseite wird nun eine exemplarische Inhaltsseite erstellt. Dabei muss nur beim Anlegen einer neuen *Web Form* im Dialog *Neues Element hinzufügen* die Option *Masterseite auswählen* aktiviert werden. Nach Bestätigen des Dialogs erscheint ein weiteres Dialogfenster mit der Bezeichnung *Masterseite auswählen*. Hier werden die vorhandenen Dateien mit der Endung *.master* angezeigt. Wählen Sie aus der Liste einen Eintrag aus.



**Abbildung 2.6** Eine Webseite erhält eine Vorlage

Die so erzeugte ASPX-Seite hat wesentlich weniger Inhalt als eine gewöhnliche Seite. Der komplette HTML-Teil mit den Bereichen `<HEAD>` und `<BODY>` entfällt, da sich diese Definitionen ja bereits im Master befinden. Bis einschließlich des Form-Elements fällt alles weg. In der Page-Deklaration findet sich dafür der Verweis auf die Masterseite.

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>
```

**Listing 2.1** Eine neue Inhalts-Seite

### HINWEIS

Um ASPX-Seiten nachträglich mit einer Master Page zu versehen, muss im HTML-Code alles außerhalb des Form-Elements (einschließlich dieses Elements) entfernt werden. Dann müssen Sie nur noch in der Page-Deklaration das Attribut *Masterpage* setzen und sind am Ziel.

### Titel setzen

Man möchte meinen: Einfacher geht's fast nicht mehr. Allerdings steckt auch hier der Teufel im Detail. Da die *Head*-Deklarationen ja alle im Master gemacht werden, sind sie für alle Content-Seiten identisch. Also mit anderen Worten, alle Seiten haben den gleichen Seitentitel. Dieser wird im Browser in der Taskleiste angezeigt und ist auch ein wichtiges Kriterium für eine erfolgreiche Suchmaschinenplatzierung.

Hier ist die Lösung ebenfalls ganz einfach.

Im Dialog *Property's* der aktuellen Seite findet sich unter den zahlreichen Einträgen auch der *Title*. Wenn Sie diesen hier setzen, wird in der *Page*-Deklaration das Attribut *Title* angehängt:

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" Title="meine erste Inhaltsseite" %>
```

Aber auch die restlichen Meta-Tag-Informationen sind durchaus wichtig. So macht es Sinn, über Meta-Tags die Beschreibung oder Keywords der Seite einzupflegen. Das kann zur Laufzeit über die *Page*-Eigenschaft *Header* gemacht werden. Der Header-Auflistung wird dazu ein HTMLMeta Control angehängt.

```
Dim tmpMeta As New HtmlMeta()
tmpMeta.Name = "Keywords"
tmpMeta.Content = "ASP net, Master Page"
Page.Header.Controls.Add(tmpMeta)
```

Dies setzt allerdings voraus, dass das Head-Element in der Masterseite den Zusatz *runat=server* beinhaltet. Wenn man dann diesem Element noch eine ID hinzufügt, lässt sich dieses auch direkt ansprechen bzw. mit *FindControl* finden. Mit folgendem Code wird ein externes Stylesheet eingebunden.

```
<head runat="server" id=kopf>
<link href="style2.css" rel="stylesheet" type="text/css" runat=server id="css1"/>
</head>
```

### Listing 2.2 Externes Stylesheet einbinden

In der Inhaltsseite wird innerhalb des *Page\_Load*-Ereignisses das *Link*-Steuerelement mit *FindControl* über die ID angesprochen und in den passenden Datentyp umgewandelt, um am Ende die Eigenschaft *href* setzen zu können.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    CType(Master.FindControl("css1"), HtmlLink).Href = "css1.css"
End Sub
```

### Listing 2.3 Stylesheet dynamisch verändern

Auch der Title kann so über eine Eigenschaft des *Page.Title* zur Laufzeit noch geändert werden.

Weitere Details zu den neuen HTML Server Steuerelementen finden Sie in Kapitel 5.

## Zugriff auf Masterseite

Zum vollendeten Glück fehlt nur noch der Zugriff auf einzelne Bestandteile der Masterseite. Dies können Steuerelemente, Eigenschaften oder sogar Ereignisse sein. Über das statische Objekt *Master* steht dem auch nichts im Wege. Über dieses Objekt finden Sie alles Nötige. Mit der Funktion *FindControl* lassen sich alle Web-Steuerelemente über den Namen der Master Page finden. Im folgenden Beispiel befindet sich ein Hyperlink-Steuerelement in der Master Page. Dieses soll auf einen personalisierten Hyperlink verweisen. Also darf der eigentliche Link erst zur Laufzeit gesetzt werden. Das Steuerelement wird in der Master Page mit der ID *lnkFav* benannt und ist vom Typ *Hyperlink*. Da die Funktion *FindControl* keinen spezifischen Typ liefert, muss sofort per *CType*-Funktion eine Umwandlung in den richtigen und benötigten Hyperlink-Typ erfolgen. Dadurch erhalten Sie eine Referenz in *mylnk*, über deren Eigenschaft *NavigateUrl* der Link dann gesetzt wird.

```
Dim mylnk As HyperLink = CType(Master.FindSteuerelement("lnkFav"), HyperLink)
mylnk.NavigateUrl = "http://www.devtrain.de"
```

### Listing 2.4 Zugriff aus der Inhaltsseite auf ein Steuerelement der Masterseite

Etwas aufwändiger wird es, wenn auf selbst definierte Eigenschaften der Masterpage zugegriffen werden muss. Dazu wird zunächst in der Klasse der Master Page eine öffentliche Eigenschaft (*public Property*) definiert. Wenn Visual Basic zum Einsatz kommt, werden auch gleich die *Get*- und *Set*-Methoden erstellt

```
Public _farbe As String
Public Property Farbe() As String
    Get
        Return _farbe
    End Get
    Set(ByVal value As String)
        _farbe = value
    End Set
End Property
```

**Listing 2.5** Definieren der Eigenschaft in der Masterseite

Als Nächstes wenden wir uns der Inhaltsseite zu. Durch eine zusätzliche Deklarationszeile *MasterType* wird eine strenge Typisierung der Masterseite erreicht. Es mutet etwas seltsam an, zweimal die gleiche Master Page zu referenzieren, ist aber unumgänglich:

```
<%@ MasterType virtualpath="master.master" %>
```

Der Aufwand lohnt: Schon in der Entwicklungsumgebung weist nun innerhalb der Inhaltsseite das Master-Objekt die zusätzliche Eigenschaft *Farbe* auf. Die Zuweisung eines Wertes ist dann ein Kinderspiel:

```
Master.Farbe = "rot"
```

Eine weitere Methode, auf den Inhalt der Master Page zuzugreifen, besteht darin, einfach eine Instanz davon zu erzeugen. Diese muss noch in den Typ *Master* umgewandelt werden. Dann kann direkt auf Eigenschaften zugegriffen werden, ohne dass diese als Eigenschaft deklariert sein müssen:

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim mp As _02_master
    mp = CType(Master, _02_master)
    mp.farbe = "blau"
End Sub
```

**Listing 2.6** Eine Objektreferenz auf die Master Page erstellen

---

**HINWEIS** Beginnt der Name der Inhaltsseite oder der Master Page mit einer Ziffer, wird dem Klassennamen ein Unterstrich vorgesetzt.

---

Es ist immer guter Programmierstil auf externe Objekte, und nichts anderes ist eine Masterseite, über definierte Schnittstellen zuzugreifen und nicht über interne Eigenschaften. Wenn eine Eigenschaft eines Steuer-elementes auf der Masterseite angesprochen werden soll, geschieht dies am besten über eine öffentliche Eigenschaft der Masterseite.

## Masterseite dynamisch laden

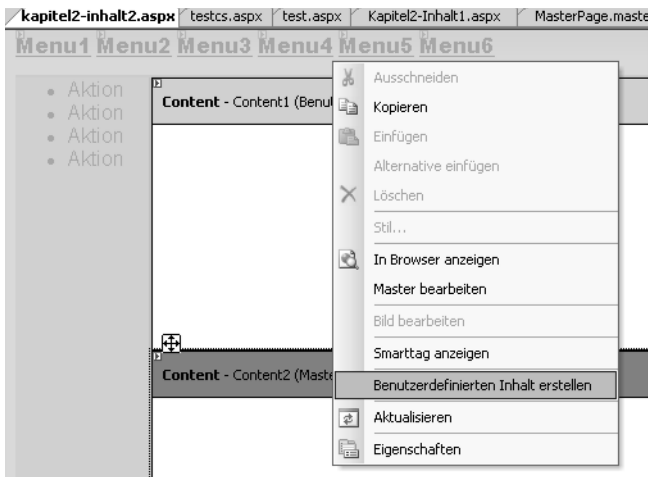
Das Zuweisen der Masterseite ist grundsätzlich auch zur Laufzeit möglich. Allerdings muss die Inhaltsseite schon vorher mit einer beliebigen Master Page verbunden sein. In dem Event *Pre\_Init* kann dann, noch bevor ASP.NET den HTML-Code erzeugt, die Verbindung zur Masterseite geändert werden. Dabei wird genau wie in der *Page*-Direktive die Eigenschaft *Masterpagefile* gesetzt.

```
Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
    Me.MasterPageFile = "~/master2.master"
End Sub
```

**Listing 2.7** Wechseln der Master Page zur Laufzeit

## ContentPlaceholder hinzufügen

Wenn in der Masterseite nachträglich ein *ContentPlaceholder*-Steuerelement hinzugefügt wird, ist dieses nicht automatisch in den Inhaltsseiten verfügbar. Das zusätzliche Steuerelement ist zwar in der Entwurfsansicht sichtbar, aber in der Farbe ein dunkleres Grau. Erst ein Rechtsklick auf das Steuerelement zeigt im erscheinenden Kontextmenü den Eintrag *Benutzerdefinierten Content erstellen* mit dem dann auch ein Content-Steuerelement in der Inhaltsseite erstellt wird.



**Abbildung 2.7** Contentplaceholder in Inhaltsseite aktivieren

Ein ContentPlaceholder-Steuerelement lässt sich auch an Stellen platzieren, die auf den ersten Blick ungewöhnlich erscheinen. Mit dieser Vorgehensweise kann man auch die Suchmaschinen Optimierung von Webseiten vereinfachen, indem man dieses Steuerelement einfach in das *Head*-Element der Masterseite legt. Um die Zuordnung später zu erleichtern, wird das *ID*-Attribut auf einen sprechenden Namen gesetzt, wie Metainfos.

```
<head runat="server" id=kopf>
<link href="style2.css" rel="stylesheet" type="text/css" runat=server id="css1"/>
  <asp:contentplaceholder runat="server" id="Metainfos"></asp:contentplaceholder>
</head>
```

**Listing 2.8** ContentPlaceholder wird für Metatags verwendet

In den Inhaltseiten können dann die Metatags ganz ohne Eingriff in den Code bearbeitet werden, wie das Designer oder auch SEO's (*Search Engine Optimizer*) gewohnt sind. Selbst die Entwicklungsumgebung zeigt sowohl die Masterseite als auch Inhaltsseiten korrekt im Entwurfsmodus an.

```

<%@ Page Language="VB" MasterPageFile="~/MasterPage2.master" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MetaInfos" Runat="Server">
  <meta NAME="Title" CONTENT="Termine Schulung Weiterbildung">
  <meta NAME="Copyright" CONTENT="ppedv AG 2005">
  <meta NAME="Revisit" CONTENT="After 30 days">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>

```

**Listing 2.9** Inhaltsseite mit Metatags

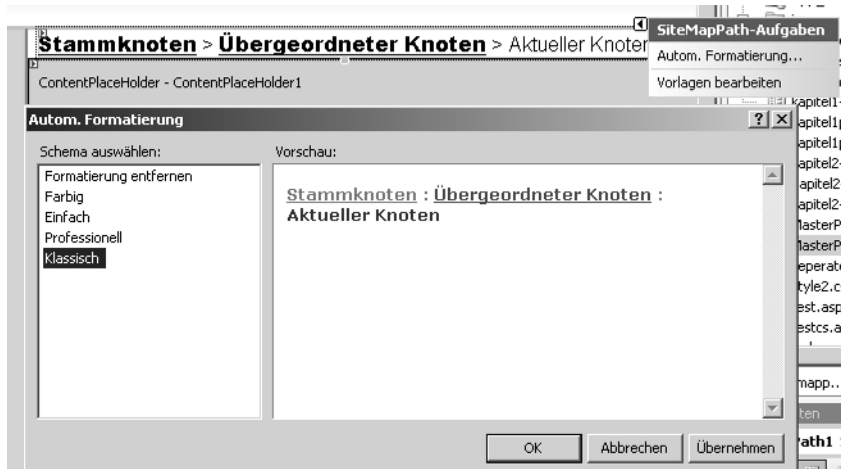
## Navigation

Benutzerführung ist eine der wichtigsten Aufgaben eines Entwicklers von Benutzerschnittstellen. Oft spielen dabei auch Design-Gesichtspunkte oder die Optimierung für Suchmaschinen eine Rolle. In jeder Webanwendung müssen Sie sich mit Benutzerführung anhand von Menüs und Navigationshilfen auseinandersetzen. Dies erfordert Zeit und ist nicht besonders spannend. Auch hier können Sie in ASP.NET 2.0 mit weniger Aufwand mehr erreichen.

Die drei »Navigations-Musketiere« *Menu*, *TreeView* und *SiteMapPath* kämpfen gemeinsam für die gute Sache. Diese Steuerelemente befinden sich in der Werkzeugleiste unter dem Punkt *Navigation*.

Allen dreien ist gemein, dass sie an Datenquellen gebunden werden können. Das *TreeView*- und *Menu*-Steuerelement lassen sich alternativ oder ergänzend auch manuell per Deklaration mit Einträgen füllen. Eine Neuerung in ASP.NET 2.0 ist, dass das Kontextmenü *Aufgaben* (Smarttag) bei den meisten Steuerelementen verfügbar ist. In diesem Menü finden sich Steuerelement spezifische Aufgaben, die meist durch Assistenten unterstützt werden. Das Aufgaben-Menü wird durch einen Klick auf den kleinen schwarzen Pfeil rechts oben im Steuerelement aktiviert.

Auch die Navigations-Steuerelemente nutzen dies und so kann z.B. das visuelle Outfit aus Design-Vorlagen ausgewählt werden.



**Abbildung 2.8** Mit *Autom. Formatierung* dem *SiteMapPath*-Steuerelement ein professionelles Design zuweisen

Mit dem Design-Template-System lassen sich auch ausgefallene Wünsche realisieren. Meist reicht es aber aus, mit den einfachen Standardeinstellungen zu arbeiten. Details zur Verwendung von Steuerelement-Templates lernen Sie in Kapitel 8 bei der Beschreibung der Daten-Steuerelemente näher kennen.

Die eigentliche Struktur einer Webanwendung ist in der Regel hierarchisch. Dabei werden meist nicht alle vorhandenen Seiten in die Navigation eingebaut. Manche Seiten will man nicht anzeigen bzw. machen für sich genommen keinen Sinn. Das folgende Beispiel zeigt eine typische Struktur einer Webanwendung :

```
Home
  Artikel
    ASPNET-Artikel
    XML-Artikel
    SQL-Artikel
  Foren
  ...
```

Die Erstellung einer solchen Benutzerführung wird im Folgenden erläutert.

## Seitenübersicht erstellen

Genau solche Strukturen werden mithilfe von *Sitemap*-Dateien abgebildet. Leider gibt es nur wenig Hilfsmittel oder Assistenten für das Erstellen der Seitenübersicht und es muss viel von Hand gemacht werden. Eine Sitemap-Datei kann über *WebSite/Neues Element hinzufügen* und die Vorlage *Seitenübersicht* erstellt werden. Im Editor von Visual Studio steht IntelliSense beim Editieren zur Verfügung. Eine Funktion zum Einlesen der vorhandenen Dateien in die Struktur ist nicht vorhanden.

Die Seitenübersicht basiert auf XML und muss sich deshalb an die Wohlgeformtheitsregeln halten. Das Root-Element heißt *siteMap*. Darunter muss es mindestens einen *SiteMapNode* geben. *SiteMapNodes* lassen sich in beliebiger Tiefe schachteln. Die Schachtelung stellt so die Struktur der Anwendung dar, hat aber nichts mit den wirklichen Verzeichnissen zu tun.

Als Attribute sind die URL der Seite samt Pfad, ein Titel und eine Beschreibung anzugeben. Die URL darf auch Querystrings enthalten, muss aber innerhalb der Seitenübersicht im Ganzen eindeutig sein.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode url="default.aspx" title="home" description="Startseite">
    <siteMapNode url="02/artikel.aspx" title="Artikel" description="Artikel">
      <siteMapNode url="artikelASPNET.aspx" title="ASP.NET Artikel" description="ASP.NET Artikel"/>
      <siteMapNode url="artikelXML.aspx" title="XML Artikel" description="XML Artikel"/>
      <siteMapNode url="artikelSQL.aspx" title="SQL Artikel" description="SQL Artikel"/>
    </siteMapNode>
    <siteMapNode url="forum.aspx" title="Forum" description="Forum" />
    <siteMapNode url="fake.aspx" title="Fake" description="Fake">
      <siteMapNode url="02/navigation.aspx" title="Navigation Sample" description="Navigation Beispiel" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

**Listing 2.10** Die Seitenübersicht im XML Format

Es gibt noch weitere mögliche Attribute eines *SiteMapNode*-Elements, die hier kurz erläutert werden.

Attribut	Verwendung
<i>Description</i>	Der Description-Tag wird im Browser als ToolTip angezeigt.
<i>Provider</i>	Liest oder setzt den verwendeten Sitemap-Provider.
<i>Roles</i>	Liest oder setzt die Rollen. Mehrere Rollen werden durch Komma getrennt. Damit werden die Zugriffsrechte auf den Menüpunkt gesteuert.
<i>Title</i>	Dies dient zum Anzeigen des Menütextes.
<i>Url</i>	Die Zieladresse als Zeichenkette, die bei Auswahl des Menüpunktes aufgerufen werden soll.
<i>SitemapFile</i>	Damit kann eine externe Sitemap-Datei spezifiziert werden, die weitere Node Elemente enthält.
<i>SecurityTrimmingEnabled</i>	Sicherheitseinstellungen werden je nach booleschem Wert ignoriert. Dies wird am besten zentral in den Einstellungen des Providers gesetzt.
<i>ResourceKey</i>	Zeichenkette, die als Schlüssel für die Lokalisierung verwendet wird.

**Tabelle 2.1** Ausgewählte Attribute des SiteMapNode-Elements

**HINWEIS** Das Attribut *SecurityTrimmingEnabled* in einer Sitemap-Datei funktioniert nicht mit dem *SitemapDataSource*-Steuerelement. Es wird zur Laufzeit eine Exception ausgelöst. Als Alternative kann das *XMLDataSource*-Steuerelement verwendet werden, um die *Sitemap*-Datei an ein Navigations-Steuerelement zu binden.

## Alternative SiteMaps verwenden

Eigentlich wurde hier beschrieben, dass es nur eine Seitenübersicht im Stammverzeichnis der Webanwendung geben kann. Und doch sind alternative Seitenübersichten möglich. Dazu muss man zuerst wissen, dass in der zentralen Konfigurationsdatei *web.config* (*C:\WINDOWS\Microsoft.Net\Framework\v2.0.50727\CONFIG\*) ein Bereich *siteMap* definiert ist. Dort wird dann ein Sitemap-Provider deklariert, der diese wunderbare Verbindung zur *web.sitemap* herstellt.

```
<siteMap>
  <providers>
    <add siteMapFile="web.sitemap" name="AspNetXmlSiteMapProvider"
      type="System.Web.XmlSiteMapProvider, System.Web, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
  </providers>
</siteMap>
```

**Listing 2.11** Auszug aus *web.config*

Der Provider ist im Attribut *type* deklariert. Darin versteckt sich die eigentliche Funktionalität.

Was hier so reibungslos funktioniert, lässt sich natürlich auch in der Konfigurationsdatei der Webanwendung nutzen. In der beiliegenden Beispielanwendung wird für die Verwaltung der Samples eine Sitemap mit dem Namen *beispiele.sitemap* verwendet, um nicht mit dem eigentlichen Beispielcode zu kollidieren. Es wird dafür einfach nur ein zweiter identischer Sitemap-Provider an die Provider-Auflistung angehängt. Das passiert in der Datei *web.config* einfach per *Add*.

```
<siteMap enabled="true">
  <providers>
    <add name="MySiteMapProvider"
      description="eine andere Sitemap"
      type="System.Web.XmlSiteMapProvider, System.Web, Version=2.0.3600.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"
      siteMapFile="beispiele.sitemap" />
  </providers>
</siteMap>
```

**Listing 2.12** So wird die *web.config* um einen Provider erweitert

Der Name des Providers muss dann in der Eigenschaft *SiteMapProvider* des *SiteMapPath* oder *SiteMapDataSource*-Steuerelements angegeben werden.

```
<asp:SiteMapPath ID="SiteMapPath1" Runat="server" SiteMapProvider="MySiteMapProvider">
```

## SiteMapPath verwenden

Umfangreiche Webseiten helfen dem Benutzer, indem sie ihm jederzeit die aktuelle Position anzeigen, also die jeweilige Seite und den Pfad, der zur Seite hinführt. Diese Funktionalität wird auch als *Breadcrumb* (etwa: *Brotkrumen*) bezeichnet. Das *SiteMapPath*-Steuerelement visualisiert dies mithilfe der Seitenübersichts-Steuerdatei, die als Karte für die Anwendung fungiert.

Nachdem Sie die Übersichtsdatei erstellt haben, wird die Navigation in die Webanwendung mithilfe der Visualisierungs-Steuerelemente eingebaut. Am besten platzieren Sie in Ihrem Projekt die Navigationsinformationen in der Masterseite. Andernfalls muss man in jede ASPX-Seite die Steuerelemente einfügen. Eventuell kann man auch mehrere Steuerelemente in Benutzersteuerelementen zusammenfassen.

Ziehen Sie also aus der Werkzeugleiste das *SiteMapPath*-Steuerelement in ihre Masterseite. Erstellen Sie dann eine Inhaltseite und starten Sie diese im Browser.



**Abbildung 2.9** Das *SiteMapPath*-Steuerelement

Wenn der Name der angezeigten Seite zu einem Knoten (*Node*) des Seitenübersichts-Dokuments passt, wird der gesamte Pfad angezeigt. Andernfalls bleibt der gesamte Pfad leer.

### HINWEIS

Der Name der ASPX-Seite und der komplette Pfad müssen im Attribut *URL* angegeben werden.



Mit Hilfe des *SiteMapPath*-Steuerelements kann sich der Benutzer zwar rückwärts in der Seite bewegen, aber nicht vorwärts, tiefer in die Hierarchie hinein. Um das zu realisieren, brauchen Sie andere Menü-Steuerelemente wie *Menu*.

## Menu-Steuerelement

Was benötigt jede Webanwendung, was aber in ASP.NET 1.x nicht vorhanden ist? Richtig: Ein Menü-Steuerelement.

Es ist fast schon selbstverständlich, dass es dieses jetzt in der Version 2 endlich gibt. Und es wurde richtig gut umgesetzt. Das Menu-Web-Steuerelement lässt kaum Wünsche offen und ist dabei wirklich leicht zu handhaben.

Ohne eine Zeile Code schreiben zu müssen, kann eine komplette Seitennavigation erstellt werden. Diese ist sowohl mit PopUps oder mit statischer Anzeige möglich.

### Menü per Hand

Zum Kennenlernen empfiehlt es sich, das Menü-Steuerelement einfach aus der Werkzeugleiste auf die Webseite zu ziehen. In dem Kontextmenü *Menü Aufgaben* lassen sich mithilfe von Assistenten die wichtigsten Aufgaben erledigen. Dort wird auch der *Menüelemente bearbeiten*-Editor gestartet, der visuelles Editieren der Menüeinträge auf verschiedenen Ebenen erlaubt. Es werden einfach Menüpunkt, Ziel-URL oder die URL für das Icon eingetragen.

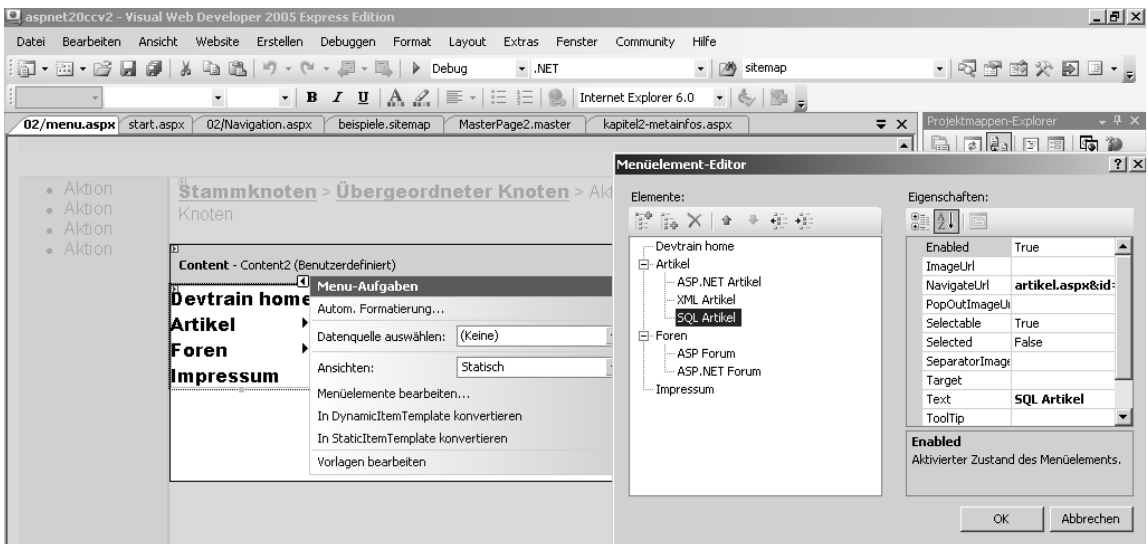


Abbildung 2.10 Menü mit manuell erstellten Einträgen

Das Menu Steuerelement enthält dann ein *Items*-Element, in dem sich die *ASP.MenuItem*-Elemente befinden, die in sich nochmals geschachtelt sein können.

```

<asp:Menu ID="Menu1" runat="server">
<Items>
<asp:MenuItem Text="Devtrain home" Value="New Item" NavigateUrl="home.aspx">
  </asp:MenuItem>
  <asp:MenuItem NavigateUrl="artikel.aspx" Text="Artikel" Value="Artikel">
    <asp:MenuItem NavigateUrl="artikel.aspx&id=ASP.NET" Text="ASP.NET Artikel" Value="ASP.NET Artikel">
  </asp:MenuItem>
    <asp:MenuItem NavigateUrl="artikel.aspx&id=XML" Text="XML Artikel" Value="XML Artikel">
  </asp:MenuItem>
    <asp:MenuItem NavigateUrl="artikel.aspx&id=SQL" Text="SQL Artikel" Value="SQL Artikel">
  </asp:MenuItem>
  </asp:MenuItem>
  <asp:MenuItem Text="Foren" Value="Foren">
    <asp:MenuItem Text="ASP Forum" Value="ASP Forum">
  </asp:MenuItem>
    <asp:MenuItem Text="ASP.NET Forum" Value="ASP.NET Forum">
  </asp:MenuItem>
  </asp:MenuItem>
  <asp:MenuItem Text="Impressum" Value="Impressum">
  </asp:MenuItem>
</Items>
  <StaticMenuItemStyle HorizontalPadding="0px" VerticalPadding="0px" />
  <DynamicMenuItemStyle HorizontalPadding="0px" VerticalPadding="0px" />
</asp:Menu>

```

**Listing 2.13** Ein Teil eines ASPNET Menü-Web-Steuerelements

Soweit ist das erste Menü also fertig gestellt und funktionsfähig. Die oberste Ebene wird zur Laufzeit statisch angezeigt. Die Darstellung der unteren Menü-Ebenen erfolgt dynamisch als PopUp. Das dazu verwendete JScript funktioniert auch in anderen Browsern wie Mozilla Firefox.

#### HINWEIS

Wenn ein Menüpunkt nicht auswählbar sein soll, setzen Sie das Attribut *Selectable=false*. Die Untermenüpunkte sind weiter wählbar.

## Datengebundenes Menü

Oft kommen die Inhalte von Menüs aus einer Datenbank. Das Konzept von ASP.NET 2.0 trennt mithilfe eines Providers Visualisierung und Datenspeicherung. Um die Verbindung wieder herzustellen, wird ein weiteres Steuerelement eingesetzt, genannt *SiteMapDataSource*. Prinzipiell können aber alle in der Werkzeugleiste im Bereich *Daten* vorhandenen DataSource-Steuerelemente verwendet werden.

Auch dazu muss kein eigentlicher Quellcode geschrieben werden. Es reicht aus, die Eigenschaften zuzuweisen und z.B. ein *XMLDataSource*- oder *SiteMapDataSource*-Steuerelement zu binden.

Ziehen Sie aus der Werkzeugleiste das *SiteMapDataSource*- und das *Menu*-Steuerelement in den Kopfbereich der Masterseite. Dann wählen Sie im Kontextmenü des Menü-Steuerelements *Menu Aufgaben* in der Aufklappliste *Datenquelle auswählen*. Danach wählen Sie den Namen des vorher angelegten *SiteMapDataSource* Steuerelements *DataSource1* aus.

Im Browser klappt das Menü dann beim Kontakt mit dem Mauszeiger die Untermenüs aus. Ob ein weiterer Menüpunkt vorhanden ist, zeigt das kleine schwarze Dreieck an.



Abbildung 2.11 Ein einfaches Menü aus einer Sitemap generiert

Den Ästhetern wird das etwas zu einfach wirken. Auch für diese Menschen stellt das *Aufgabenmenü* einen Assistenten für die automatische Formatierung bereit. Dieser zeigt sofort eine Vorschau auf das zu erwartende Ergebnis an.

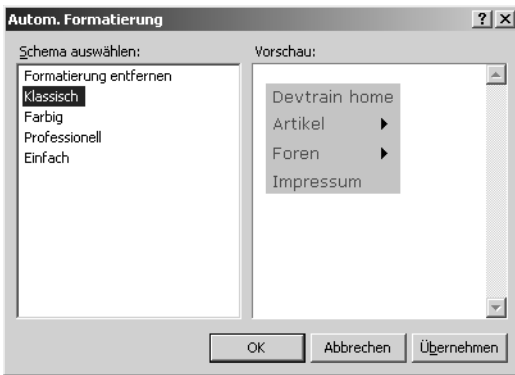


Abbildung 2.12 Der Format-Assistent

Menüs können statisch oder dynamisch sein. Das dynamische Menü klappt bei Bedarf über den vorhandenen Inhalt der Webseite weitere Menüpunkte auf. Mit dem Attribut *ToolTip* des Menüeintrags kann sogar die genauere Beschreibung des Menüpunktes angezeigt werden.



Abbildung 2.13 Menü mit dynamischen PopUps

PopUp-Menüs sind nicht jedermanns Sache. Auch Suchmaschinen können solch dynamisch erzeugten Links eventuell nicht folgen. Also kann man die Anzahl der statisch angezeigten Ebenen verändern. Diese Einstellung kann über den Dialog *Eigenschaft* vorgenommen werden. Dort muss die Eigenschaft *StaticDisplayLevels* auf »3« gesetzt werden, um drei Ebenen statisch anzuzeigen. Die Darstellung der Menüpunkte erfolgt dann je Ebene versetzt. Der vorher überlappte Text wird nun komplett unterhalb dargestellt.



**Abbildung 2.14** Dauerhaft und vollständig vorhandenes Menü

Die Menüstruktur muss in der Seitenübersicht immer mit einem Root-Element beginnen, das in Abbildung 2.13 als *Devtrain home* dargestellt wird. Dies soll vermutlich dem Benutzer den Weg auf die Startseite erleichtern, wirkt aber manchmal unschön. Aber auch dieses Verhalten lässt sich beeinflussen, wenn die Menüstruktur aus einer *Sitemap*-Datei kommt. Das *SiteMapDataSource*-Steuerelement beinhaltet das Attribut *ShowStartingNode*, mit dem dieser Root Node ausgeblendet werden kann.

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" ShowStartingNode="false"/>
```

### Vertikale oder horizontale Anordnung der Menüelemente

Sehr oft erwartet der Benutzer wie bei *Windows.Forms* die Menüs am oberen Bildschirmrand. Bei Auswahl eines Menüpunktes klappt dann eine Box mit den Menüeinträgen herunter. Um das auch mit Menü-Steuer-elementen auf ASPX-Seiten zu realisieren, gibt es die Eigenschaft *Orientation*. Setzen Sie einfach in den Attributen des Menü-Steuer-elements folgende Werte:

```
Orientation="Horizontal" StaticDisplayLevels="2">
```

Da die *Sitemap* nur ein Root-Element enthält, ist es sinnvoll gleich die zweite Menü-Ebene mit einzublenden. Das geschieht mit dem Attribut *StaticDisplayLevels*. Die Darstellung findet anschließend komplett horizontal statt. So werden die Menü-Einträge der Ebene eins und zwei in einer Zeile dargestellt und die Einträge der Ebene drei per *PopUp*.



**Abbildung 2.15** Horizontales Pull-down-Menü

### Programmieren von Menüs

Natürlich lassen sich Menüs auch richtig programmieren. Ob zum Füllen der Menüpunkte oder zur Ereignisbehandlung spielt dabei keine Rolle. Exemplarisch wird hier das *Click*-Ereignis beschrieben. Das Menü dient dann weniger der Navigation, sondern der Programmsteuerung. Das Ereignis *MenuItemClick* wird ausgeführt, wenn der Benutzer einen Menüeintrag auswählt. Die Funktion erwartet dabei zwei Parameter. Über den zweiten Parameter *e* kann dann der vom Benutzer gewählte Eintrag ermittelt werden.

```
Private Sub MenuItemClick(ByVal sender As Object, ByVal e As MenuEventArgs)
    Select Case e.Item.Value
        Case "Forum"
    ...

```

**Listing 2.14** Behandlung des Click-Ereignisses im Menü-Steuerelement

## Zusätzliche Eigenschaften

Wenn Sie auf das Erscheinungsbild einzelner Menüpunkte Einfluss nehmen möchten, bietet sich das Ereignis *MenuItemDataBound* an. Damit können Menüeinträge direkt bei ihrer Erzeugung verändert werden. So kann man als Beispiel ein neues Fenster im Browser öffnen, statt einfach auf die neue Seite umzuleiten. Dies geschieht in einem Link-Element mit dem Zusatz Attribut *target=\_blank*. Um diese Information überhaupt zur Verfügung zu stellen, wird dieses Attribut in der Seitenübersicht ergänzt. Intellisense bietet dies nicht in der Liste an.

```
<siteMapNode url="02/portal.aspx" title="WebParts" description="externe Portal Seite ohne Masterseite"
target="_blank" ></siteMapNode>
```

Im Menü-Steuerelement wird dieses im *MenuItemDataBound* Ereignis ausgewertet. Dazu wird das *DataItem* in ein Objekt vom Typ *SiteMapNode* umgewandelt und darin das Attribut *target* ausgelesen. Der Parameter *e* der Ereignisfunktion enthält eine Referenz auf das *Item* des Menüeintrags. Darin befindet sich wiederum das HTM-Attribut *Target*.

```
Protected Sub MenuItemDataBound(ByVal sender As Object,
                                ByVal e As System.Web.UI.WebControls.MenuEventArgs)
    e.Item.Target = CType(e.Item.DataItem, SiteMapNode)("target")
End Sub
```

**Listing 2.15** Der Menüpunkt wird zur Laufzeit verändert

## TreeView

Das *TreeView*-Steuerelement befindet sich ebenfalls im Bereich *Navigation* der Werkzeugleiste. Trotzdem kann und wird dieses Steuerelement sehr häufig auch für andere Zwecke als Navigation verwendet. Durch die Baumstruktur ist es gut geeignet zur Visualisierung jeder Art von hierarchischen Daten. So ist es auch nicht verwunderlich das in ASP.NET 1.x die Frage nach einem *TreeView*-Steuerelement eine der häufigsten ist (bzw. war).

Mit dem *TreeView*-Steuerelement lassen sich XML-Daten oder relationale Daten binden und anzeigen. Dazu verwenden Sie am besten eines der *DataSource*-Steuerelemente wie z.B. das *SiteMapDataSource*-Steuerelement. Alternativ können die Knoten auch manuell per Deklaration erzeugt werden.

### HINWEIS

Die WYSIWYG-Darstellung mit den Echtdaten funktioniert leider nur bei fest deklarierten Knoten.

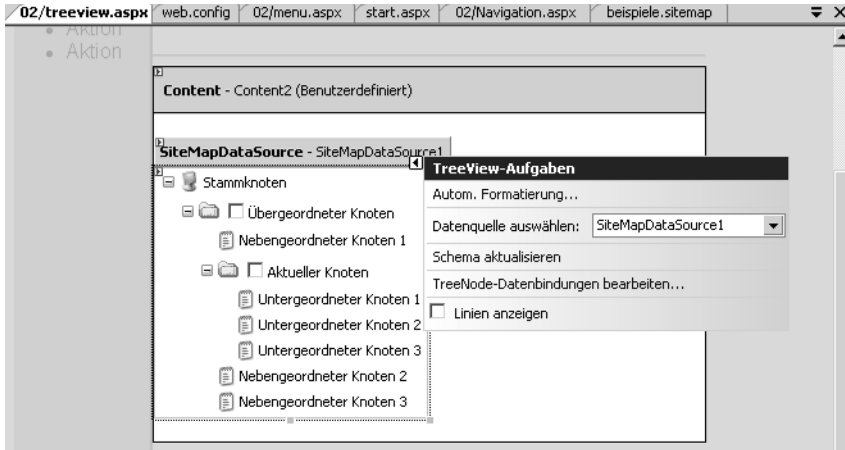


Abbildung 2.16 Ein TreeView Steuerelement wird an die Seitenübersicht gebunden

Die einzelnen Nodes können als normaler Text oder als Hyperlink angezeigt werden. Je nach Browser funktioniert das Auf- und Zuklappen direkt am Client ohne Postback. Sogar Checkboxes lassen sich mit dem Attribut *ShowCheckBoxes* in die Darstellung einbinden. Aus zahlreichen vorhandenen Design-Templates kann schnell das passende ausgewählt werden.

```
<asp:TreeView ID="TreeView1" Runat="server" ImageSet="XPFileExplorer"
  NodeIndent="15" ShowCheckBoxes="Parent" OnSelectedNodeChanged="TreeView1_SelectedNodeChanged"
  DataSourceID="SiteMapDataSource1">
  <SelectedNodeStyle BackColor="#B5B5B5"></SelectedNodeStyle>
  <NodeStyle VerticalPadding="2px" Font-Names="Tahoma" Font-Size="8pt" HorizontalPadding="2px"
    ForeColor="Black"></NodeStyle>
  <HoverNodeStyle Font-Underline="True" ForeColor="#6666AA"></HoverNodeStyle>
</asp:TreeView>
```

Listing 2.16 Ein TreeView Steuerelement in der HTML Ansicht

Und schon sind Sie stolzer Besitzer eines Baum-Menüs im Windows XP-Stil.



Abbildung 2.17 Browser-Darstellung eines Menüs im XP-Stil

Bei großen Datenmengen macht es Sinn, die Daten erst zu laden, wenn der entsprechende Zweig vom Benutzer geöffnet wurde. Das passende Ereignis dazu ist *TreeNodePopulate*. Über den Parameter *e* kann dann eine Programmentscheidung getroffen werden, welcher Wert im gewählten Knoten enthalten ist. Das folgende eher konzeptionelle Beispiel erläutert dies:

```

Sub TreeView1_TreeNodePopulate(ByVal sender As Object, ByVal e As TreeNodeEventArgs)
    If e.Node.ChildNodes.Count = 0 Then
        Select Case e.Node.Dept
            Case 0
                Fuelle(e.Node)
            Case 1
                Fuelle(e.Node)
        End Select
    End If
End Sub

```

**Listing 2.17** Ein Node wurde vom Benutzer selektiert

In der Hilfsfunktion *Fuelle* wird der Baum um Subtrees erweitert. Für jeden neuen Datensatz wird ein neuer *TreeNode* erzeugt. Dieser wird an den per Referenz übergebenen *node* angehängt. Es kann auch noch definiert werden, ob der Node aufgeklappt (*Expand*) sein soll und überhaupt aufklappbar (*PopulateOnDemand*) sein soll.

```

Sub Fuelle(ByVal node As TreeNode)
    ...
    For Each ...
        Dim NewNode As TreeNode = New TreeNode(Key, Value)
        NewNode.PopulateOnDemand = True
        NewNode.SelectAction = TreeNodeSelectAction.Expand
        NewNode.Text= ...
        NewNode.Value= ...
        node.ChildNodes.Add(NewNode)
    Next
End Sub

```

**Listing 2.18** Zur Laufzeit werden neue Nodes im Baum erzeugt und hinzugefügt

Es existieren eine Menge Layout-Templates und Anwendungsbeispiele für das *TreeView*-Steuerelement – zu viele, um sie alle in diesem Buch vorzustellen.

## Mehrsprachige Websites

Webanwendungen können theoretisch von jedem Internetanschluss der Welt aus genutzt werden. Dabei stellen sich jede Menge Fragen: Wie versorgt man die Benutzer, die über verschiedene Endgeräte, Zeichensätze, Landessprachen, Zahlenschemas, Gesetze usw. verfügen, mit der jeweils passenden Webseite? Sicher sind Sie schon mal über nicht passende Datumsformate gestolpert. Ein typischer Effekt zeigt sich spätestens am 13. eines Monats. Ob die Reihenfolge Monat/Tag oder Tag/Monat ist, ist dann durchaus von Bedeutung. Zwar denken die Entwickler meist daran, dies im Code zu berücksichtigen, ignorieren aber unterschiedliche Länderschemata auf Servern und Clients.

### Globalisierung

Mit der Globalisierung von Webanwendungen wird genau dieses Problem berücksichtigt, nämlich unterschiedliche Kulturen abzudecken. Die beiden wichtigen Eigenschaften dabei sind *UICulture* und *Culture*. Die Einstellung des Landesschemas beeinflusst die Webanwendung z.B. bei der Formatierung von Datums- oder Währungswerten. *UICulture* wird verwendet um die passenden Ressourcen zu laden. Sie wird durch die Ein-

stellungen des Benutzers am Browser gesetzt, genauer gesagt im http-Header bei jedem Request mit versandt. Man kann aber programmatisch diese Werte auch überschreiben.

Die möglichen *Culture*-Werte sind vorgegeben. Beispielhaft als gültige Einstellungen möchte ich hier nur *de* oder *en-US* erwähnen. Der erste Teil definiert die Sprache und der zweite Teil das Landesschema. Innerhalb einer Sprache kann es ja durchaus z.B. unterschiedliche Währungen geben. Die Liste der Culture-Schemas findet man z.B. in den Spracheinstellungen des Browsers.

Wie in ASP.NET üblich können Einstellungen wie die Culture-Infos in der *Page*-Deklaration der Seite oder auch für die ganze Anwendung per *web.config* oder dynamisch zur Laufzeit gesetzt werden. Die folgende *Page*-Deklaration setzt die Sprache und Culture der Seite auf Englisch.

```
<%@ Page Language="VB" MasterPageFile="~/master.master" Title="Untitled Page" Culture="en-US" UICulture="en"%>
```

**Listing 2.19** Setzen der Culture in der Page-Deklaration

Setzt man *UICulture* auf den Wert *auto*, wird die favorisierte Sprache des Browser benutzt. Dies kann vom Benutzer eingestellt werden.

Eine weitere Möglichkeit besteht über einen Eintrag in der *web.config* im Bereich *globalization*, der für die gesamte Anwendung gültig ist.

```
<globalization
  culture="de-DE"
  uiCulture="de"
/>
```

**Listing 2.20** Setzen der Culture-Infos für die gesamte Anwendung in der *web.config*

Wenn die Culture-Info dynamisch also im Code gesetzt werden soll, müssen zwei Namensräume eingebunden werden, nämlich *Threading* und *Globalization*. So kann spezifisch pro Thread das Landesschema gesetzt werden.

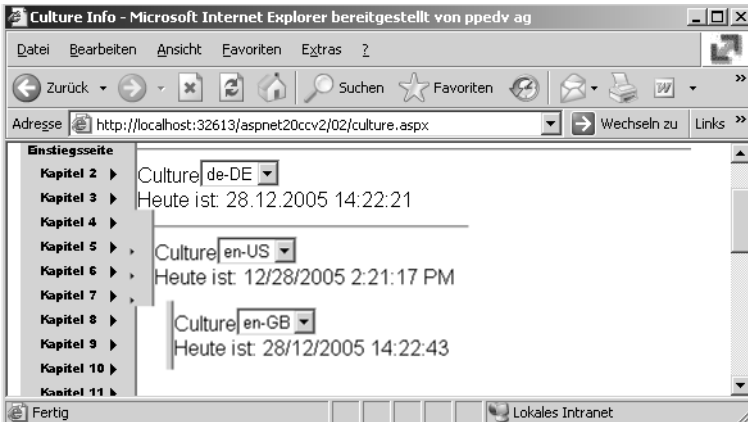
```
<%@ Import Namespace="System.Threading" %>
<%@ Import Namespace="System.Globalization" %>
<script runat="server">
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
  Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(DropDownList1.SelectedValue)
  Thread.CurrentThread.CurrentUICulture = New CultureInfo(DropDownList1.SelectedValue)
  ...
End Sub
```

**Listing 2.21** Zur Laufzeit die Culture des aktuellen Threads setzen

Dies ist z.B. sinnvoll, wenn ein Datenbankzugriff mit einem bestimmten Landesschema erfolgen muss, um die Datentypen passend zu bekommen.

Für das folgende Beispiel wird eine Dropdown-Box verwendet, die mit den Culture-Werten gefüllt wird. Beim Auswählen eines Eintrages wird dann im *Thread.CurrentThread* die neue Culture gesetzt. Die Datumsanzeige erfolgt per Label, dem einfach *Date.Now.ToString* zugewiesen wird.





**Abbildung 2.18** Dynamisch geänderte Culture erzeugt unterschiedliche Datumsformate

## Sprachressourcen

In der Windows-Entwicklung ist die Verwendung von Ressourcen zum Verwalten von mehrsprachigen Informationen seit langem üblich. Dabei werden Zeichenketten und auch Bilder in Resource-Dateien (.resx) unabhängig vom eigentlichen Code abgelegt und verwaltet. Mit ASP.NET und dem Resource Manager wurde auch dieses Feature für den Web-Entwickler verfügbar. Leider war die Umsetzung viel zu kompliziert.

Die eigentliche Ressourceninformation wird in XML-basierten Dateien mit der Endung *.Resource* gespeichert und muss erst kompiliert werden.

## Ressourcen ohne Reue

Das Schönste an der Ressourcenverwaltung in ASP.NET 2.0 ist *nicht*, dass man keinen Code benötigt. Sie haben das vielleicht schon erwartet. Das Schönste ist, dass man die Website entwickeln kann, ohne an Funktionalitäten zur Berücksichtigung von Mehrsprachigkeit zu denken.

Im Menü von Visual Studio 2005 gibt es in der Entwurfsansicht unter *Extras/Lokale Ressource generieren* einen Assistenten, der ohne weiteres Nachfragen die Ressourcendatei im *app\_LocalResources*-Unterverzeichnis erzeugt. Die Datei wird lediglich mit der zusätzlichen Erweiterung *.resx* versehen. Dabei werden alle relevanten Texte aus der ASPX-Seite in die Ressourcendatei geschrieben. Dieses Verzeichnis kann auch in Unterverzeichnissen existieren bzw. muss es sogar, wenn die ASPX-Seite sich in diesem befindet.

Diese Funktion fehlt in Visual Web Developer 2005 Express. Die Verwendung von Ressourcen ist trotzdem möglich. Lediglich die *.resx*-Dateien müssen von Hand erzeugt werden. Dafür legen Sie das Verzeichnis *App\_LocalResources* an. Visual Web Developer bietet hierzu noch einen entsprechenden Menüpunkt *ASP.NET-Ordner hinzufügen* im Menü *Website*. Im Kontextmenü dieses Ordners kann dann die Ressourcendatei mittels dem Menüpunkt *neues Element hinzufügen- Ressourcendatei* erzeugt werden.

Die Ressourcendatei kann dann in jeder Version von Visual Studio 2005 ganz normal per Doppelklick geöffnet und auch editiert werden. Dabei werden die Ressourcen selbst in einer Art *Name-/Wert*-Liste verwaltet. Für jeden verwendeten Namen muss hier dann auch ein Eintrag vorhanden sein bzw. mit Visual Web Developer per Hand erzeugt werden.

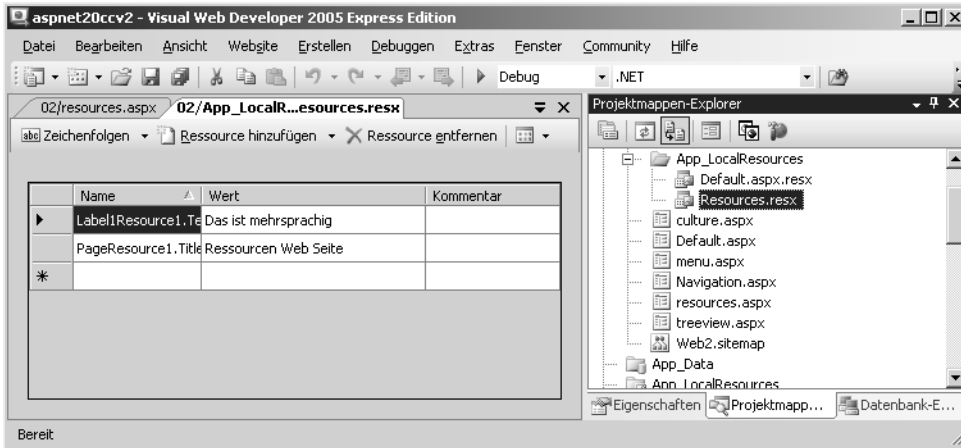


Abbildung 2.19 Die Ressourcen werden direkt in der Entwicklungsumgebung erfasst

Je nach Art des Steuerelements werden unterschiedliche Attribute per Ressource gesteuert. Ein *Image* besitzt zum Beispiel das Attribut *AlternateText* in den Ressourcendaten. Wenn Sie eigene Steuerelemente verwenden, die auch mehrsprachig sein sollen, können Sie das per Deklaration definieren.

```
<System.ComponentModel.Localizable(True)> _
Public Property Aufruf() As String
    Get
    End Get
    Set(ByVal value As String)
    End Set
End Property
```

Listing 2.22 Custom-Attribut für Lokalisierung

Die XML-Struktur der Ressourcendateien ist allerdings so komplex, dass ein vollständig manuelles Erzeugen wenig sinnvoll erscheint. Die eigentlichen Daten werden in einem *Data*-Element über Attribute abgelegt.

```
<data name="Arbeitstitel"><value>ASP.NET 2.0 Schnelleinstieg</value></data>
```

Die Zuordnung der Ressourcen Information erfolgt dann in der ASPX-Seite im Steuerelement mittels einem zusätzlichen Meta-Attribut *resourcekey*.

```
<asp:Image ID="Image1" meta:resourcekey="ImageResource1" runat="server" /></asp:Content>
```

Wenn das Steuerelement nicht lokalisiert werden soll, kann dies auch per Meta-Attribut erreicht werden. Dazu setzen Sie *meta:localize* auf *false*:

```
<asp:Image ID="Image1" meta:localize="False" runat="server" ImageUrl="bild.jpg" />
```

Selbst ganze Bereiche der Webseite lassen sich lokalisieren. So könnten z.B. Vertragsbestimmungen sprachspezifisch ausgegeben werden. Dazu wird das neue *localize* Steuerelement verwendet:

```
<asp:localize runat="Server">Dieser Text wird mehrsprachig</localize>
```

Ein weiterer häufiger Anwendungsfall ist das Einbinden von Grafiken. Dazu werden die Referenzen über die *ImageUrl* in den Ressourcendateien gepflegt.

**HINWEIS** Der Assistent *Generate Local Resource* kann auch mehrmals ausgeführt werden. Allerdings werden dann die Änderungen in den Ressourcendateien überschrieben. Jedenfalls waren das meine bisherigen Erfahrungen.

Änderungen an den Texten müssen dann ausschließlich in der Ressourcendatei direkt vorgenommen werden. Die Textänderungen im Eigenschaftsdialog des Steuerelements werden ignoriert.

## Spracherkennung

Noch sind wir nicht so weit, dass unsere Browser unsere gesprochene Sprache verstehen, aber zumindest sollte die Website wissen, welche Sprache wir gerne lesen würden. Dies funktioniert auch ganz automatisch. Im Browser sind die entsprechenden Einstellungen vorhanden, die im http-Header zum Server versandt werden. Um einer ASP.NET-Seite diesen Automatismus vorzugeben, werden in der *Page*-Deklaration die *Culture*-Werte auf *auto* gesetzt.

```
<%@ Page Language="VB" MasterPageFile="~/master2.master" UICulture="auto" Culture="auto"%>
```

Genauso einfach lässt sich eine Seite unter völliger Ignoranz der Browsereinstellungen auf eine spezifische Sprache einstellen.

```
<%@ Page Language="VB" MasterPageFile="~/all.master" Culture="en-US" UICulture="en"
Title="Ressourcen Test" meta:resourcekey="PageResource1" %>
```

Die Verwendung des Meta-Attributes wird auch als *implizite Lokalisierung* bezeichnet.

## ASP.NET-Ausdrücke

Eine der ganz besonders nützlichen und einfachen Neuerungen sind ASP.NET-Ausdrücke (Expressions). Damit wird über eine *<%\$*-Syntax ein Zugriff auf die Daten der Datei *web.config* – wie Connectionstrings – oder der Ressourcendateien möglich. Das Schlüsselwort *Resources* verweist auf eine Ressourcendatei – im folgenden Beispiel mit dem Namen *dictionary.resx*. Der letzte Wert ist der Schlüssel (*Key*) zum eigentlichen Wert. Diese Methode wird *explizite Lokalisierung* genannt.

**HINWEIS** Die Ressourcendatei muss hierzu im *app\_GlobalResources*-Verzeichnis liegen!

Legen Sie also eine Ressourcendatei mit dem Namen *dictionary.resx* im *app\_GlobalResources*-Verzeichnis unterhalb des Anwendungsstammverzeichnisses an. Der erste Wert nach dem Schlüsselwort *Resources* beschreibt den Namen der Ressource und der zweite Wert den Key:

```
<asp:Literal ID="Label1" Runat="server" Text="<% $Resources:dictionary,Haus%>"></asp:Label>
```

Auf diese Weise erzeugte Ressourcen gelten für die gesamte Anwendung.

Die Ressourcen-Werte lassen sich sogar per Code unter zu Hilfenahme von Intellisense ansprechen. Dazu bietet das *resources* Objekt die *Dictionary* Klasse.

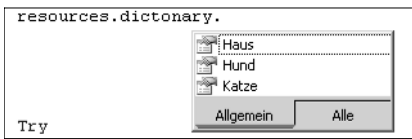


Abbildung 2.20 Resource Keys definiert in *App\_GlobalResources*

Ein ASP.NET Ausdruck kann an jede Eigenschaft eines Web- oder HTML-Server-Steuerelements gebunden werden. Allerdings ist es nicht zulässig, diese direkt im HTML-Code zu verwenden.

Für Liebhaber des interaktiven Designs stellt Visual Studio in den *Properties* eines Steuerelements über die Eigenschaft *Expressions* einen Dialog bereit.

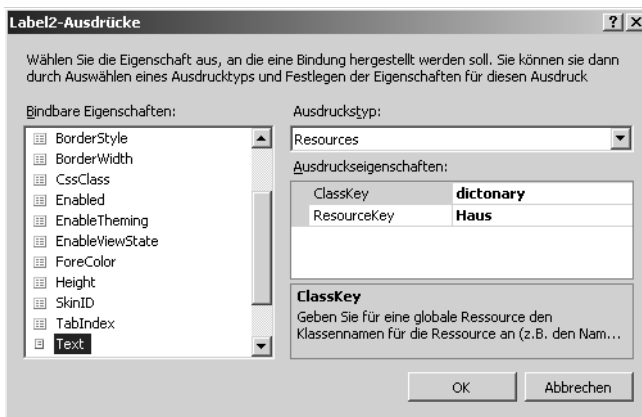


Abbildung 2.21 Visuelles Bearbeiten der ASP.NET-Ausdrücke

Selbst kompletter HTML-Code lässt sich so mit ASP.NET-Ausdrücken von externen Quellen einpflegen. Dazu wurde ein neues HTML-Element namens *localize* geschaffen. Dieses Element wird in die ASPX-Seite eingebaut.

Genau wie bisher beschrieben, erfolgt der Zugriff dann über das Schlüsselwort *resources* und den *Key*-Namen des Eintrags.

```
<localize runat="server" text="<% $resources: HTMLContent %>">
  Placeholder text
</localize>
```

Listing 2.23 HTML-Sequenzen können aus Ressourdateien geladen werden

## Pro Sprache eine Datei

Nun haben wir zwar eine Ressourcendatei, aber wie erstellt man die Versionen für anderen Sprachen? Im nächsten Schritt kopieren Sie dazu die komplette Ressourcendatei in eine neue Datei mit dem Länderkürzel als Zusatz – z.B. *resources.en.resx* für eine englische Ressourcenbeschreibung. Pro Sprache benötigen Sie eine Datei. Die Datei ohne Zusatz ist die Standarddatei, die verwendet wird, wenn keine passende Sprachressource vorhanden ist. Die Auswahl erfolgt nur nach den Einstellungen des aufrufenden Browsers, wenn *UICulture* in der Seite auf *auto* gestellt ist.

**HINWEIS** Die Spracheinstellungen im Internet Explorer ändern Sie im Menü *Extras/Internetoptionen*. Im Dialogfeld wählen Sie im Reiter *Allgemein* den Button *Sprachen*. Fügen Sie *en* hinzu, und ändern Sie die Reihenfolge.

Jede ASPX-Seite besitzt ihre eigene Ressourcendatei im Verzeichnis *App\_LocalResources* – deshalb müssen Sie auch nicht auf Namensüberschneidungen achten.

## Designvorlagen für Seiten

Frontpage-Benutzer kennen diese Funktionalität schon lange. Designvorlagen erlauben es, zu jeder Zeit das komplette Design der Anwendung mit nur einem Knopfdruck zu ändern. Buttons werden so blau statt grau und der Hintergrund braun statt weiß. Dies ist seit Frontpage unter dem Namen *Themes* oder auch *Designs* bekannt.

Sie können solche Design-Vorlagen selbst erstellen oder auch von der Microsoft Webseite herunterladen. Es ist auch zu erwarten, dass per Web-Download hier in Zukunft eine größere Auswahl verfügbar sein wird.

Damit ist die Funktionalität mit CSS (*Cascading Style Sheets*) vergleichbar, Themes bieten aber mehr Möglichkeiten. CSS können auch Bestandteil von Themes sein. Themes können aber im Gegensatz zu CSS auch Grafiken beinhalten. Ein weiterer wichtiger Unterschied ist, dass die erweiterten Attribute der Webserver-Steuererelemente gesetzt werden können. So hat das Calendar-Steuererelement eine Eigenschaft *TodayDayStyle.BackColor*, die per Theme gesetzt werden kann.

### Erzeugen von Themes

Themes müssen im *App\_Themes*-Unterverzeichnis abgelegt werden. Das *Themes*-Verzeichnis muss im Projektmappen-Explorer manuell angelegt werden, ist also nicht automatisch vorhanden. Pro Theme wird ein weiteres Unterverzeichnis unterhalb von *App\_Themes* angelegt. Der Name des Verzeichnisses ist gleichzeitig der Name des Themes. In unserem Beispiel erzeugen wir das Theme *Devtrain*.

Themes wiederum beinhaltet eine oder mehrere Designdateien (Skins). In der Designdatei wird das individuelle Design eines Steuererelements festgelegt. Designdateien haben die Erweiterung *.skin*. Es ist möglich, aber nicht erforderlich, pro Steuererelement eine Skin-Datei anzulegen – z.B. *Button.skin*, um das Design von Buttons zu definieren.

Wenn in der Anwendung unterschiedliche Button-Designs benötigt werden, kann diese Unterscheidung über das Attribut *SkinID* getroffen werden. Das heißt: In der Datei *Button.Skin* werden zwei Varianten mit unterschiedlicher *SkinID* für den Button angelegt.

Zum Erzeugen einer Designdatei wählen Sie aus dem Kontextmenü des Themes *Verzeichnisse* den Punkt *Neues Element hinzufügen*. Aus den vorhandenen Vorlagen wählen Sie *Designdatei* aus. Geben Sie der Datei einen Namen, und speichern Sie die komplette Datei in Ihrem *Design*-Verzeichnis, also z.B. *Devtrain.skin* im Unterverzeichnis *Devtrain* unterhalb von *App\_Themes*.

**HINWEIS** Im Editor für die Designdatei (Skin-Datei) ist leider keine Unterstützung durch IntelliSense vorhanden. Ich erstelle deshalb die Skins immer in einer normalen ASPX-Seite, um auch die visuelle Kontrolle zu haben und kopiere dann das Steuererelement in die Skin-Datei. Achten Sie darauf, dabei das Attribut *ID* zu entfernen.

Anschließend werden für jedes Webserver-Steuerelement die Eigenschaften festgelegt:

```
<asp:Button runat="server" borderstyle="Solid" borderwidth="1px" bordercolor="Black" bgcolor="orange" />
<asp:Button runat="server" SkinID="alternativ" borderstyle="Solid" borderwidth="2px" bordercolor="Black"
  bgcolor="gray" />
<asp:TextBox runat="server" ForeColor="darkgray" borderwidth="1px" Font-Size="11pt" Font-Names="Verdana" />
```

**Listing 2.24** Design für zwei Buttons und eine TextBox

Die Skin-Deklarationen berücksichtigen die Groß-/Kleinschreibung und müssen dementsprechend identisch mit den Originalnamen des jeweiligen Web-Steuerelements sein.

## Einbinden des Designs

Jetzt muss die Designvorlage noch eingebunden werden. Dazu gibt es drei Möglichkeiten: Per *Page*-Deklaration, mithilfe der *web.config* oder dynamisch zur Laufzeit. In diesem Beispiel wird in der *Page*-Direktive per *Theme*-Attribut das Design zugewiesen. IntelliSense erkennt die von Ihnen erzeugten und vorhandenen Designs und zeigt diese als Dropdown-Liste zur Auswahl im Editor an.



**Abbildung 2.22** Einer Seite wird ein Design zugewiesen

In der Webseite werden dann die Webserver-Steuerelemente wie gewohnt platziert. Wenn für ein Steuerelement ein Design definiert ist, kommt dieses automatisch zur Laufzeit zur Anwendung. Nur Steuerelemente, bei denen die Eigenschaft *SkinID* gesetzt ist, erhalten das Design, das im Theme für diese *SkinID* vorgesehen ist.

```
<%@ Page Language="VB" MasterPageFile="-/master2.master" Title="Themes" Theme="DevTrain" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
  <asp:Button ID="Button1" Runat="server" Text="Button" />
  <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  <asp:Button ID="Button2" Runat="server" Text="Button" SkinID="alternativ" />
</asp:Content>
```

**Listing 2.25** Ein Button erhält per SkinID eine anderes Design

Falls der *SkinID*-Wert nicht definiert ist, kommt das Standarddesign zur Anwendung. Um ein Steuerelement explizit im Standarddesign zu verwenden, muss die Eigenschaft *EnableTheming="False"* gesetzt sein.



**Abbildung 2.23** Die Anzeige von »skinned«-Buttons im Browser

Die einfachste Methode einer Webanwendung nachträglich ein Design zuzuordnen, ist die *web.config*:

```
<configuration>
  <system.web>
    <pages enableViewState="false" Theme="DevTrain"></pages>
  </system.web>
</configuration>
```

**Listing 2.26** Ein Design für die ganze Applikation

---

**HINWEIS** Mithilfe des `<Location>`-Elements der `web.config` lässt sich der Gültigkeitsbereich auf Dateien oder Verzeichnisse weiter einschränken.

---

Wenn Sie die Zuweisung eines Themes zur Seite oder zu einzelnen Web-Steuer-elementen zur Laufzeit per Programmcode vornehmen wollen, müssen Sie darauf achten, dass dies in der Funktion `Page_PreInit` passiert.

Eine CSS-Datei die sich im ausgewählten Theme-Verzeichnis befindet wird automatisch in die Seite eingebunden, so als ob sie mit `<link` verlinkt wäre. Damit kann man Webseiten nachträgliche CSS Stil Dateien zuweisen, ohne diese Seiten zu verändern.

## Stylesheettheme

Mit Skin-Defintionen lassen sich alle Attribute eines Webserver-Steuer-elementes überschreiben, die als *Themeable* gekennzeichnet sind. Das kann z.B. auch *Text* sein, um z.B. den Text auf einem Suchbutton vorzubelegen.

Möchten Sie jedoch den expliziten Einstellungen der Website den Vorrang geben und dennoch Stylesheets verwenden, können Sie dies mithilfe des Attributes *StylesheetTheme* erreichen. In diesem Fall werden die Einstellungen in der Skin-Datei nicht berücksichtigt, es wird lediglich das Stylesheet verwendet.

## WebParts

Webanwendungen erleben eine Wiedergeburt, dank neu verstandener Benutzerfreundlichkeit. Technologisch sind das Technologien wie beispielsweise AJAX, die unter dem Kunstbegriff Web 2.0 zusammengefasst werden. Dazu gehören auch voll vom Benutzer zu personalisierende Webseiten. Herausragendes Merkmal solcher Seiten sind umfangreiche Inhalte, die sich der Besucher selbst zusammenstellen kann. Nicht nur die Inhalte sondern auch Stil und Design müssen den eigenen Wünschen entsprechend veränderbar sein.

Dazu war bisher aufwändige Codierung mit JScript und DHMTL notwendig, um Bereiche programmatisch ein- und auszublenden. Natürlich sollten die veränderten Einstellungen bis zum nächsten Besuch und auch darüber hinaus erhalten bleiben. In öffentlich zugänglichen Websites hat sich dies eher nicht durchgesetzt und ist heute kaum anzutreffen. Microsoft versucht hier mit Live.com einen entsprechenden Prototyp zu betreiben. Meist sind es aber wenn überhaupt, nur kleine Teile oder Unterseiten, die individualisierbar sind. In Intranet-Anwendungen dagegen findet sich diese Technologie des Öfteren, vermutlich hauptsächlich deshalb, weil viele Intranets mit einer fertigen Content-Management-Software (wie z.B. dem SharePoint Portal Server) aufgesetzt sind. Gerade der SharePoint Portal Server lohnt näherer Betrachtung, da die dortige Portlet-Technik jetzt auch in ASP.NET enthalten ist.

---

**HINWEIS** Beachten Sie, dass nicht jede Version von SharePoint mit ASP.NET 2.0 WebParts kompatibel ist.

---

Mit den WebParts von ASP.NET 2.0 ist es möglich, solche Portalseiten mit weniger Aufwand zu erstellen.

Im Gegensatz zu vielen anderen neuen ASP.NET 2.0-Technologien muss die WebParts-Technologie bereits bei der Planung berücksichtigt werden. Bestehende Webseiten auf WebParts umzurüsten ist sehr aufwändig. Wenn Sie in Ihrer Anwendung intensiv Webbenutzer-Steuerelemente verwendet haben, wird es jedoch ein wenig leichter.

Webbenutzer-Steuerelemente sind auch der Schlüssel zu WebParts. Zusammenhängende Funktionalität und Informationen werden in einem Webbenutzer-Steuerelement zusammengeführt.

Alternativ lassen sich auch echte WebPart-Steuerelemente erstellen. Dazu muss das Interface *IWebPart* eingebunden werden. Das erfordert dann die Implementierung einiger öffentlicher Eigenschaften wie z.B. *Title*. Dies soll aber nicht Thema dieses Buches sein, hier werden nur die bereits vorhandenen WebPart Steuerelemente besprochen.

WebParts verwenden die ASP.NET-Personalisierungsfunktion, die erst in einem späteren Kapitel im Detail erläutert wird. So viel vorweg: Die Personalisierung ist automatisch aktiviert. Allerdings ist erst mit erfolgter Benutzeranmeldung eine echte Personalisierung möglich. Andernfalls gelten alle Einstellungen eben für alle Benutzer. In jedem Falle wird dazu eine Datenbank benötigt. Die Standard-Konfiguration geht hierbei von einer installierten SQL Server Express-Edition aus. Beim ersten Aufruf einer personalisierten Seite wird dann im Verzeichnis *App\_Data* eine Datei *aspnetdb.mdf* erzeugt. Zu SQL Express gibt es im Kapitel 12 zusätzliche Informationen, Personalisierung wird im Kapitel 6 genauer beschrieben.

## WebPart-Manager

Da der Einstieg leicht unübersichtlich werden kann, wird hier auf die Verwendung einer Masterseite verzichtet. Ihre erste Aktivität ist es, ein *WebPartManager*-Steuerelement auf die neue ASPX-Seite zu ziehen. Ohne dieses Steuerelement geht nichts und es muss ganz oben stehen. Es übernimmt die zentrale Management-Funktion. Sie brauchen daran auch keine weiteren Einstellungen vorzunehmen. Das Steuerelement wird im Browser zur Laufzeit nicht angezeigt.

```
<asp:WebPartManager ID="WebPartManager1" runat="server">
</asp:WebPartManager>
```

Allerdings besitzt das *WebPartManager*-Steuerelement einige zusätzliche Attribute, die durchaus von Interesse sind:

Attribut	Verwendung
<i>CloseProviderWarning</i>	Ausgegebener Meldungstext, wenn ein WebPart geschlossen wird, der Daten für andere WebParts zu Verfügung stellt. Es existiert ein englischer Vorgabetext, wie auch für die folgenden Warnungen.
<i>DeleteWarning</i>	Die Meldung, die ausgegeben werden soll, wenn ein WebPart geschlossen wird.
<i>ExportSensitiveDataWarning</i>	Ein Sicherheitshinweis, wenn Attribute eines WebParts vom Benutzer exportiert werden.
<i>EnableClientScript</i>	Boolescher Wert mit dem Client Script deaktiviert werden kann.
<i>EnableViewState</i>	Boolescher Wert.
<i>Personalization-Enabled</i>	Boolescher Wert.

**Tabelle 2.2** Attribute des WebPart Managers



Attribut	Verwendung
<i>Personalization-InitialScope</i>	Mögliche Werte: Shared und User.
<i>Personalization-ProviderName</i>	Name eines eventuell abweichenden Personalisierungsprovider.

**Tabelle 2.2** Attribute des WebPart Managers (Fortsetzung)

Wenn eine Masterseite zum Einsatz kommt und das *WebPartmanager*-Steuerelement sich auf dieser befindet, muss auf der Inhaltsseite zusätzlich das *ProxyWebPartmanager*-Steuerelement platziert werden.

Der WebPart-Manager übernimmt später die Steuerung der WebParts. Wenn vom Benutzer das Layout geändert werden soll, muss der Modus des *WebPartManager*-Steuerelements gewechselt werden. Es wird die Eigenschaft *DisplayMode* verwendet, um in einen der fünf Standard-Modi zu wechseln.

```
WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode
```

Je nach Modus stehen dem Benutzer dann verschiedene Möglichkeiten zur Verfügung. Diese können Interaktiv wie Drag&Drop oder durch zusätzliche Menüpunkte in den WebParts erscheinen. Die genaue Anwendung wird etwas später am kompletten Beispiel erläutert.

Außerdem können über Unterelemente noch statische Verbindungen zwischen WebParts definiert werden. Diese als *Connection* bezeichnete Eigenschaft ist nötig, um Daten zwischen zwei WebParts auszutauschen. Davon gibt es die statische und dynamische Variante, wobei letztere durch den Benutzer definiert wird. Weiterhin wird auch die Personalisierung durch zusätzliche Attribute gesteuert.

```
<asp:WebPartManager ID="WebPartManager1" runat="server">
  <Personalization Enabled="true" InitialScope="Shared" ProviderName="" />
  <StaticConnections><asp:Connection ID="dsa"></asp:Connection> </StaticConnections>
</asp:WebPartManager>
```

**Listing 2.27** Erweiterte Konfiguration des WebPart Managers

## WebPartZone

Der nächste unverzichtbare Teil im WebParts-Konzept sind die Zonen. Diese dienen später als Gruppierungsbereiche. Zonen sind Layouthilfen für den Benutzer. In einer Zone befinden sich dann die eigentlichen WebParts. WebParts selbst sind keine speziellen Steuerelemente, sondern können aus einem ASP.NET-Server-Steuerelement, einem Benutzer-Steuerelement oder einem manuell codierten (*GenericWebPart*) Steuerelement bestehen.

### Zonen definieren

Im folgenden Beispiel werden zwei Zonen mit der ID *wpzFinanzberichte* und *wpzInfoCenter* erzeugt. Um dem Designer die Arbeit zu erleichtern empfiehlt es sich mit HTML Tabellen zu arbeiten. Erzeugen Sie eine Tabelle mit einer Zeile und 3 Spalten. Ziehen Sie dann aus der Werkzeugleiste ein *WebPartZone*-Steuerelement in die erste Spalte und eines in die zweite Spalte. Ändern Sie die IDs. Den Kopftext, der erst in speziellen Modi angezeigt wird, können Sie per Attribut *HeaderText* setzen.

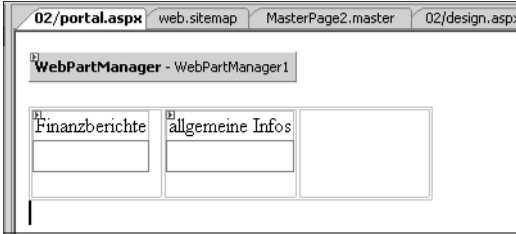


Abbildung 2.24 Zwei WebPart-Zonen dienen als Platzhalter

In jede Zone werden dann ein oder mehrere Webserver- oder Benutzer-Steuerelemente per Drag & Drop gezogen. Im folgenden Beispiel werden so fünf Benutzer-Steuerelemente mit unterschiedlichen Inhalten zu WebParts.

### TIPP

Umfangreiche Seiten mit vielen WebParts und Zonen sind schwer zu editieren. Man trifft oft einfach nicht das richtige Steuerelement. Hier hilft die SmartNavigation am unteren Rand des visuellen Editors von Visual Studio, wie in folgender Abbildung zu sehen:



Abbildung 2.25 Die SmartNavigation hilft bei der Selektion des gewünschten Elements

## Layout

Die Benutzer-Steuerelemente in der WebPartZone erhalten bereits in der Vorschau automatisch einen Rahmen mit einer Titelzeile. Der Standard-Titel ist *Untitled*. Bei einer voll implementierten Seite (wir stehen erst am Anfang) kann der Benutzer auch den Titeltext im Browser ändern. In der HTML- Ansicht finden sich die Steuerelemente dann im Bereich *ZoneTemplate*.

Wenn Sie in einer Zone z.B. den Titel ausblenden wollen, müssen Sie das Attribut *PartChromeType* setzen. IntelliSense schlägt in der HTML-Ansicht die möglichen Werte vor. In der WebPartZone wird dann das Attribut *PartChromeType="TitleAndBorder"* stehen.

Solche gemeinsamen Designelemente werden als *Chrome* bezeichnet und finden sich an verschiedenen Stellen. Natürlich können diese Eigenschaften auch im *Property*-Dialog bearbeitet werden.

Den Titel der einzelnen WebParts können Sie über das Attribut *Title* des *Benutzer-Steuerelements* setzen. Dies ist aktuell nur direkt im HTML-Code möglich. IntelliSense bietet dieses Attribut nur an, wenn es sich um ein WebPart mit *IWebPart*-Interface handelt.

```
<asp:WebPartZone ID="wpzFinanzBerichte" runat="server" HeaderText="Finanzberichte"
    PartChromeType="TitleAndBorder" >
  <ZoneTemplate >
    <uc2:umsatz ID="Umsatz1" runat="server" title="Umsatz" />
    <uc3:offeneAuftraege ID="OffeneAuftraege1" runat="server" title="offene Aufträge" />
  </ZoneTemplate>
</asp:WebPartZone>
```

Listing 2.28 Zwei WebParts in einer Zone

Auch einzelne Webserver-Steuerelemente können direkt in der Zone verwendet werden. So lässt sich innerhalb eines LiteralControl-Steuerelements HTML-Code einbetten. Wenn Sie längere Beschreibungen wie z.B. einen Hilfetext in ein WebPart bringen wollen, bietet sich das Panel-Steuerelement an. Dieses lässt sich in der Höhe begrenzen und unterstützt darüber hinaus Scrollbars.

```
<asp:Panel Runat="server" ID="Panel1" ScrollBars="Vertical" Height="100">...</asp:Panel>
```

## Stil festlegen

In der Standardausführung sehen so entworfene Webseiten noch etwas trist aus. Zonen können über das zugehörige Kontextmenü mit einer *automat. Formatierung* versehen werden. Dabei kann aus vier Style-Vorlagen ausgewählt werden.



Abbildung 2.26 WebPart-Zonen werden mit Stil-Vorlagen versehen

Die so erstellten Seitendesigns sind aber nicht besonders ausgefeilt. Es fehlt jede Art von Grafik, und die Beschriftungen sind auf Englisch.

Das kann man natürlich ändern. Die beiden Unterelemente *PartTitleStyle* und *PartStyle* können an die eigenen Bedürfnisse angepasst werden. Das erste Element dient der Titelgestaltung und das andere dem Inhalt. Daneben gibt es noch eine Reihe weiterer Attribute und Elemente, wie im folgenden Codebeispiel demonstriert wird.

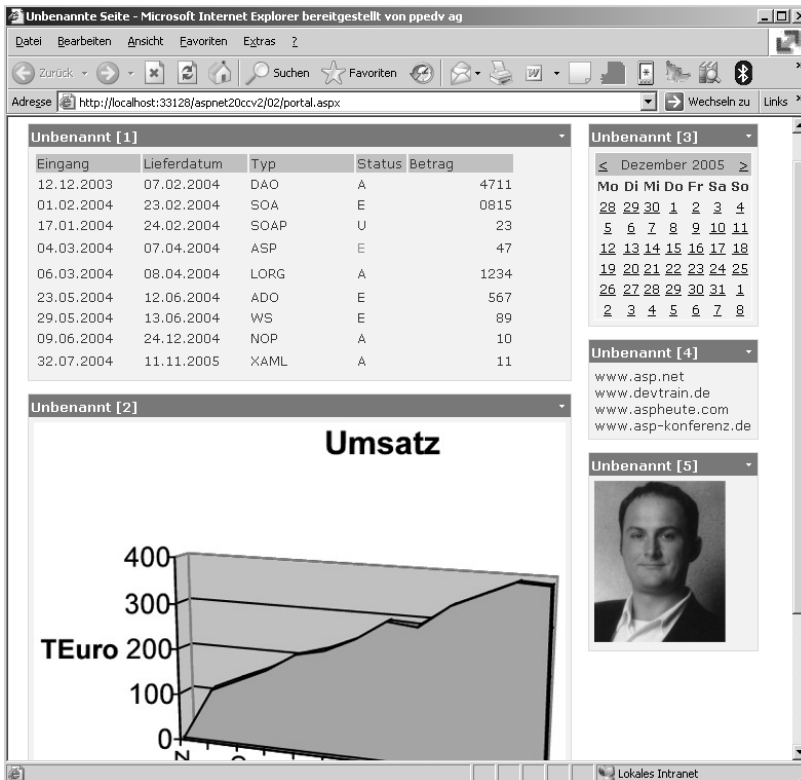
```

<asp:WebPartZone ID="wpzFinanzberichte" runat="server" Width="368px" BorderColor="#CCCCCC"
    Font-Names="Verdana" Padding="6" PartChromeType="TitleAndBorder">
<ZoneTemplate >
    <uc1:umsatz ID="Umsatz1" runat="server" title="Umsatzbericht" />
    <uc3:offeneAuftraege ID="OffeneAuftraege1" runat="server" title="Offene Aufträge"/>
</ZoneTemplate>
<PartChromeStyle BackColor="#EFF3FB" BorderColor="#D1DDF1" Font-Names="Verdana" ForeColor="#333333" />
<MenuLabelHoverStyle ForeColor="#D1DDF1" />
<EmptyZoneTextStyle Font-Size="0.8em" />
<MenuLabelStyle ForeColor="White" />
<MenuVerbHoverStyle BackColor="#EFF3FB" BorderColor="#CCCCCC" BorderStyle="Solid"
    BorderWidth="1px" ForeColor="#333333" />
<HeaderStyle Font-Size="0.7em" ForeColor="#CCCCCC" HorizontalAlign="Center" />
<MenuVerbStyle BorderColor="#507CD1" BorderStyle="Solid" BorderWidth="1px" ForeColor="White" />
<PartStyle Font-Size="0.8em" ForeColor="#333333" />
<TitleBarVerbStyle Font-Size="0.6em" Font-Underline="False" ForeColor="White" />
<MenuPopupStyle BackColor="#507CD1" BorderColor="#CCCCCC" BorderWidth="1px" Font-Names="Verdana"
    Font-Size="0.6em" />
<PartTitleStyle BackColor="#507CD1" Font-Bold="True" Font-Size="0.8em" ForeColor="White" />
</asp:WebPartZone>

```

**Listing 2.29** Eine einfach formatierte WebPart-Zone

Die ersten Erfolgserlebnisse zeigen sich in der Browser-Ansicht. Es ist alles vorhanden: Ein Rahmen, eine Titelleiste und der Inhalt.



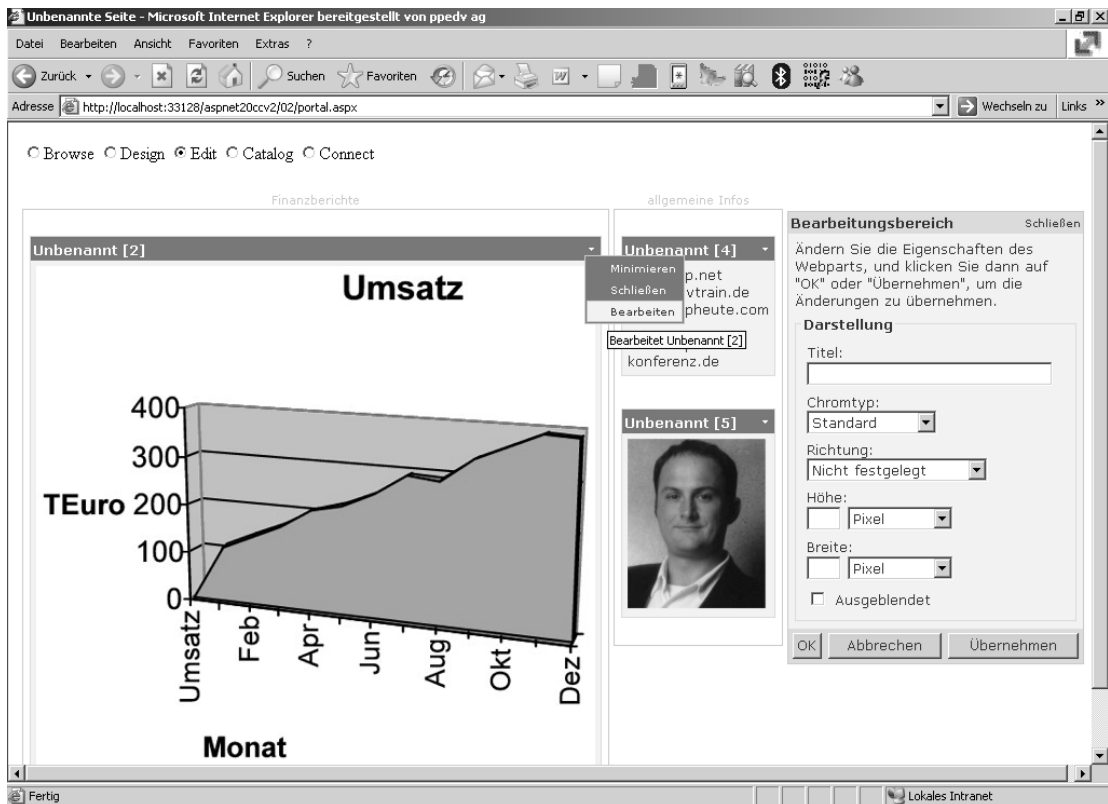
**Abbildung 2.27** Eine WebPart-Seite

Aus Platzgründen und auch wegen der Optik möchten Sie vielleicht Symbole statt Text im Titel haben. Es wird Sie nicht wirklich überraschen, dass in einem der tausend möglichen Attribute auch dafür das Passende dabei ist. Mit diesen Attributen können die Image-URL und auch der ToolTip-Text gesetzt werden. Hier werden die Icons für das Schließen, das Verkleinern und das Wiederherstellen gesetzt:

```
<asp:WebPartZone ID="wpzFinanzberichte" Runat="server"
  CloseVerb-ImageUrl="~\images\close.gif" CloseVerb-Description="Bereich schliessen"
  MinimizeVerb-ImageUrl="~\images\min.gif" MinimizeVerb-Description="Bereich verkleinern"
  RestoreVerb-ImageUrl="~\images\max.gif" RestoreVerb-Description="Wiederherstellen"
  DragHighlightColor="244, 198, 96">
```

**Listing 2.30** WebPartZone mit Icons versehen

Das gesamte Ergebnis kann sich im Browser durchaus sehen lassen. Zwei Zonen nebeneinander beherbergen die WebParts. Einzelne WebParts wie z.B. hier das WebPart mit dem Titel *Anruf von* können minimiert oder sogar ganz ausgeblendet werden.



**Abbildung 2.28** Das erste Portal

Die Titelleiste eines WebParts hat im rechten Bereich ein Kontextmenü, mit dem das WebPart geschlossen oder minimiert werden kann. Dies wird durch einen Klick auf den kleinen Pfeil geöffnet und zeigt eine Liste der aktuell möglichen Aktionen. Diese Aktionen werden auch als *Verbs* bezeichnet. Ein WebPart, das über das Verb *schließen* geschlossen wird, ist dauerhaft nicht mehr sichtbar.

Wenn Sie dann während der Testphase Teile Ihrer Webseite vermissen, können Sie die *PersonalizationAdministration* verwenden, um den Status wieder zurückzusetzen.

```
PersonalizationAdministration.ResetAllState(PersonalizationScope.User)
```

## Editieren von WebParts

Um dem Benutzer die Möglichkeit zu geben, selbst an das Design der Webseite Hand anzulegen, müssen Sie die Funktion den Modus zu wechseln selbst implementieren. Dabei muss jeder Modus einzeln gesetzt werden. Dazu kann z.B. ein *HyperlinkButton*-Steuerelement verwendet werden, in dessen *Click*-Ereignis die Eigenschaft *DisplayMode* des WebPart-Managers geändert wird.

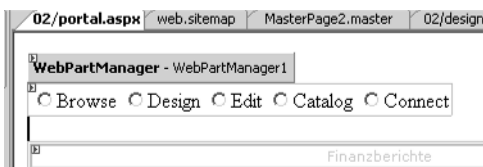
```
Webpartmanager1.DisplayMode = WebPartManager.EditDisplayMode
```

Die folgende Tabelle beschreibt die verschiedenen Ansichtsmodi.

DisplayMode	Verwendung
<i>BrowseDisplayMode</i>	Dies ist der Standardmodus. Der Benutzer sieht den Inhalt der Webseite.
<i>DesignDisplayMode</i>	In diesem Modus kann der Benutzer die WebParts auf der Webseite per Drag & Drop umgruppieren.
<i>EditDisplayMode</i>	Im Edit-Modus wird das Editor-Zone-Template zusätzlich angezeigt, in dem die Eigenschaften von WebParts vom Benutzer verändert werden können.
<i>CatalogDisplayMode</i>	Im Katalog-Modus kann der Benutzer WebParts von der Page entfernen oder aus der Katalogauflistung wieder in der Seite platzieren.
<i>ConnectDisplayMode</i>	Im Connect-Modus können die Connections zwischen WebParts vom Benutzer im Browser bearbeitet werden.

**Tabelle 2.3** WebPart-Display-Modi verändern die Webseite

Im Beispielprojekt wurde ein *CheckBoxList*-Steuerelement verwendet, um den Modus zu ändern. Das Steuerelement wird am besten außerhalb der Zonen platziert. Dann kann vom Benutzer der Modus gewechselt werden.



**Abbildung 2.29** Wechseln des Modus

Wichtig ist, dass das Attribut *autopostback=true* in der *CheckBoxList* aktiviert ist, um beim Statuswechsel einen automatischen Server-Roundtrip und damit das Auslösen des *SelectedIndexChanged*-Ereignisses zu bewirken.

```
Protected Sub RadioButtonList1_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs)
    Select Case RadioButtonList1.SelectedValues
    Case 1
```

**Listing 2.31** Ändern des DisplayMode

```

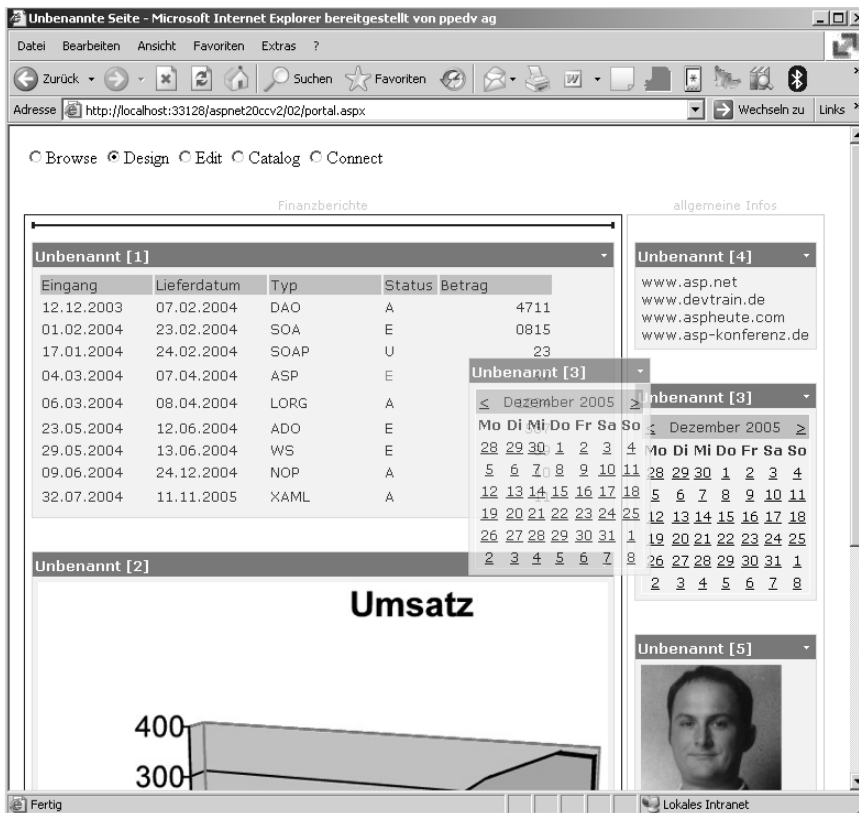
WebPartManager1.DisplayMode = WebPartManager.BrowseDisplayMode
Case 2
WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode
Case 3
WebPartManager1.DisplayMode = WebPartManager.EditDisplayMode
Case 4
WebPartManager1.DisplayMode = WebPartManager.CatalogDisplayMode
Case 5
WebPartManager1.DisplayMode = WebPartManager.ConnectDisplayMode
End Select
End Sub

```

**Listing 2.31** Ändern des DisplayMode (Fortsetzung)

## Design-Modus

Wenn der Benutzer nun den Design-Modus auswählt, können alle WebParts im Browser innerhalb der Zonen frei platziert werden. Dazu bewegen Sie die Maus auf den Titelbereich eines WebParts. Der Mauszeiger nimmt daraufhin die Gestalt des Symbols für Bewegung an. Halten Sie nun mit der Maus das WebPart-Steuerelement fest, und schieben Sie es auf einen anderen Bereich der Webseite. Die Zone wird blau umrandet und erhält eine Einfügemarke. Wenn Sie das Steuerelement dort loslassen, bleibt es auch dort.



**Abbildung 2.30** Ein WebPart wird verschoben

## EditorZone

Damit der Benutzer die Layout-Einstellungen von WebParts auch wirklich ändern kann, benötigt das Portal noch eine *EditorZone*, in der die Editor-WebParts platziert werden.

Als Erstes ziehen Sie aus der Toolbox das *EditorZone Steuerelement* in die dritte Spalte rechts neben die Tabelle auf der Webseite. Beachten Sie, dass dieses im Edit-Modus zusätzlich angezeigt werden muss und Platz benötigt.

Als Nächstes ziehen Sie die beiden Steuerelemente *AppearanceEditorPart* und *LayoutEditorPart* in die Editor-Zone. Diese beiden Steuerelemente erlauben es, Einstellungen an den einzelnen WebParts direkt vom Benutzer vornehmen zu lassen. Insgesamt gibt es vier mögliche Editor Parts.

EditorPart	Verwendung
<i>AppearanceEditorPart</i>	Damit kann der Benutzer Oberflächen-Eigenschaften von WebParts verändern. Dies sind z.B. Titel, ChromeType oder Größe.
<i>BehaviorEditorPart</i>	Damit kann der Benutzer das Verhalten von WebParts verändern. Das sind z.B. die möglichen Aktionen über die Verbs.
<i>LayoutEditorPart</i>	Damit kann der Benutzer Layout-Eigenschaften verändern. Dies sind z.B. ChromeState oder Zone.
<i>PropertyGridEditorPart</i>	Damit kann der Benutzer vom Entwickler zusätzlich definierte Eigenschaften von WebParts verändern.

**Tabelle 2.4** WebPartEditor Parts

Als Nächstes kann der Benutzer auf Details wie Überschrift und Größe Einfluss nehmen. Dazu wählen Sie im Benutzer-Steuerelement *Edit* als Anzeigemodus aus. Im Kontextmenü jedes einzelnen WebPart-Steuerelements kann man mit dem nun zusätzlichen Menüpunkt *Bearbeiten* in den Bearbeitungsmodus wechseln (Abbildung 2.31).

Mit Drücken des *OK*- bzw. *Übernehmen*-Buttons wird die Änderung dann übernommen. Der Browser zeigt die geänderte Webseite sofort an. Die Einstellungen werden dauerhaft in der Personalisierungsdatenbank gespeichert.

Es kann sein das verschiedene Editor Parts nicht angezeigt werden, wie in diesem Fall das *BehaviourEditorPart*. Dies hängt von einigen äußeren Einflussfaktoren wie Personalisierungsmodus oder Steuerelementtyp ab. In diesem Fall ist das *IsShared*-Attribut des zu editierenden WebParts mit seinem Standardwert *True* dafür verantwortlich. Für vertiefende Information zu diesem Thema ist die Dokumentation zu empfehlen.



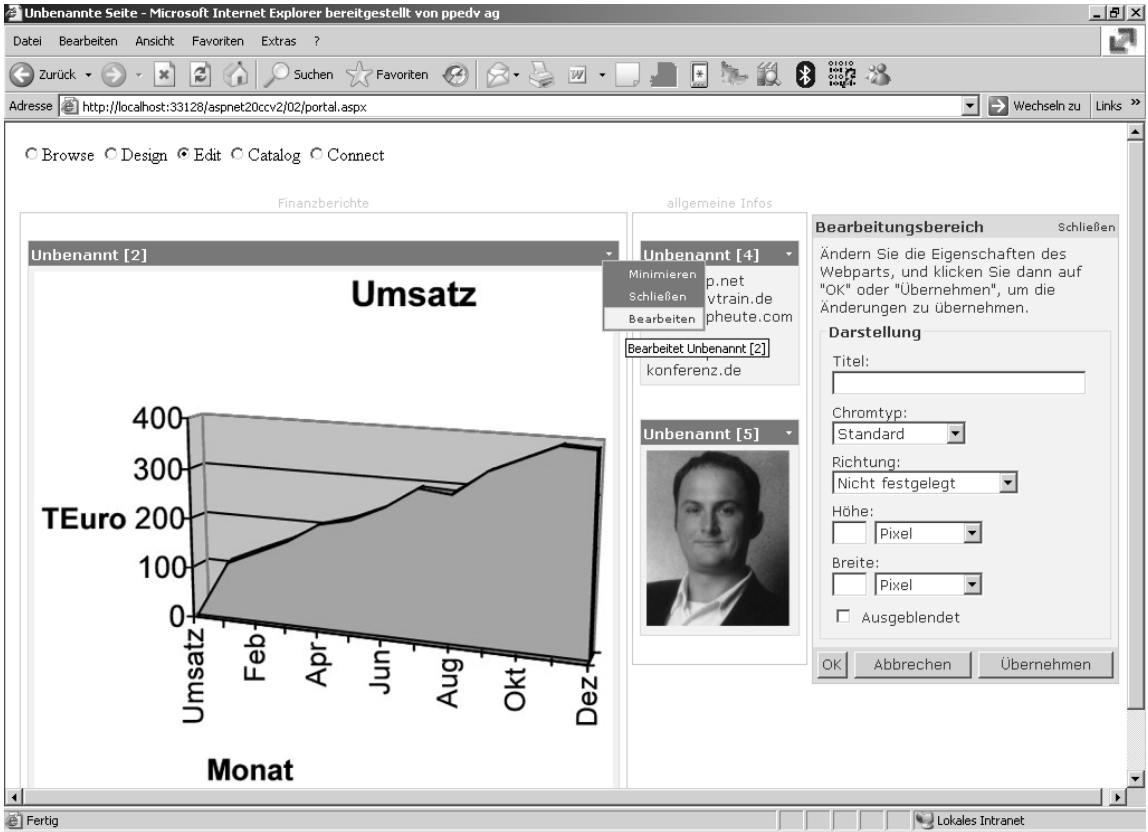


Abbildung 2.31 Der Benutzer ändert die Eigenschaften eines WebParts

## Catalog-Zone

Ihnen ist ein WebPart abhanden gekommen, weil Sie auf das Symbol für Schließen gedrückt haben? Keine Angst – das lässt sich wieder mithilfe einer *CatalogZone* herbeizaubern. Diese Zone enthält einen oder mehrere Kataloge von verfügbaren Steuerelementen für die Webseite. Allerdings unterstützt dieser Zonentyp nicht Drag & Drop, wie das einige unter Ihnen vielleicht vom SharePoint Portal Server kennen.

Ziehen Sie ein *CatalogZone* Steuerelement auf Ihre Webseite, am besten in die dritte Spalte unterhalb der Editorzone. Platzieren Sie in dieser Zone die drei weiteren Steuerelemente *PageCatalog*, *ImportedWebPart Catalog* und *DeclarativCatalog*.

EditorPart	Verwendung
<i>PageCatalog</i>	Im Seiten-Katalog werden die aus der Seite vom Benutzer entfernten WebParts aufgelistet.
<i>DeclarativCatalog</i>	Der deklarative Katalog wird vom Entwickler definiert und bietet eine Auflistung an WebParts.
<i>ImportedWebPartCatalog</i>	Im Import-Katalog kann der Benutzer selbst Katalog-Deklarationen importieren und so neue WebParts hinzufügen.

Tabelle 2.5 Catalog Parts

Zur Laufzeit werden im Seitenkatalog die vom Benutzer durch Schließen ausgeblendeten WebParts verwahrt. Im Katalog *Declarative* kann man darüber hinaus WebParts deklarieren, die auch mehrfach vom Benutzer in die Seite übernommen werden können.

Nach dem Aktivieren des Katalogmodus kann der Benutzer aus den drei Katalogen den Seiten-Katalog auswählen. Er sieht dann die von ihm vorher durch *Schließen* ausgeblendeten WebParts. Aus der Liste können dann ein oder mehrere WebParts ausgewählt und wieder in einer Zone platziert werden:

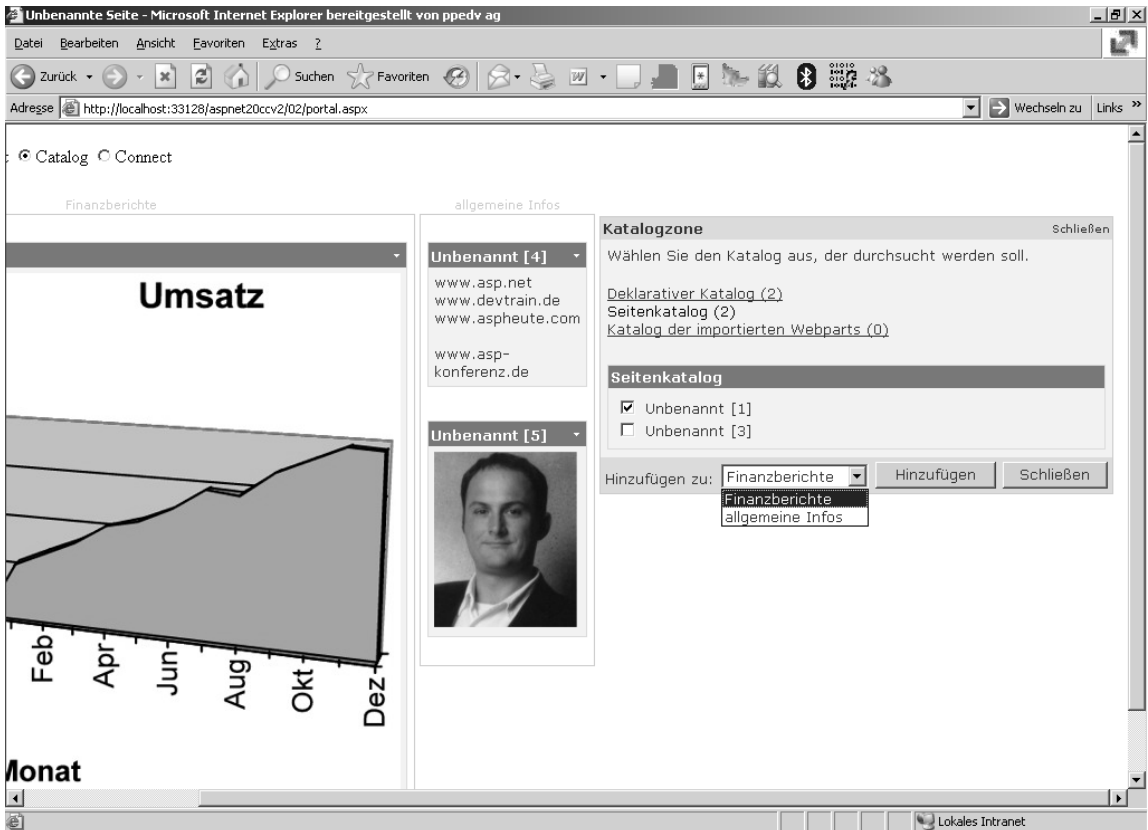


Abbildung 2.32 Das WebPart wurde ausgeblendet und befindet sich nun im Seitenkatalog

## Export und Import von WebParts

Die Einstellungen, die der Benutzer durchgeführt hat, lassen sich auch exportieren. In der Standardeinstellung ist dies nicht möglich. Erst mithilfe des Attributs *ExportMode* lässt sich eine der drei möglichen Optionen *All*, *None* und *NonSensitiveData* auswählen. Dies gilt für jeweils ein WebPart. Dazu fügen Sie in der HTML-Ansicht einem der Benutzer-Steuerelemente im Abschnitt *ZoneTemplate* das *ExportMode*-Attribut hinzu:

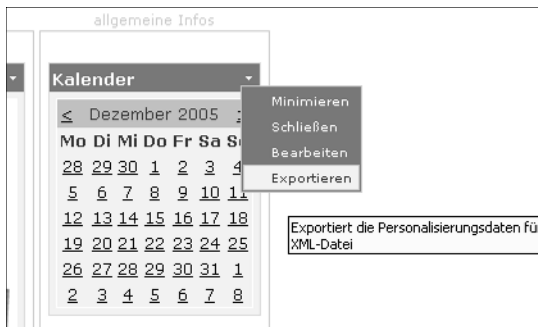
```
<ZoneTemplate >
  <uc2:umsatz ID="Umsatz1" runat="server" title="Umsatz" ExportMode="All" />
  <uc3:offeneAuftraege ID="OffeneAuftraege1" runat="server" title="offene Aufträge" />
</ZoneTemplate>
```

**Listing 2.32** Ein WebPart exportierbar machen

Allerdings muss in der *web.config* der Webanwendung die Funktion erst grundsätzlich erlaubt werden. Dazu wird das Attribut *enableExport* im *WebParts*-Element auf *true* gesetzt.

```
<webParts enableExport="true">
```

Der Benutzer sieht dann in der Webseite im Menü des WebParts einen zusätzlichen Eintrag *Export*.



**Abbildung 2.33** Export von WebPart-Einstellungen

Wenn mit der Einstellung *All* gearbeitet wird, werden auch vertrauliche Daten exportiert. Deshalb kommt in diesem Fall eine Warnmeldung (per JScript). Der Text dieser Meldung kann geändert werden, indem der Eigenschaft *ExportSensitiveDataWarning* des *WebPartManager* der gewünschte Text als String zugewiesen wird. Ist dieser leer, unterbleibt die Anzeige der Warnmeldung gänzlich.

```
WebPartManager1.ExportSensitiveDataWarning = ""
```

Die Daten werden dann in einer XML-Datei mit der Erweiterung *WebPart* lokal abgelegt und lassen sich auch wieder importieren.

Der *WebPartManager* erlaubt ebenfalls das Exportieren von WebParts. Die Funktion *ExportWebPart* erwartet als Parameter das *WebPart* und einen *XmlStreamWriter*.

Um die Daten wieder zu importieren, wird das Steuerelement *ImportCatalogPart* verwendet. Dieses wird in die Zone *Catalog* abgelegt.

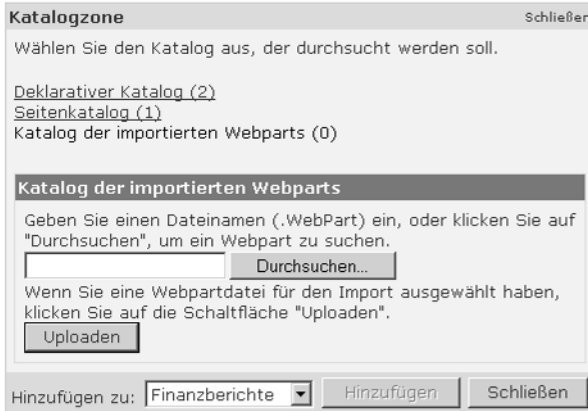


Abbildung 2.34 Import von WebPart-Definition

Alternativ kann auch die Funktion *ImportWebPart* des WebPart Managers verwendet werden. Diese erwartet einen *XmlStreamReader* als Parameter.

## Connection-Modus

Der letzte Modus ist der *ConnectionMode*. Damit können vom Anwender im Browser Verbindungen zwischen zwei WebParts hergestellt werden. Nehmen wir als Beispiel einen Suchdialog dessen Suchergebnisse als Liste in einem anderen WebPart angezeigt werden soll. Der Suchtext ist dabei per Connection zu transferieren. Dazu braucht man die Connection-Zone. Außerdem muss ein Webpart ein Provider-Interface beinhalten und mindestens ein zweites WebPart als so genannter *Consumer* auftreten. Um die Kommunikation noch zu standardisieren, muss der Suchtext in einem Interface definiert werden.

Dieses Kapitel konnte Ihnen in Anbetracht der Komplexität des Themas natürlich nur eine Einführung in die grundlegende Funktionalität der WebParts geben – doch die Grundlagen für erste eigene Gehversuche sind auf jeden Fall gelegt. Es ist übrigens zu erwarten, dass an den WebParts weiter gearbeitet wird und diese im Zusammenspiel mit den Windows Share Point Services und dem SharePoint Portal Server arbeiten.

## Kapitel 3

# Caching

### **In diesem Kapitel:**

Grundlagen	78
Output-Caching	78
Daten-Caching	84

# Grundlagen

Gerade bei Webanwendungen können erhebliche Lastspitzen auftreten. Nach dem Versand eines E-Mail-Newsletters klicken schon mal einige tausend Benutzer binnen weniger Minuten auf einen im Werbetext enthaltenen Hyperlink. Die Architektur, um auch mit solchen Szenarien umzugehen, wird dabei oft als große Kunst gehandelt. Vielfach wird versucht, diesem Aufkommen mit einer mehrschichtigen Architektur, auch als *n-Tier* bekannt, zu begegnen. Die Kommunikation der dabei verwendeten Softwareschichten und damit der entsprechenden Objekte wird über Remoting, Web Services oder Enterprise Services implementiert. Das mag für den Entwickler eine Herausforderung sein oder sich in den Projektspezifikationen gut lesen – notwendig ist es selten.

Dabei liegt der Flaschenhals meist bei den Daten. Diese sollen einmalig vorhanden sein. Häufige Lesezugriffe sind zu vermeiden. Auch die Erstellung von immer wieder gleichem HTML-Code aus der ASPX-Seite verbraucht viele Prozessor-Ressourcen.

Genau hier setzt ASP.NET mit Caching ein. Damit werden Daten, egal ob erzeugte HTML-Seiten, Tabellendaten oder anderes, im Arbeitsspeicher des Webservers gespeichert. Performancesteigerungen sind dabei bis um den Faktor 100 möglich. Dabei können Änderungen in dem Caching-Verhalten einzelner Seiten ohne größere Eingriffe in den Code gemacht werden. Auch ein nachträgliches Aktivieren von Caching in bestehenden Seiten ist einfach möglich. Dabei teilt sich eine Anwendung den Cache.

Caching ist im Internet-Umfeld eine alte und bewährte Technologie. Auf diese Weise arbeiten Browser und Proxy Server schon seit vielen Jahren. In der Windows-Welt kommt dieses Konzept fast nie zum Einsatz.

Artikeldaten eines Online-Shops ändern sich in der Praxis selten und müssen daher auch nicht bei jeder Suchanfrage neu aus der Datenbank geladen werden. Die Daten werden stattdessen mithilfe von Caching im Arbeitsspeicher zwischengespeichert. Das verringert die Prozessorbeltastung des SQL Servers und den Netzwerkverkehr. Außerdem ist ein Lesezugriff auf den Arbeitsspeicher um ein Vielfaches schneller als auf eine Festplatte. Was damit möglich ist, zeigt die Webseite [www.asp.net](http://www.asp.net). Diese wird nur wegen der Ausfallsicherheit mit zwei Windows 2003-Webservern betrieben. Als dritter Server kommt ein SQL Server zum Einsatz. In dieser Hardware-Architektur ist keine externe Objektkommunikation notwendig. Dennoch genügt diese Hardwarekonfiguration den Anforderungen vollauf – so hat z.B. der Forumbereich mehr als eine halbe Million registrierte Benutzer.

Caching ist eine einfache, anspruchslose Technologie. Sogar zu einem späteren Zeitpunkt lässt sich an kritischen Stellen der Anwendung Caching implementieren und das, ohne große Codeänderungen vornehmen zu müssen. Dabei verwaltet sich das Caching weit gehend selbst. Wenn der Arbeitsspeicher zur Neige geht, werden veraltete oder selten genutzte Daten im Cache einfach entladen. Sogar ein sich selbst pflegender Cache ist möglich. Wenn sich die Quelldaten ändern, kann der nun veraltete Inhalt des Caches automatisch erneuert werden. Gerade hier wird es die Kenner freuen, zu hören, dass jetzt auch eine SQL-Server-Cache-Abhängigkeit definiert werden kann. Für jede Anforderung bietet Caching das Richtige.

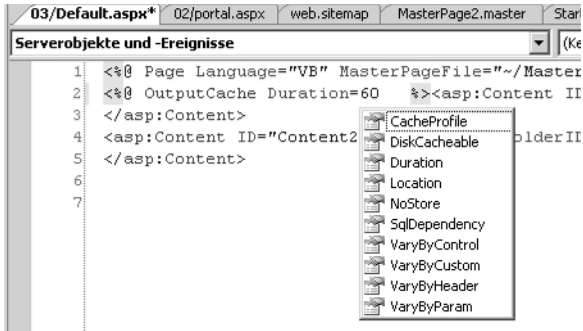
In ASP.NET wird die Caching-Technologie in zwei Gruppen unterteilt: *Output-Caching* und *Daten-Caching*.

## Output-Caching

Das Erstellen des HTML-Codes aus der ASPX-Seite braucht relativ viel Zeit. Dieser Vorgang wird *Rendern* genannt. Da die gleiche ASPX-Seite mit gleichen Daten immer zum gleichen HTML-Ergebnis führt, ist es

eine erhebliche Zeitersparnis, diesen erzeugten Code einfach am Server zwischenspeichern. Das lässt sich ganz einfach per Deklaration einschalten und konfigurieren.

Die Einstellungen zum Caching werden dazu in der ASPX-Seite nach der *Page*-Anweisung mit der *OutputCache*-Deklaration erstellt.



**Abbildung 3.1** Für das Caching stehen reichlich Optionen zur Verfügung

Die Kombination verschiedener Attribute beeinflusst das Caching-Verhalten. Wenn eine Seite abhängig von der QueryString-Variablen *ArtikelID* jeweils das Datenblatt des Artikels anzeigt, reicht es nicht, die Seite nur einmal im Cache zu halten. Es muss pro Artikel eine Variante der Seite existieren. Das erreicht man mit dem Attribut *VaryByParam* und dem Namen der Querystring-Variablen. Damit existiert die aufgerufene Seite mehrfach im Cache.

```
<%@ OutputCache Duration="60" NoStore="false" VaryByParam="ArtikelID" CacheProfile="myCache" %>
```

Die Möglichkeiten, das Caching zu beeinflussen, sind sehr umfangreich. Im Folgenden werden die Attribute der *OutputCache*-Deklaration beschrieben.

Attribut	Verwendung
<i>CacheProfile</i>	Die Cache-Einstellungen werden aus einem benannten Profil ( <i>Profile</i> ) geladen. Profiles können in der Datei <i>web.config</i> deklariert werden.
<i>DiskCacheable</i>	Dieser boolesche Wert hat keine Auswirkung mehr. Die Funktion wurde in einem sehr späten Stadium gestrichen.
<i>Duration</i>	Die Lebensdauer in Sekunden. Nach Ablauf der Zeit wird die ASPX-Seite neu ausgeführt und das neue Ergebnis in den Cache geladen.
<i>Location</i>	Sogar der Speicherort des Caches lässt sich vordefinieren. Ob dieser allerdings auch funktioniert, hängt stark von der Umgebung ab. Mögliche Werte sind: <i>Any</i> , <i>Client</i> , <i>Downstream</i> , <i>Server</i> , <i>ServerAndClient</i> oder <i>None</i> . Standardeinstellung ist <i>Any</i> .
<i>NoStore</i>	Damit wird im Header <i>no-store</i> gesendet, sodass Proxy-Server diese Seite nicht zwischenspeichern.
<i>Shared</i>	Wird nur bei UserSteuerelementen verwendet und gibt an, ob der Cache des UserSteuerelements über verschiedene Seiten hinweg verwendet wird.
<i>SqlDependency</i>	Definiert eine SQL-Abhängigkeit. Üblicherweise wird dieser String im Format Datenbank:Tabellenname zugewiesen.

**Tabelle 3.1** Attribute für den OutputCache

Attribut	Verwendung
<i>VaryBySteuerelement</i>	Definiert anhand von Steuerelement-Namen die Abhängigkeit von diesen. Mehrere Steuerelemente werden durch Kommas getrennt.
<i>VaryByCustom</i>	Damit kann z.B. eine Abhängigkeit des Caches vom Browsertyp definiert werden. ASP.NET erzeugt pro Browser unterschiedliche HTML-Seiten. Die Parameter werden als Komma separierte Liste angegeben.
<i>VaryByHeader</i>	Damit kann der Cache in Abhängigkeit des Headers erstellt werden.
<i>VaryByParam</i>	Damit wird der Cache vom QueryString abhängig gemacht. Wenn es mehrere QueryString-Variablen gibt, werden diese durch Kommata getrennt angegeben. Dieses Attribut ist Pflicht und kann dann z.B. mit <i>none</i> vorbelegt werden.

**Tabelle 3.1** Attribute für den OutputCache (Fortsetzung)

Das Verhalten des Output-Caches kann auch zur Laufzeit kontrolliert werden. Dafür besitzt das Response-Objekt eine Cache-Eigenschaft. Die Eigenschaften und Funktionen unterscheiden sich allerdings erheblich. In der Hauptsache wird dabei der HTTP Header beeinflusst.

```
Response.Cache.VaryByParams("*") = True
```

## Cache in web.config

Natürlich lassen sich die Cache-Einstellungen wie üblich für die gesamte Anwendung vornehmen. Dies geschieht im Bereich *Caching*. Im Folgenden wird diese Sektion in der Datei *web.config* im Detail besprochen.

Unterhalb des `<caching>`-Elements können vier Bereiche existieren:

Unterelement	Verwendung
<i>cache</i>	Damit wird die Cache-API angesprochen. Zusätzliche Attribute legen die Einstellungen fest.
<i>outputCache</i>	Damit wird das Output Caching konfiguriert.
<i>outputCacheSetting</i>	Hier werden Cache-Profile über Unterelemente definiert.

**Tabelle 3.2** Cache-Elemente in *web.config*

## cache

Im Folgenden wird ein kurzer Beispielausschnitt des Cache-Elements aus der *web.config* mit möglichen Attributen gezeigt.

```
<cache disableDependencies="True" disableExpiration="True" disableMemoryCollection="True"
percentagePhysicalMemoryUsedLimit="10" privateBytesLimit="20"></cache>
```

**Listing 3.1** Beispiel *cache*-Element

## outputCache

Das *outputCache*-Element übernimmt zentral die Steuerung des Cache-Verhaltens mithilfe von Attributen. Die meisten davon sind selbsterklärend und haben den Wert *true*. Einzig das Attribut *OmitVaryStar* lässt eigentlich keine Rückschluss auf seine Bedeutung zu. Dies steuert das Caching-Verhalten abhängig von der Header-Information. Der Standardwert ist *false*.



```
<outputCache omitVaryStar="True" enableFragmentCache="True" enableOutputCache="True"
sendCacheSteuerelementHeader="True">
```

## outputCacheSettings

*outputCacheSettings* beinhaltet keine Attribute. Das einzige Unterelement ist *outputCacheProfiles*.

## outputCacheProfiles

Je nach Abfragehäufigkeit und Datenmenge wird Caching dazu passend eingestellt. Dabei ist Caching nicht die Lösung aller Probleme. Es kann auch welche verursachen. So sind die Daten im Cache in der Regel nicht aktuell. Es gilt also Kompromisse zu finden, die am besten auf die Bedürfnisse abgestimmt sind. In der Regel müssen unterschiedliche Seiten unterschiedlich gecacht werden, um das beste Ergebnis zu erzielen. Das kann man natürlich alles in den einzelnen Seiten festlegen. Aber die Datenmenge und die Nutzung ändern sich – und was gestern gut war, kann heute schon schlecht sein, mit der daraus folgenden Notwendigkeit: Man müsste in jeder einzelnen Webseite das Caching ändern.

Da kommt es gerade richtig das ASP.Net 2.0 nun *Cache Profile* anbietet. Diese sind sozusagen eine Zusammenfassung von Cache-Regeln. In der Seite wird nunmehr das Profil über den Namen referenziert. Wenn sich etwas an den Regeln ändern soll, kann man diese Änderung zentral in der Datei *web.config* vornehmen. Im Unterelement *outputCacheProfiles* wird per *Add* ein neues Profil angelegt. Die Attribute werden von Intellisense im Editor vorgegeben, sind aber identisch mit denen in der Page-*outputCache*-Deklaration.

```
<aching>
<outputCacheSettings>
  <outputCacheProfiles>
    <add name="myCache" enabled="true" duration="60"/>
  </outputCacheProfiles>
</outputCacheSettings>
</aching>
```

**Listing 3.2** Ein Cache-Profil aus *web.config*

In der ASPX-Seite wird dann mit dem Attribute *CacheProfile* der Name des Profiles angegeben:

```
<%@ OutputCache CacheProfile="myCache" %>
```

Ein Profil kann in einer untergeordneten *web.config* mit *Remove* wieder entfernt werden.

## Fragment-Caching

Diese Funktionalität ist seit ASP.NET 1.x vorhanden, wird aber der Vollständigkeit halber hier kurz beschrieben. Fragment-Caching erlaubt es, Teile der Seite zwischenspeichern. Genauer gesagt wird der Inhalt von Benutzer-Steuerelementen, die in die Seite eingebettet sind, zwischengespeichert.

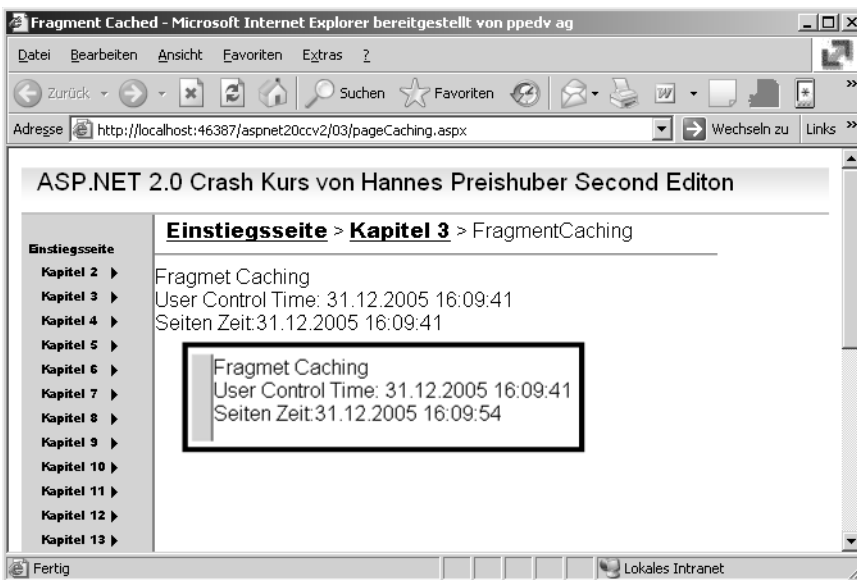
Dazu muss aber in der Datei *web.config* überprüft werden, ob Fragment-Caching auch erlaubt ist. Dazu muss das Attribut *enableFragmentCache* im Cache-Element gesetzt sein.

```
<outputCache enableFragmentCache="true"/>
```

**Listing 3.3** Caching-Einstellungen in *web.config*

Ein Benutzer-Steuerelement zeigt so im folgenden Beispiel nach dem Refresh eine frühere Uhrzeit an als die Seite selber. Der Inhalt des Benutzer-Steuerelements zeigt zwar auch die Uhrzeit an, aber eben gecacht. Das Caching wird in der ASCX-Datei, also dem Benutzer-Steuerelement ebenfalls per *OutputCache*-Deklaration eingestellt.

```
<%@ OutputCache Duration="30" VaryByParam="None" %>
```



**Abbildung 3.2** Die UserSteuerelement-Zeit hinkt hinter der Seitenzeit her

In der ASPX-Seite darf allerdings kein *outputCache* deklariert werden. Damit wird immer nur ein Teil der Seite zwischengespeichert. Genau den umgekehrten Effekt erreichen Sie durch den Einsatz des Substitution-Steuerelements.

## Substitution

Wenn sich nur ein kleiner Teil der Seite ändert und der Großteil der Seite immer gleich bleibt, bietet sich die Substitution an. Dazu wird eine Funktion festgelegt, die immer ausgeführt wird und deren Rückgabe dann in die gecachte Seiten eingebaut wird.

In der Praxis kann damit z.B. ein Produkt angezeigt werden, dessen Eckdaten wie Bild und Beschreibung sich selten ändern. Der Lagerbestand, der auf der gleichen Seite steht, muss aber immer aktuell sein.

In der Umsetzung haben Sie die Wahl, dies entweder mit dem *Substitution*-Steuerelement oder im Response-Objekt mit der Funktion *WriteSubstitution* zu erledigen.

## Substitution-Steuerlement

Das Steuerlement befindet sich in der Werkzeugleiste und kann per Drag&Drop auf der ASPX-Seite platziert werden. Es gibt eigentlich nur ein wichtiges Attribut im Substitution-Steuerlement, nämlich *MethodName*. Diesem wird der Name der aufzurufenden Funktion übergeben.

```
<asp:Substitution ID="Substitution1" Runat="server" MethodName="FillPlace" />
```

Der Inhalt des *Substitution*-Steuerlements wird nicht gecacht. Die Methode liefert die Ausgabe als Zeichenkette zurück. Als Parameter muss der aktuelle HttpContext übergeben werden und die Methode muss *shared* (VB) oder *static* (C#) sein.

```
Shared Function aktuelleZeit(context As HttpContext) As String
    return DateTime.Now.ToString()
End Function
```

### Listing 3.4

### Listing 3.5 Methode zum Substitution-Steuerlement

## WriteSubstitution

Das *Response*-Objekt beinhaltet die Funktion *WriteSubstitution*. Es wird ein Delegat auf die Funktion übergeben, die den dynamischen Inhalt der Seite liefert. In VB 2005 wird dazu *AddressOf* verwendet.

```
<%@ OutputCache Duration="60" VaryByParam = "none" %>
<script runat="server">
Shared Function aktuelleZeit(context As HttpContext) As String
    return DateTime.Now.ToString()
End Function
</script>
Uhrzeit: <% Response.WriteSubstitution(New HttpResponseSubstitutionCallback(AddressOf aktuelleZeit)) %>
```

### Listing 3.6

### Listing 3.7 Beispiel für *Response.WriteSubstitution*

## Zeiten in der Praxis

Die nun erworbenen Kenntnisse werden im folgenden Beispiel auch gleich in die Praxis umgesetzt. Einer ASPX-Seite wird eine Cache-Lebensdauer von 60 Sekunden eingeräumt. Es wird die aktuelle Zeit in einem Label-Steuerlement ausgegeben. Diese Anzeige kann also bis zu 59 Sekunden die Zeit aus der Vergangenheit ausgeben. Im *Substitution*-Steuerlement wird durch die selbst deklarierte Funktion *neueZeit* genau dasselbe gemacht.

```
<%@ OutputCache DiskCacheable="true" Location="ServerAndClient" Duration="60" NoStore="false"
VaryByParam="none" %>
<script runat="server">
Shared Function neueZeit(ByVal context As HttpContext) As String
    Return "aktuell:" + Date.Now.TimeOfDay.ToString()
End Function
```

### Listing 3.8 Zeit und gecachte Zeit können sich unterscheiden

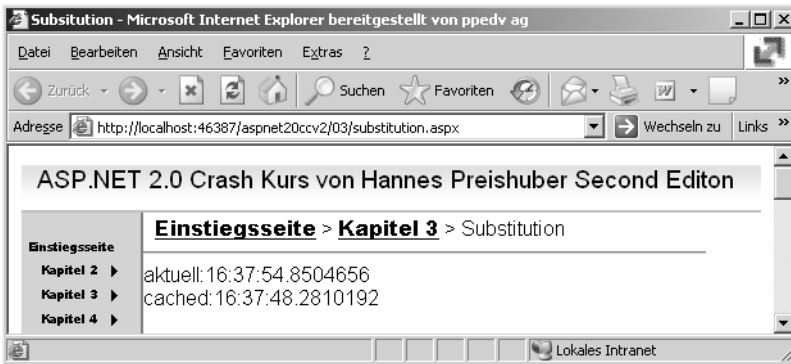
```

Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Label1.Text = "cached:" + Date.Now.TimeOfDay.ToString
End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <asp:Substitution ID="Substitution1" Runat="server" MethodName="neueZeit" /><br />
    <asp:Label ID="Label1" Runat="server" Text="Label"></asp:Label>
</asp:Content>

```

**Listing 3.8** Zeit und gecachte Zeit können sich unterscheiden (*Fortsetzung*)

Den Funktionsnachweis liefert die Abbildung 3.3. Beim ersten Aufruf sind die Zeiten noch identisch, doch schon beim nächsten Refresh der Seite sieht es anders aus. Die Zeiten unterscheiden sich deutlich, da die eine Zeitanzeige aktuell in der Funktion berechnet wurde und die andere aus der gecachten Seite stammt.



**Abbildung 3.3** Gecachte Seite mit »aktueller« Uhrzeit

## Daten-Caching

Neben dem Output-Caching können auch Daten zwischengespeichert werden. Der Zugriff auf das Cache-Objekt erfolgt meist per Code über die *Cache API*. Aber auch Steuerelemente wie das *SqlDataSource*-Steuerelement machen davon Gebrauch.

Das Cache-Objekt gilt immer einmalig pro Anwendung und ist deshalb mit einer *Application*-Variable zu vergleichen. Sowohl der *Application*-Variablen als auch dem Cache-Objekt ist es egal, welche Art von Daten gespeichert wird. Der Cache dabei ist allerdings intelligenter und kann autark über seinen eigenen Inhalt entscheiden. Beide brauchen übrigens bei intensiver Verwendung viel Arbeitsspeicher: Geizen Sie deshalb bei Webservern nicht mit Speicher! Falls der zur Verfügung stehende Arbeitsspeicher doch nicht reichen sollte, werden nach einer bestimmten Logik ältere und unwichtigere Daten daraus entfernt.

Eine weitere wichtige Eigenschaft von gecachten Daten ist, dass sie sich quasi selbst erneuern können, wenn bestimmte Ereignisse eintreffen. Das könnte z.B. die Änderung einer Datei sein oder das Erreichen eines bestimmten Zeitpunktes. Diese Abhängigkeiten werden auch als Cache-Dependency bezeichnet. Auch bringt ASP.NET 2.0 eine Neuerung, die so genannte SQL-Dependency, also die Abhängigkeit von im SQL Server gespeicherten Daten.

Zusammenfassend gesagt handelt es sich beim Caching um eine besonders nützliche und einfache Möglichkeit, schnellere Webanwendungen zu erstellen.

## Cache API

Das Cache-Objekt befindet sich weit gehend unter der Kontrolle Ihres Codes. Sie können Daten als Objekte laden und wieder entfernen. Mit zusätzlichen Parametern wird die Lebensdauer der Objekte im Cache gesteuert. Das Cache Objekt verwaltet über eine Hashtable per Schlüssel die Werte als Objekt. Wenn die Daten aus dem Cache gelesen werden, müssen in der Regel zuerst diese Objekte in den notwendigen Datentyp umgewandelt werden.

### Insert

Um Daten in den Cache zu legen, kommt eine der Methoden *Add* oder *Insert* zum Einsatz. Dabei existiert *Insert* in mehrfachen Überladungen mit verschiedenen möglichen Parametern.

```
Cache.Insert("Cache1", Date.UtcNow)
```

Mit einem weiteren Parameter lässt sich eine Abhängigkeit des gecachten Objektes von seinen Ursprüngen definieren. Dazu muss ein *CacheDependency*-Objekt erstellt werden. In diesem wird z.B. eine Abhängigkeit von einer Datei konfiguriert. Wenn sich die Datei ändert, wird der Cache erneuert.

```
Dim newCachDP As New CacheDependency(Server.MapPath("web.config"), DateTime.Now)
Cache.Insert("cache2", Date.UtcNow, newCachDP)
```

#### Listing 3.9 Cache Dependency erstellen

Wenn Sie in der Insert-Methode nicht alle Parameter füllen können, übergeben Sie einfach *nothing* (VB) oder *null* (C#) als Wert.

Etwas gewöhnungsbedürftig sind die beiden Optionen zur Lebensdauer. Entweder man setzt mit dem vierten Parameter die absolute Zeit oder dem fünften Parameter eine Zeitspanne. Die Zeitspanne wird dabei als *Timespan*-Objekt übergeben, und bei seiner Instanziierung werden entsprechende Parameter mit Stunden, Minuten und Sekunden zur Festlegung der Zeitspanne verwendet. Allerdings muss in diesem Fall der vierte Parameter auf *Date.MaxValue* gesetzt sein, anderenfalls wird eine Ausnahme ausgelöst:

```
Cache.Insert("Cache3", Date.UtcNow, Nothing, Date.MaxValue, New TimeSpan(0, 0, 10))
```

Die letzte Überladung erlaubt es, eine Funktion auszuführen, wenn der Cache-Vorhaltezeitraum abläuft. Das funktioniert mithilfe eines Delegates auf einen Funktionsnamen:

```
Cache.Insert("cache4", Date.UtcNow, Nothing, _
    Date.Now.AddMinutes(1), Nothing, CacheItemPriority.Normal, _
    New CacheItemRemovedCallback(AddressOf RemovedCallback))
```

#### Listing 3.10 Einstellung der automatischen »Datenpflege« im Cache-Objekt

Der Vollständigkeit halber wird hier auch noch die *CallBack*-Funktion aufgeführt. Diese kann verwendet werden, um die Cache-Daten erneut zu laden.

```
Public Sub RemovedCallback(ByVal key As String, ByVal value As Object, _
                          ByVal reason As CacheItemRemovedReason)
    ListBox1.Items.Add(Date.Now.ToString)
End Sub
```

**Listing 3.11** Anwendung der Callback-Funktion

Die vierte Überladung von *Insert* ist identisch zum Parameterisieren mit der *Add*-Funktion.

```
Cache.Add("Cache1", Date.UtcNow, Nothing, Date.MaxValue, New TimeSpan(0, 0, 1), _
        CacheItemPriority.Normal, Nothing)
```

Die Priorität kann in sechs Stufen beginnend von *Low* bis *High* deklariert werden. Als siebten Wert gibt es die Option *notRemoveable*.

Mit der Methode *Cache.Remove* kann ein Objekt über seinen Schlüssel auch programmatisch entfernt werden.

## SQL-Abhängigkeit

Wenn Daten einer Tabelle dargestellt werden sollen, bietet ASP.NET 2.0 nun ein spezielles Steuerelement dazu an, das DataSource-Steuerelement. Damit steht zur Entwurfszeit ein visuelles Steuerelement für den Datenzugriff zu Verfügung. Zur Laufzeit wird dieses nicht angezeigt. Es dient einzig und allein als Brücke zu den visuellen Steuerelementen wie GridView, die zur Darstellung genutzt werden. Natürlich wird dazu weiterhin ein SQL-Kommando und ein Connection-String benötigt. Die Details dazu finden Sie in den späteren Kapiteln zum Datenzugriff.

Hier wird gezeigt wie man mit dem Attribut *EnableCaching* das Caching für dieses Steuerelement aktiviert.

Durch zusätzliche Attribute kann die Lebensdauer der Daten gesteuert werden. Das zentrale Attribut ist hier in diesem Beispiel aber *SqlCacheDependency*. Damit kann die Abhängigkeit von der Datenbank und Tabelle angegeben werden, und zwar in der Form *Datenbankname:Tabellenname*. Achten Sie auch hier auf die korrekte Schreibweise, da diese die Groß-/Kleinschreibung berücksichtigt.

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server" SelectCommand="SELECT * FROM [Products]"
    ConnectionString="Server=localhost;Integrated Security=True;Database=Northwind"
    EnableCaching="true"
    CacheDuration="Infinite"
    CacheExpirationPolicy="Absolute"
    SqlCacheDependency="Northwind:Products" >
</asp:SqlDataSource>
```

**Listing 3.12** SqlDataSource-Steuerelement mit Abhängigkeit

Darüber hinaus ist die korrekte Konfiguration des Bereiches *caching* in der *web.config* notwendig.

---

**HINWEIS** SQL Express kommt ohne vorinstallierte Datenbanken. In Kapitel 12 erfahren Sie wie man die Nordwind-Demo-datenbank in Betrieb nimmt.

---

## SQL-Abhängigkeit konfigurieren

Wenn eine Tabelle angezeigt werden soll, reicht es in einigen Fällen nicht aus, diese einfach alle 60 Sekunden neu zu laden. Vielleicht verspüren Sie den Wunsch, den Cache zu erneuern, wenn die Daten sich geändert haben. Dabei hilft Ihnen das Attribut `sqlCacheDependency`.

Es gibt zwei grundlegende Verfahren, die dabei zu Anwendung kommen. Für SQL Server 7.0 und SQL Server 2000 kommt ein so genannter Poll-Mechanismus zum Einsatz. Dabei wird in einem festgelegten Intervall die Datenbank mit einer gespeicherten Prozedur (*Stored Procedure*) abgefragt. Wenn sich die Daten geändert haben sollten, werden diese neu geladen. Änderungen betreffen immer eine ganze Tabelle.

SQL Server 2005 verfügt für diesen Zweck über einen Benachrichtigungs-Service, der in einem »Push«-Verfahren über Änderungen an den Daten bis auf Satzebene hinunter berichtet.

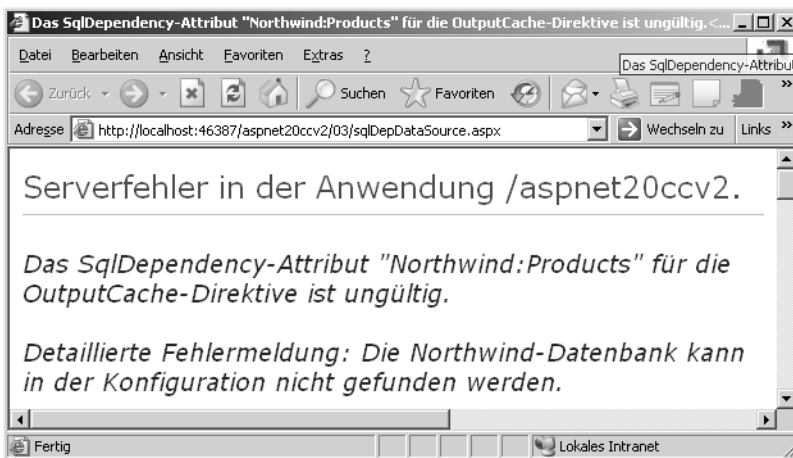
Da Datenbank-Zugriffe erst in späteren Kapiteln detailliert behandelt werden, beschränken sich die folgenden Ausführungen auf die Grundlagen und Administration der SQL-Abhängigkeit.

Im folgenden Beispiel wird das `SqlDataSource`-Steuerelement so konfiguriert, dass es auf die Tabelle `Products` aus der `Northwind` Datenbank zugreift. Zusätzlich wird das Caching aktiviert und die Cache-Abhängigkeit auf diese Tabelle definiert.

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server" SelectCommand="SELECT * FROM [Products]"
  ConnectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;Initial Catalog=Northwind"
  EnableCaching="true"
  CacheDuration="Infinite"
  CacheExpirationPolicy="Absolute"
  SqlCacheDependency="Northwind:Products">
</asp:SqlDataSource>
```

**Listing 3.13** Ein Datenbank-Zugriff mit SQL Cache Dependency

Das Ergebnis im Browser ist allerdings ernüchternd.



**Abbildung 3.4** Fehlermeldung bei Einsatz von `sqlDependency`

## Konfigurieren

Um die Datenbank auf die Aufgabe vorzubereiten, den Cache zu bedienen, gibt es das Befehlszeilen-Tool *aspnet\_regsql*. Damit kann man auch einen SQL Server auf das Membership-System von ASP.NET vorbereiten, indem man den Aufruf ohne Parameter ausführt.

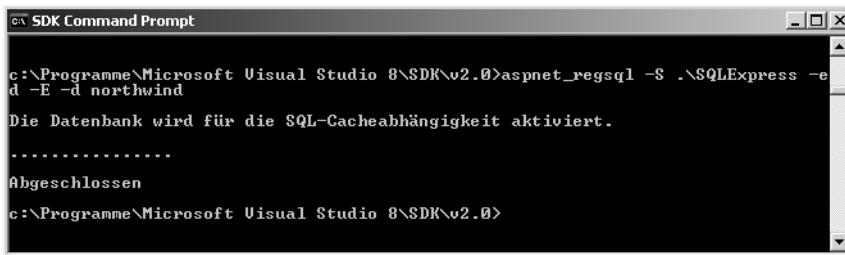
Am besten starten Sie die Befehlszeile und wechseln Sie in das Verzeichnis in dem *aspnet\_regsql* liegt. Dies ist je nach Installation z.B. *C:\WINDOWS\Microsoft.Net\Framework\v2.0.50727\*. Alternativ können Sie auch die Visual Studio 2005-Eingabeaufforderung verwenden, in der die Pfade so gesetzt sind, dass ein direkter Aufruf möglich ist.

### HINWEIS

Achten Sie darauf dass die benötigten Netzwerkprotokolle TCP/IP oder Named Pipes aktiviert sind. Dies geschieht mit dem Werkzeug *SQL Server Configuration Manager*. In der Standardeinstellung sind diese beim SQL Server 2005 deaktiviert.

Im ersten Schritt wird die Datenbank vorbereitet. Dazu muss der Parameter *-ed* angegeben werden. Mit dem Parameter *-E* wird eine *Trusted Connection* verwendet, also kein Benutzername und Passwort. Der Parameter *-d* gibt den Namen der Datenbank an. Vergessen Sie nicht den Namen der SQL Server Instanz mit dem Parameter *-S* zu bestimmen. Geben Sie folgenden Befehl an der Befehlszeile ein:

```
aspnet_regsql -S .\SQLEXPRESS -ed -E -d northwind
```



```

c:\Programme\Microsoft Visual Studio 8\SDK\v2.0>aspnet_regsql -S .\SQLEXPRESS -ed -E -d northwind
Die Datenbank wird für die SQL-Cacheabhängigkeit aktiviert.
.....
Abgeschlossen
c:\Programme\Microsoft Visual Studio 8\SDK\v2.0>

```

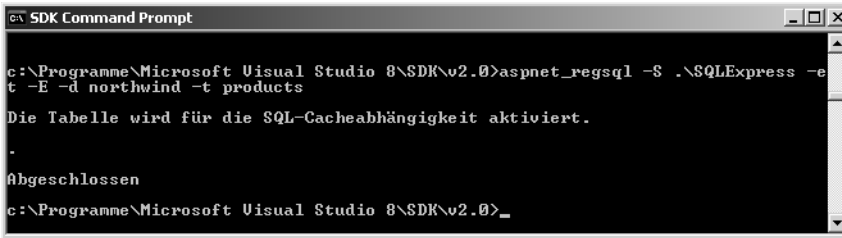
Abbildung 3.5 Erster Schritt für die SQL-Datenbank

Als Nächstes wird *aspnet\_regsql* mit dem Parameter *-et* aufgerufen, um die Tabelle anzugeben. Der Name folgt dem Parameter *-t*. In diesem Beispiel wird mit dem Parameter *-U* der Benutzername angegeben. Wenn das Passwort nicht angegeben wird, erhält man eine Eingabeaufforderung, um dieses zu erfassen.

```
aspnet_regsql -et -U sa -d northwind -t customers
```

Password:





```
C:\ SDK Command Prompt
c:\Programme\Microsoft Visual Studio 8\SDK\v2.0>aspnet_regsql -S .\SQLExpress -e
-t -E -d northwind -t products
Die Tabelle wird für die SQL-Cacheabhängigkeit aktiviert.
.
Abgeschlossen
c:\Programme\Microsoft Visual Studio 8\SDK\v2.0>_
```

**Abbildung 3.6** Die Tabelle wird im zweiten Schritt aktiviert

Will man später einmal wissen, ob alles funktioniert hat, ruft man `aspnet_regsql` mit dem Parameter `-lt` auf.

```
aspnet_regsql -lt -U sa -d northwind
```

---

**ACHTUNG** Achten Sie auf die korrekte und einheitliche Schreibweise von Datenbank- und Tabellennamen, da hier zwischen Klein- und Großschreibung unterschieden wird.

---

Dadurch werden im SQL Server jeweils eine Stored Procedure und eine Tabelle angelegt. Die Tabelle mit dem Namen `AspNet_SqlCacheTablesForChangeNotification` beinhaltet für jedes Polling einen Datensatz mit dem entsprechenden Tabellennamen.

## web.config

In der Datei `web.config` der Webanwendung werden im Bereich `Caching` die Einstellungen global vorgenommen.

Dazu wird ein Unterelement `SqlCacheDependency` angelegt. Dort wird für jede Datenbank ein Element `Databases` erzeugt. In ihm können per `Add`-Element neue Tabellen zur Abhängigkeit hinzugefügt werden. Das Intervall, in dem ASP.NET an den Datenbankserver die Stored Procedure für das Polling absetzt, wird per `pollTime` in Millisekunden deklariert.

```
<caching>
<sqlCacheDependency enabled="true" pollTime="10000">
<databases>
  <add name="Northwind" connectionStringName="AppConnectionString1" />
</databases>
</sqlCacheDependency>
</caching>
```

**Listing 3.14** Cache-Bereich aus `web.config`



## Kapitel 4

# Architektur

**In diesem Kapitel:**

Sicherheit im Design	92
Schichten	96
Statusinformationen	98
Entwickeln im Team	101

# Sicherheit im Design

Sicherheit ist heute auch in der Anwendungs- und Internetentwicklung ein wichtiges Thema. Sie sollten immer darauf achten, sichere Software zu entwickeln, auch wenn man am Ende im fertigen Produkt kaum etwas von den Bemühungen sieht. Trotz bester Infrastruktur und verwendeter Basis-Technologie ist man nie vor Sicherheitslücken gefeit. So sind viele der Sicherheitslücken im Internet Explorer erst aus Funktionen entstanden, die ursprünglich aus noblen Gründen zur Steigerung der Anwenderfreundlichkeit, Flexibilität oder Ergonomie implementiert wurden. Deshalb soll an dieser Stelle auch der eindeutige Hinweis auf Ihre Verantwortung als Entwickler ergehen, nach Möglichkeit ständig beim Stand der Sicherheitstechnik auf dem Laufenden zu bleiben und dies in Ihrer Entwicklung zu berücksichtigen.

Um eine sichere Anwendung zu erstellen, braucht es viele Faktoren.

---

**ACHTUNG** Nehmen Sie die Sicherheit als Kernpunkt in Ihre Projektplanung auf. Es ist wesentlich günstiger, dies in der Planung zu berücksichtigen, als Ihre Produkte später mit Service Packs zu reparieren.

---

## Infrastruktur

Eine sichere Infrastruktur ist ein wichtiges Glied in der Sicherheitskette. Wir sprechen hier nicht von Hardware oder Cluster-Systemen sondern nur von der verwendeten Software. Nach dem heutigen Stand der Technik ist Windows 2003 die sicherste Basis für den Betrieb eines Webservers. Zur Drucklegung dieses Buches sind zum darin enthaltenen Internet Information Server (IIS) in der Version 6 keine Sicherheitslücken bekannt. Darüber hinaus sollten immer die aktuellsten Service Packs eingespielt werden. Die beste Infrastruktur hilft davon abgesehen nichts, wenn mit den Passwörtern bzw. den Benutzerkonten nachlässig umgegangen wird.

Der anonyme Zugriff auf Ihre ASP.NET-Anwendung erfolgt mit dem lokalen Benutzer *ASPNET*. Dieser ist nur mit minimalen Rechten ausgestattet, die z.B. kein Schreiben ins Dateisystem erlauben. Speziell die Zugriffe auf die Datenbank dürfen ebenfalls nur mit minimalen Rechten durchgeführt werden. Ein Konto, das von einer Anwendung benutzt wird, um auf eine Datenbank zuzugreifen, braucht nur im seltensten Fall die Rechte, um eine Tabelle zu löschen. Also verzichten Sie am besten auf die Verwendung solcher mächtig ausgestatteter Konten, wie es z.B. der *sa* beim SQL-Server darstellt.

## Architektur

Je nach verwendeter Technologie ist diese während der Entwicklung mehr oder weniger anfällig für Fehler, die die Sicherheit beeinträchtigen. C++ birgt ein vergleichsweise großes Potential, durch die Verwendung von Zeigern Sicherheitslücken zu erzeugen – Visual Basic 6 eher wenig. Keine Sprache ist allerdings gänzlich davor gefeit. Bei der Webentwicklung passieren hauptsächlich zwei Arten von Fehlern:

### SQL-Injection

Beim Zusammensetzen von SQL-Kommandos durch Zeichenkettenoperationen kann unbeabsichtigt oder auch böswillig ein SQL-Befehl erzeugt werden, der Schaden verursachen kann. Wenn dies vorsätzlich vom Benutzer durch Eingaben provoziert wird, spricht man von *SQL-Injection*, da dabei SQL Befehle »injiziert« werden. Dagegen hilft, ausschließlich parametrisierte Abfragen zu verwenden. Auch gespeicherte Prozedu-

ren (*Stored Procedures*) arbeiten mit Parametern und sind damit vor schädlichen SQL-Kommandos sicher. ADO.NET als Datenzugriffsschicht schützt Ihre Anwendung dann ganz automatisch vor Schlimmeren.

## Cross Site Scripting

Böswillige Benutzer wissen, dass dynamische Webseiten Benutzereingaben verarbeiten, speichern und wieder anzeigen. Wenn nun HTML, JScript oder anderer Code Benutzereingaben verarbeitet, kann das unterschiedliche schädliche Auswirkungen haben, weil sich unter bestimmten Bedingungen HTML in die Benutzereingaben einbauen lässt und dann beim möglichen wiederholten Anzeigen der ursprünglichen Eingabe auch als HTML ausgeführt wird. Dieses Prinzip nennt man *Cross Site Scripting*.

Prüfen Sie deshalb alle Benutzereingaben, bevor diese verarbeitet werden. Trauen Sie keiner Eingabe. Das beste Prinzip ist, erst einmal grundsätzlich alles zu verbieten und schließlich nur das zu erlauben, was man unbedingt benötigt. Sehr hilfreich in diesem Zusammenhang ist die Funktion *Server.HtmlEncode*, die beispielsweise HTML Steuerzeichen in entsprechende Entitäten umwandelt. Die folgende Tabelle stellt weitere, in diesem Zusammenhang nützliche Methoden vor:

Methode	Verwendung
<i>Server.HtmlEncode</i>	Mit dieser Funktion wird ein HTML String kodiert. Aus <code>&lt;SCRIPT&gt;</code> wird somit beispielsweise <code>&amp;lt;SCRIPT&amp;gt;</code> ; Mit der zweiten Überladung kann direkt in ein <i>TextWriter</i> -Objekt geschrieben werden.
<i>Server.UrlEncode</i>	Damit wird eine URL und damit auch der Querystring kodiert.
<i>Server.UrlPathEncode</i>	Kodiert nur den Pfad-Bestandteil einer URL.

**Tabelle 4.1** Serverkodierungsfunktionen

Seit ASP.NET 1.1 ist es nicht mehr möglich, Code für HTML-Elemente in *TextBox*-Steuerelemente zu schreiben. Das müsste erst durch die *Page*-Direktive der Seite mit *ValidateRequest=»false«* aktiviert werden.

**ACHTUNG** Benutzereingaben stammen nicht nur aus Textboxen. Auch Cookies und QueryStrings werden oft von der Programmlogik ausgewertet und stellen damit eine Möglichkeit zur Beeinflussung der Logik dar.

## Betrieb

Wenn Ihre Anwendung im Einsatz ist, lauern Gefahren an allen Ecken und Enden. Administratoren, die in Unkenntnis der Fakten z.B. Schreibzugriff auf das *Bin*-Verzeichnis gewähren, oder Attacken, die sich bislang unentdeckte Sicherheitslöcher zu Nutze machen, sind nur einige der möglichen Bedrohungen. Am besten hilft dauerndes Überwachen der Anwendung. Wenn Zustände außerhalb der Norm auftreten, kann sich der Entwickler per E-Mail darüber informieren lassen. Dies trägt auch zur Stabilität der Anwendung bei. Einige Systemadministratoren oder Entwickler favorisieren auch noch zusätzlich eine Firewall am bzw. vor dem Webserver.

Für das Überwachen der Anwendung stehen verschiedene Methoden zu Verfügung:

### Protokolldateien (Logfiles)

In den Protokolldateien werden alle http-Zugriffe auf den Webserver aufgezeichnet. Das übliche W3C-Format ist textbasiert und auch für Menschen lesbar. Sinnvoll ist der Einsatz eines Analysewerkzeugs, das aus

Massendaten reduzierte Übersichtsdaten macht. Microsoft bietet ein sehr mächtiges und kostenfreies Kommandozeilen-Werkzeug, den *Logparser*.

## Ereignisprotokolle

In der Ereignisanzeige eines Windows-Systems werden fehlerhafte Zustände dauerhaft archiviert. Wenn z.B. ein ASP.NET-Arbeits-Prozess neu gestartet wurde, finden Sie dort einen entsprechenden Eintrag.

## Leistungsüberwachung (Performance-Monitor)

Schon seit Urzeiten von Windows NT gibt es die Leistungsüberwachung. Damit können Leistungsindikatoren visuell dargestellt werden. Dabei gibt es spezielle Indikatoren für ASP.NET, um z.B. die Cache-Leistung zu überwachen.

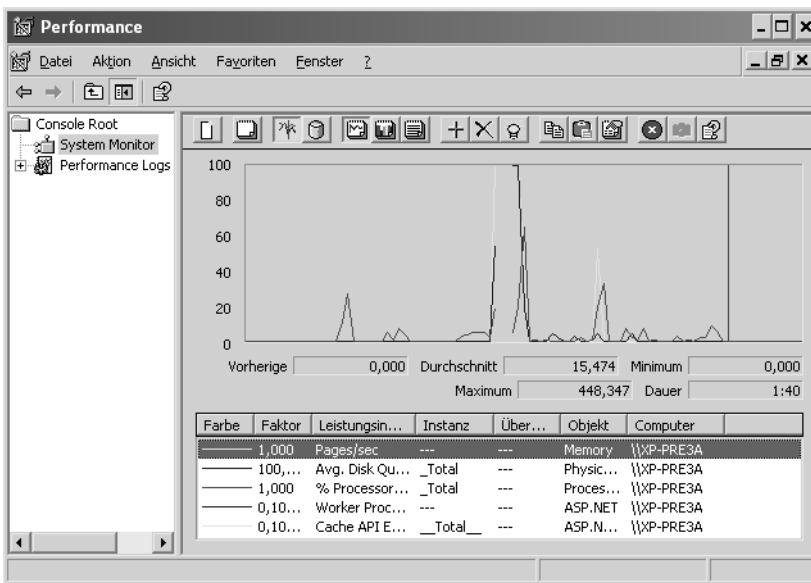


Abbildung 4.1 Der Performance-Monitor hilft bei der Überwachung

## Drittanbieter

Für einen sicheren und stabilen Betrieb eines Webserver gibt es natürlich noch eine Menge nützlicher Werkzeuge von Drittanbietern.

## Healthmonitor

In der Datei *web.config* kann im Bereich *Healthmonitoring* eine eigene Art der Überwachung aktiviert werden. Dabei werden in einem festzulegenden Intervall Systemdaten in eine auch zu konfigurierende Datenquelle wie SQL Server geschrieben. Per eigenem Code oder Werkzeug können diese Daten dann auch von entfernten Computern aus ausgewertet werden. Kapitel 9 liefert dazu noch ein detailliertes Beispiel.

## Global.asax

Die Datei *global.asax* stellt eine Art globaler Anwendungsklasse dar. In dieser findet sich eine Ereignismethode *Application\_Error*. Diese wird immer aufgerufen, wenn in der Anwendung ein unbehandelter Fehler auftritt. Der Fehler kann dann per *GetLastError* ausgelesen werden und z.B. per E-Mail an den Webmaster versandt werden.

## Verschlüsseln von Verbindungszeichenfolgen

Verbindungszeichenfolgen (*Connection Strings*) werden oft in der Datei *web.config* abgelegt. Jedenfalls ist das ein sehr sicherer und einfach zu verwendender Speicherplatz. Noch sicherer ist es allerdings, Zeichenketten zu verschlüsseln. Dies geht jetzt mit dem Element *<protectedData>* in der *web.config*.

Allerdings lässt sich wiederum auch nicht alles verschlüsseln. Die folgenden Elemente sind davon ausgenommen.

- *<processModel>*
- *<runtime>*
- *<mscorlib>*
- *<startup>*
- *<system.runtime.remoting>*
- *<protectedData>*
- *<satelliteassemblies>*
- *<cryptologySettings>*
- *<cryptoNameMapping>*
- *<cryptoClasses>*

Eine verschlüsselte *web.config*-Datei sieht dann etwa wie folgt aus:

```
<connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
  <EncryptedData Type="http://www.w3.org/2001/04/xmenc#Element"
    xmlns="http://www.w3.org/2001/04/xmenc#">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#tripleDES-cbc" />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <KeyName>Rsa Key</KeyName>
        </KeyInfo>
      </EncryptedKey>
    </KeyInfo>
  </EncryptedData>
  <CipherValue>TN6JWkwna0tF8f1u5FFmaX0nIbVQzsNOZiToaWv5ei31W3Rdk19+XaUUXp20HPtguk81GxGGNCJjdtxSEZtjNrB0gNr/1fMeL4zPTDMiQKoAecYBdb1NGfK4KMbaXzfb1DzuTCW0Dqr/JAVPH4+5tr7NRNhwREjBo4YiK04q60g=</CipherValue>
  </CipherData>
</EncryptedKey>
</KeyInfo>
<CipherData>
  <CipherValue>JURACU8prog3Rzkhg9G8qdLam65ssQ8pMFqCM9iP7a8=</CipherValue>
</CipherData>
</EncryptedData>
</connectionStrings>
```

**Listing 4.1** Verschlüsselter Bereich ConnectionStrings

Natürlich kann das kein Mensch mehr sinnvoll pflegen, deshalb gibt es, wie so oft, ein Werkzeug dafür. Genau gesagt handelt es sich dabei um das Werkzeug *aspnet\_regiis*, das für diese Aufgabe erweitert wurde. Mit folgender Syntax können Sie so die Verbindungszeichenfolgen in Ihrer *web.config*-Datei verschlüsseln. Rufen Sie dazu in der Kommandozeile den Befehl mit den nachfolgenden Parametern auf.

```
aspnet_regiis -pe connectionStrings -app virtual-Pfad
```

Dies setzt allerdings eine im IIS konfigurierte Anwendung voraus. Mit der Option *-pef* (statt *pe*) kann direkt der physikalische Pfad als Parameter angegeben werden. Mit *-pdf* kann dies wieder entschlüsselt werden.

```
aspnet_regiis -pef connectionStrings physikalischer-Pfad
```

---

**ACHTUNG** Wenn Sie Ihre Web Anwendung in einer Webfarm betreiben, müssen Sie für das Verschlüsseln einen gemeinsamen Schlüssel verwenden. Auch dieser kann mit *aspnet\_regiis* erzeugt und in eine XML-Datei exportiert werden. Dazu verwenden Sie die Option *-px*.

---

Das Wichtigste dabei ist: Ihre Anwendung funktioniert ohne Änderungen am Code genauso wie bisher.

## Codezugriffssicherheit (Code Access Security)

Wenn ein Win32 Programm ausgeführt wird, läuft dieses mit bestimmten Benutzerrechten. Da viele Benutzer sich selbst gerne als Administrator sehen, können dann mit diesen Rechten ausgestattet böswillige Programme auch entsprechenden Schaden anrichten. Seit .NET ist es möglich, Code bestimmte Rechte zuzuweisen, die unabhängig vom jeweils angemeldeten Benutzer und den daraus resultierenden Rechten sind. Sogar der Ursprungsort des Codes beeinflusst seine Rechte. So hat ein Programm, dessen Quelle die computereigene Festplatte darstellt, mehr Möglichkeiten als eines von einer unbekanntenen Webseite.

Darauf möchte ich in diesem Rahmen jedoch nicht weiter eingehen, da Code Access Security in der Hauptsache bei Windows.Forms-Anwendungen eine Rolle spielt und weniger im Web. Der Standardbenutzer einer Website hat ohnehin die geringste Berechtigungsstufe und darf eigentlich gar nichts auf dem eigentlichen WebServer.

## Schichten

Ich werde in diesem Buch nur wenige Worte zur mehrschichtigen Softwareentwicklung verlieren. Es gibt dazu weiterführende Literatur, die sich genau mit dieser Thematik beschäftigt. Aus den negativen Erfahrungen der Client-Server-Programmierung wurde der Schluss gezogen, dass eine zusätzliche Schicht für Datenzugriff und Geschäftslogik zu verwenden ist. Eine ASP.Net Anwendung ist eigentlich schon per Definition mehrschichtig. Ein Teil der Logik liegt im Datenbanksystem, hauptsächlich in Form von Stored Procedures. Die nächste Schicht ist der Code der ASP.NET Anwendung um z.B. eine Datenbankabfrage durchzuführen. Der Browser stellt schlussendlich die Anzeigeschicht dar, in der z.B. auch noch Logik zur Eingabepfung enthalten sein kann.

Die Logik in einer ASP.NET Anwendung lässt sich noch in weitere Schichten unterteilen. Häufig werden hierzu eine Datenzugriffsschicht (DAL) und eine Geschäftslogik (BOL) getrennt.



Ein derartiges Schichtenmodell ist für viele Webanwendungen nicht unbedingt nötig. Es kann direkt mit der Datenquelle gearbeitet werden und sämtlicher Code in der ASPX-Seite untergebracht werden, der nötig ist. Dies kann eine deutliche Reduzierung der Entwicklungskosten der Anwendung zur Folge haben.

Auf der anderen Seite gibt es zwei wesentliche Gründe für mehrschichtige Webanwendungs-Entwicklungen. Das ist zum einen die Performance und zum anderen die Möglichkeit der Wiederverwendbarkeit. Speziell wenn es um das Performance-Argument geht sollten Sie beachten, das eine durchschnittlich umfangreiche ASP.NET Anwendung auf einem modernen Server leicht ein paar hundert Anfragen pro Sekunde bearbeiten kann. Für gestiegene Anforderungen kann der Durchsatz per NLB (Network Load Balancing) mit mehreren Servern nahezu linear erhöht werden. Eine wesentlich wichtigere Beschränkung ist da wahrscheinlich die zur Verfügung stehende Bandbreite.

Ein Beispiel: Wenn Ihre Webseite einen Umfang von 100 KBytes hat – was schon mit zwei Bildern schnell erreicht ist – und 100 Abrufe pro Sekunde stattfinden, ergibt das eine benötigte Bandbreite von 1 MByte/s (also 8 MBit) bei einer Ladezeit von 10 Sekunden. Übliche Leitungsgrößen liegen zum Vergleich bei 2 MBit – die Leitungsgröße stellt also eher den Flaschenhals dar, als die Performance der Applikation selbst.

Performance ist also in diesem Szenario ein weniger gewichtiges Argument für mittlere Schichten bei einer ASP.NET Anwendung.

Es gibt aber auch andere Anwendungsfälle in denen dies von Relevanz ist. Wenn große Datenmengen dargestellt werden sollen, ist es nicht nötig die kompletten Daten zu lesen. Genau dies tun die ASP.NET Objekte aber. Hier kann ein Datenzugriffsobjekt Funktionalitäten beinhalten die das Lesen und vor allem das Seitenweise darstellen optimieren.

Wenn die Datenzugriffsschicht in einem eigenen Prozess oder gar Server laufen soll, müssen komplexe Technologien wie .NET Remoting eingesetzt werden, um die Objektkommunikation sicherzustellen. Wenn dies nötig ist, sollten Sie auch besser Web Services innerhalb einer Web-Farm verwenden, weil diesen gemeinhin eine größere Zukunftssicherheit in einer service-orientierten Architektur (SOA) zugebilligt wird.

Bleibt das zweite Argument der Wiederverwendbarkeit von Code. Damit soll die Pflege und Weiterentwicklung kostengünstiger werden, indem man vorher in ein entsprechendes Konzept investiert. Auch hier gibt es vernünftige Pro- und Contra-Standpunkte gleichermaßen. Oft steht zwar der hehre Gedanke des »Code reusing« im Raum, aber beim nächsten Projekt werden Problemlösungen erfahrungsgemäß doch wieder ganz anders angegangen. Aus der Aufgabe ein Einfamilienhaus zu bauen, wird deshalb ein 40m tiefes Betonfundament geplant, weil man ja irgendwie mal auf die Idee kommen könnte, aus dem Einfamilienhaus ein Hochhaus zu machen.

Nach der Meinung des Autors ist in einem wesentlichen Prozentsatz der ASP.NET-Softwareprojekte keine explizite mittlere Schicht notwendig. Das sollten Sie bedenken, wenn Sie die höheren Kosten einer zusätzlichen mittleren Schicht in Ihrem Projekt budgetieren.

Ergo werden in Summe ASP.NET Projekte schneller fertig und damit kostengünstiger, wenn man den pragmatischen Ansatz wählt. In diesem Buch wird ebenfalls entsprechender Code gezeigt, der mit minimalem Aufwand das gewünschte Ergebnis erzielt.

Einen ganz interessanten Aspekt bringt in diesem Zusammenhang das *App\_Code*-Verzeichnis eines Web-Site-Projekts. Selbstverständlich dürfen und sollen Sie Code, der sich als nützlich erwiesen hat, in eine Klasse refaktorisieren und einfach im nächsten Projekt wieder verwenden können. Dazu muss dieser Code in einer VB oder CS Datei nur ins *App\_Code*-Verzeichnis kopiert werden. Diese werden automatisch mitkompiliert. Aber auch andere Arten von Code – wie die *WSDL*-Dateien eines Web Services – können im Code-Verzeichnis abgelegt werden, und es wird daraus automatisch eine Proxy-Klasse erzeugt.

Ganz ähnlich funktioniert es auch mit typisierten *DataSet*-Objekten in Form von XSD-Dateien.

Bisher wurden für solche Klassenbibliotheken oft eigene DLLs kompiliert und diese ins BIN-Verzeichnis der Anwendung gelegt. Dies ist auch weiter möglich. Einfacher erstellen Sie aber Ihre Geschäftsobjekte im *App\_Code*-Verzeichnis. Wenn Sie in Klassen, die in diesem Verzeichnis liegen Zugriff auf seitenspezifische Objekte wie Cache oder Response brauchen, hilft *HttpContext.Current*.

```
Dim myid As String = HttpContext.Current.Request.QueryString("id")
```

## Statusinformationen

Der Lebenszyklus einer Webseite in der ASP.NET-Engine ist kurz. Eine Eintagsfliege wird dagegen richtig alt. Nachdem der Code verarbeitet wurde und ASP.NET die erzeugten Seiten versendet hat, werden alle Objekte, aus denen diese Seite bestand, zerstört. Dabei brauchen wir Entwickler aber durchaus noch bestimmte Informationen von der zuvor aufgerufenen Seite, etwa welcher Benutzer angemeldet oder nach welchem Kriterium die Datenliste eben noch sortiert war. Diese Informationen werden als Statusinformationen bezeichnet. Eigentlich beginnt die Statusübergabe schon mit einer Abfragezeichenfolge, etwa:

```
Seite.aspx?id=1
```

Allgemeine und umfangreichere Daten werden mithilfe von Session-Variablen der aktuellen Session zugeordnet, etwa:

```
Session("variable")=irgendwas
```

Die Session-Informationen einer jeden Benutzersession werden serverseitig gespeichert. Allein eine Session-ID wird vom Browser von Aufruf zu Aufruf weitergereicht. Dabei können auch umfangreiche Objekte wie *DataSets* in einer Session-Variablen gespeichert werden. Beim Abrufen so gespeicherter Objekte muss dann allerdings eine Typenumwandlung (ein *Type-Casting*) in den passenden Datentyp stattfinden. In VB heißt der entsprechende Befehl dazu *CType*. Die folgende Anweisung weist beispielsweise ein in einer Session-Variablen gespeichertes *DataSet*-Objekt einer typkonformen Objektvariablen zu:

```
ds=CType(dataset,Session("Datasetvar"))
```

Ein wesentlicher Nachteil von Session-Variablen, vor allem bei exzessivem Gebrauch, ist der bisweilen große Speicherbedarf am Server.

Die zweite Art von Statusinformation wird von bestimmten Webserver-Controls direkt in der Seite gespeichert. Dazu wird ein verstecktes Feld (ein so genanntes *HiddenField*) zusammen mit dem *ViewState*-Objekt verwendet, in dem die Informationen und Daten aller Controls der Seite liegen. Es ist übrigens zwar möglich, aber eher unüblich, direkt mit dem *ViewState*-Objekt zu arbeiten, etwa wie im folgenden Beispiel.

```
ViewState("variable")=irgendwas
```

Der Speicherbedarf dieses *HiddenFields* in der Webseite kann unter Umständen schnell anwachsen, da beispielsweise bei einer Liste auch die Listendaten im *ViewState*-Objekt gespeichert werden. Übrigens ist *Enable-*

*ViewState* auch eine Eigenschaft beinahe eines jeden Webserver-Steuerelements, sie steuert, ob die Eigenschaften des Steuerelements automatisch in der Website gespeichert werden sollen (*EnableViewState=true*) oder nicht. Diese Eigenschaft lässt sich über das Eigenschaftfenster zur Entwurfszeit, aber auch programmtechnisch ein- und ausschalten.

Das Ausschalten verringert zwar die Datenmenge, aber Sie müssen dann auch Funktionalitätseinbußen hinnehmen: So funktioniert z.B. das Paging im *DataGrid*-Steuerelement in dem Fall nicht mehr.

Allerdings stellt die große Datenmenge, die bei eingeschalteter *ViewState*-Eigenschaft anfällt, nicht nur bei mobilen Geräten durchaus ein Problem dar. Hinzu kommt, dass der *ViewState*-Speicher in der generierten Web-Seite von böswilligen Angreifern decodiert werden kann und diese so an etwaige vertrauliche Daten gelangen.

Mit ASP.NET 2.0 ist es nun möglich den *ViewState* zu verschlüsseln. Dazu wird das Attribut *ViewStateEncryptionMode* in der Page-Direktive gesetzt.

Session-Variablen haben diese Problematik nicht, da diese ja am Server gespeichert werden. Die einfachste und sicherste Methode ist, die Statusinformation im gleichen Prozess zu speichern. Dazu setzen Sie im Abschnitt *SessionState* den entsprechenden Modus, etwa:

```
<sessionState
  mode="InProc"
  ...
```

**Listing 4.2** Ausschnitt aus der Datei *web.config* im Bereich *SessionState*

Gegenüber ASP.NET 1.x ist ein neuer Wert für das Attribut *mode*, nämlich *Custom* hinzugekommen. Die folgende Tabelle zeigt alle möglichen *SessionState*-Moduseinstellungen:

Wert	Bedeutung
<i>Custom</i>	Erlaubt es, eine eigene Session-Statusverwaltung zu implementieren – vor allem, um abweichende Speicherorte wie eine Oracle Datenbank zu ermöglichen. Aber auch eigene Session-IDs lassen sich so verwirklichen.
<i>InProc</i>	Session-Variablen werden im Speicherkontext der Anwendung abgelegt. Also im Cache des ASP.NET Arbeitsprozesses.
<i>Off</i>	Die Session-Verwaltung wird abgeschaltet. Daraus resultiert die schnellste Methode.
<i>SQLServer</i>	Statusinformationen werden in einer SQL Server-Datenbank abgelegt. Das hat zwei Vorteile. Die Session-Daten werden verlässlich gespeichert, und im Fall des Einsatzes einer Web-Farm können mehrere Webserver die gleichen Statusinformationen nutzen. Da jede Anfrage von einem anderen Server bedient werden kann, ist dies nämlich erforderlich.
<i>StateServer</i>	Die Session-Information wird in einem eigenen Statusserver abgelegt. Dies ist der <i>aspnet_state.exe</i> Prozess.

**Tabelle 4.2** SessionState-Modusattribute

### Custom Session-Status (benutzerdefinierter Session-Status)

Um die Session-Daten in einer eigenen Datenbank ablegen zu können, muss der Modus der Statusinformation auf *Custom* gesetzt werden. Darüber hinaus muss natürlich auch ein eigener Provider zur Verfügung gestellt werden. Dazu stehen zwei mögliche Wege zur Verfügung: Zum einen können Sie ein komplett neues Status-Modul entwickeln oder einfach einen Teil vom vorhandenen Modul nutzen. Zu diesem Zweck existiert ein Session State ID Modul, mit dem Sie eigene Session-IDs erzeugen können.

Die Basisklasse für den Session Provider ist *SessionStateStoreProviderBase*.

## Steuerelement-Status

Speziell wenn Sie UserControls entwickeln, die auf den Viewstate angewiesen sind, wäre es fatal, wenn der Entwickler oder Administrator die Viewstate-Funktionalität für die Seite abschaltet. Das kann per *Page*-Direktive in der ASPX-Seite oder in der *web.config* zentral erfolgen.

```
<%@ Page enableviewstate="false"%>
```

**Listing 4.3** Für die Seite deaktivierter Viewstate

```
<pages enableViewState="false"></pages>
```

**Listing 4.4** In der Datei *web.config* deaktivierter Viewstate

Deshalb kann nunmehr für Controls ein eigener Status verwaltet werden. Dafür wird auch das *HiddenField* mit dem Namen `__VIEWSTATE` in der Seite verwendet.

---

**ACHTUNG** Falls Sie die genaue Größe von *Viewstate* und *Controlstate* interessiert, können Sie *Page Tracing* verwenden. Im Trace gibt es dafür eine eigene Spalte, in der für jedes Control die verwendete Größe angezeigt wird.

---

## Page-Status

Speziell bei mobilen Geräten muss die Datenübertragung auf ein Minimum reduziert werden – die Verbindungen sind in der Regel nämlich langsam und teuer. Dazu lässt sich der Viewstate auf dem Webserver speichern. Das reduziert den Overhead erheblich. Weitere Details finden sich in der Dokumentation unter *PageStatePersister*.

## Page-Variablen

Obwohl der Lebenszyklus einer Webseite sehr kurz ist, können doch an verschiedenen Stellen Codefragmente an der Ausführung beteiligt sein. Das kann im *App\_Code* eine Klasse sein oder *HttpModule* oder *HttpHandler*. Um dabei Information konsistent verfügbar zu halten, kann man sich des *HttpContext* Objektes bedienen. Dies erlaubt z.B. in einer externen Klasse den Zugriff auf das Page Objekt.

```
HttpContext.Current.Request.QueryString("id")
```

Zusätzlich kann man in der dort auch vorhandenen *Items*-Auflistung Variablen ablegen. Es handelt sich dabei wie üblich um eine untypisierte Speicherung die per Schlüssel-/Wert-Paar geschieht.

```
HttpContext.Current.Items.Add("Key", "Wert")
```

So kann man sehr kurzfristig und dabei performant Variablen zur Verfügung halten. Die Anwendungsbereiche sind sehr weit gefächert.

# Entwickeln im Team

Da ASP.NET 2.0 auf die Projektmappen-Dateien verzichtet, ist es nun einfacher geworden, im Team zu entwickeln.

Ein besonderer Evolutionsschritt ist dabei Visual Studio Team System. Dabei handelt es sich um eine ganze Familie von Werkzeugen für die Softwareentwicklung.

Dabei geht es um Entwurf, Test und Archivierung. So wird das altbekannte Quellcodemanagement-System *Source Safe* durch eine webbasierte, wesentlich mächtigere Quellcodeverwaltung ersetzt. Source Safe existiert weiter und bietet auch einige neue Funktionen an.

Speziell die beinhalteten Werkzeuge für Last Test und Funktionstest sind für ein professionelles Web Projekt sehr hilfreich. Dafür empfehle ich weiterführende Literatur.



## Kapitel 5

# Webserver-Steuererelemente

### **In diesem Kapitel:**

Generelle Änderungen	104
Änderungen an bestehenden Steuererelementen	113
Änderungen am HTML-Server-Steuererelement	121
Neue Webserver-Steuererelemente	122
Neue HTML-Server-Steuererelemente	137

Erst mit Webserver-Steuerelementen wird die Web-Entwicklung so richtig einfach. Ganz wie es ein Visual Basic-Entwickler seit Jahren gewohnt ist, zieht man einfach ein passendes Steuerelement aus der Werkzeugleiste auf das Webseiten-Formular.

Was mit Design-Time-Steuerelementen in ASP ein erster holpriger Ansatz war, ist erst in ASP.NET mit den Webserver-Steuerelementen richtig praktisch zu verwenden. Man unterscheidet dabei in zwei unterschiedlichen Kategorien: Die HTML-Server-Steuerelemente und die Webserver-Steuerelemente.

Die HTML-Server-Steuerelemente bilden 1:1 die HTML-Elemente ab und erzeugen zur Laufzeit genau diesen HTML-Code im Browser. So wird aus dem *Input Server Steuerelement* das HTML-*INPUT*-Element. Webserver-Steuerelemente erzeugen dagegen je nach Browser, Benutzereinstellungen, Betriebssystem und anderen Faktoren unterschiedlichen HTML-Code.

Mit ASP.NET 2.0 gibt es nun eine Menge neuer Steuerelemente, die dem Webentwickler Standardaufgaben abnehmen. Bereits bekannte und bewährte Steuerelemente wurden um nützliche Eigenschaften erweitert. Auf alle Fälle ist Code, der für ASP.NET 1.x geschrieben wurde, auch in ASP.NET 2.0 lauffähig, da alle Steuerelemente abwärtskompatibel sind.

## Generelle Änderungen

Die Änderungen in ASP.NET 2,0 sind so umfangreich, dass leicht einige übersehen werden können. Jedes einzelne der mehr als 70 Webserver-Steuerelemente hat mindestens 50 Eigenschaften.

Für die Entwicklung von Webseiten für mobile Geräte, müssen wie gehabt die aus ASP.NET 1.x bekannten mobilen Steuerelemente verwendet werden.

Ein Visual Studio 2003 Web Projekt das mit Visual Studio 2005 geöffnet wird, wird automatisch nach ASP.NET 2.0 portiert.

## Accessibility

Microsoft legt schon seit langer Zeit großen Wert darauf, dass auch behinderte Personen ihre Produkte verwenden können. Dies betrifft vor allem sehbehinderte Personen. Diese haben spezielle Geräte, die es ihnen z.B. ermöglichen, sich Webseiten vorlesen zu lassen. Dazu muss sich der Webentwickler allerdings an bestimmte Regeln halten, um solchen Geräten zu erlauben, die Inhalte richtig zu interpretieren. Dabei geht es beispielsweise um Bilder, die per Text beschrieben werden müssen. Daten, die in Tabellen vorkommen, müssen anders behandelt werden als Formatierungstabellen.

Natürlich ist die Zielgruppe sehr klein, aber neben der sozialen Verantwortung gibt es auch noch Gesetze, die sich auf Barrierefreiheit beziehen. Es ist durchaus denkbar, dass Abmahnvereine dies einmal mehr zum Anlass nehmen werden, mit Ihrer Website Geschäfte zu machen. Wenn Sie bereits mit ASP.NET 1.0 Webseiten erstellt haben, sind Sie diesbezüglich nicht sicher. Erst ASP.NET 2.0 nimmt darauf umfassend Rücksicht. Im Folgenden werden die erweiterten Attribute der Web Steuerelemente beschrieben, die dabei helfen.

## Caption

Mit dem *Caption*-Element erhält eine HTML-Tabelle eine Überschrift, die im Browser dargestellt wird und von entsprechenden Lesegeräten ausgewertet werden kann. Dabei handelt es sich um keine eigentliche



Erweiterung von ASP.NET 2.0, es ist aber beim Anzeigen von Daten für barrierefreie Webseiten wichtig. Tabellen, die dem Zweck der Seitenformatierung dienen, dürfen keine Überschrift haben.

```
<table summary="Das ist eine Tabelle" >
  <caption>Das ist die Caption der Tabelle
  </caption>
  <tr>
```

Die Bezeichnung wird auch im Browser angezeigt.

```
<table id="ctl00_ContentPlaceholder1_Table1" rules="all" border="1">
  <caption align="Bottom">Ihre Daten</caption>
```

**Listing 5.1** Erzeugter HTML-Code des Caption-Attributs

Wenn die Beschreibung durch *Caption* nicht ausreicht, kann eine detaillierte Erklärung im Summary-Attribut erfolgen, das nicht im Browser dargestellt wird.

### Caption im Table Steuerelement

Das Webserver-Steuerelement *Table* unterstützt *Caption* ebenfalls.

```
<asp:Table ID="Table1" runat="server" Caption="Das ist die Überschrift" CaptionAlign="Right">
```

Auch in anderen Steuerelementen, die tabellarische Daten erzeugen, ist *Caption* vorhanden, z.B. im *GridView-Steuerelement*.

### TableHeaderRow

Darüber hinaus lässt sich noch eine spezielle Kopfzeile (*TableHeaderRow*) deklarieren, die in HTML mit *<th>* (*Table Header*) dargestellt wird.

```
<asp:TableHeaderRow> <asp:TableHeaderCell>Daten1</asp:TableHeaderCell></asp:TableHeaderRow>
```

**Listing 5.2** Eine Spalte und eine Reihe im Table Header

### TableFooterRow

Ganz ähnlich wird der Fußbereich mittels *TableFooterRow* erstellt.

```
<asp:TableFooterRow>
  <asp:TableCell>ende</asp:TableCell>
</asp:TableFooterRow>
```

**Listing 5.3** Fußzeile der Tabelle

### GenerateEmptyAlternateText

Wenn ein Bild nur aus optischen Gründen vorhanden ist, macht es Sinn, den alternativen Text leer zu lassen. Allerdings wird dabei explizit *alt=>><<* gesetzt. Dies wird erreicht durch das Attribut *GenerateEmptyAlternateText*.

```
<asp:Image ID="Image1" Runat="server" GenerateEmptyAlternateText="True" />
```

In jedem anderen Fall soll das *Alt*-Attribut eine Beschreibung des Bildes enthalten. Die Eigenschaft im *Image*-Steuerelement lautet *AlternateText*.

## DescriptionURL

Mit dem Attribut *DescriptionURL* verweist eine Grafik auf eine externe URL, die eine ausführlichere Beschreibung des Bildes beinhalten kann. In der HTML-Seite wird dann ein Eintrag *longdesc=>beschreibung.htm<<* erzeugt.

## UseAccessibilityHeader

Einige Webserver-Steuerelemente haben nun die zusätzliche Eigenschaft *UseAccessibilityHeader*, z.B. das bereits aus ASP.NET bekannte *DataGrid*-Steuerelement. Damit wird die erste Reihe, also die Kopfzeile, per *HTML-<TH>*-Element im Browser dargestellt.

## Eingabe

Der Benutzer soll es so einfach wie möglich haben, seine Daten in Ihre Webanwendung einzutragen. Dabei gab es bisher einige Stolpersteine wie das automatische Ausfüllen eines Formulars, die Eingabeprüfung oder das Setzen des Fokus in ein bestimmtes Feld. All das sind Dinge, die sich jetzt schon lösen lassen, aber zusätzlichen Aufwand bedeuten. ASP.NET 2.0 liefert nun ein paar Tricks, die helfen sollen, dies einfacher zu machen.

## AssociatedControlID

Ein einfacher Benutzerdialog sollte auch die Möglichkeit bieten, per HotKeys im Formular zu navigieren. Beispielsweise wird durch Drücken von **Alt+D** der Fokus direkt in ein Formularfeld gesetzt. Grundsätzlich enthalten viele Webserver-Steuerelemente deshalb die Eigenschaft *AccessKey*. Dort muss nur der Auslösebuchstabe eingetragen werden. *AccessKey* ist eine Standard-HTML-Eigenschaft.

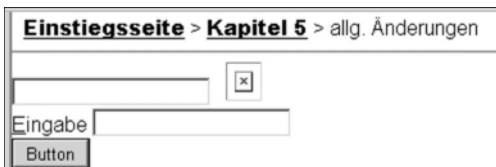
### ACHTUNG

Manche Buchstaben sind vom Browser reserviert und können deshalb nicht als *AccessKey* verwendet werden.

Allerdings sieht dann der Benutzer keinen Hinweis auf diesen HotKey. Also muss ein Label her, in dem einfach per HTML-Syntax der betreffende Buchstabe unterstrichen wird. Die Verbindung zwischen Label und Textbox wird dann durch das Attribut *AssociatedControlID* hergestellt.

```
<asp:Label ID="Label1" AccessKey="E" Runat="server" Text="<u>E</u>ingabe"
AssociatedControlID="TextBox2"></asp:Label>
```

**Listing 5.4** In einem Label wird ein Hotkey definiert



**Abbildung 5.1** Mit **Alt** + **E** kann dieses Eingabefeld direkt erreicht werden

## AutoCompleteType

Wenn Benutzer beispielsweise ihre Adresse eingeben sollen, ist es hilfreich die Tipparbeit gering zu halten. Deshalb haben alle modernen Browser einen Mechanismus, der sich eingegebene Werte passend zum Feldnamen merkt. Deshalb sollten auch solche Textboxen nicht mit *txtOrt*, sondern wie allgemein üblich mit *Ort* benannt werden.

**HINWEIS** Wenn Sie mit Masterseiten arbeiten, erzeugt ASP.NET andere ID's und Namen im HTML-Code, als Sie definiert haben. Aus Ort wird so z.B.:

```
<input name="ct100$ContentPlaceHolder1$Ort" type="text" id="ct100_ContentPlaceHolder1_Ort" />
```

Damit funktioniert das übliche Auto-Vervollständigen (*AutoComplete*) nicht mehr.

Eine weitere Möglichkeit, dem Benutzer zu helfen, besteht in dem Zugriff auf das Profil des Benutzers. Dieses wird im Browser gespeichert. Die Feldnamen sind deshalb auch vordefiniert und werden per IntelliSense angezeigt.

```
<asp:TextBox ID="ort" Runat="server" AutoCompleteType="HomeCity"></asp:TextBox>
```

**Listing 5.5** Formular-Ausfüllhilfe für den Benutzer

Ob der Benutzer dort Informationen hinterlegt hat, ist allerdings nicht sichergestellt.

**ACHTUNG** Im Form-Element kann *AutoComplete* mit dem gleichnamigen Attribut abgeschaltet werden. Das ist speziell bei Anmeldedialogen sinnvoll.

## PostBackURL

Es ist jetzt auch möglich, die per http-Post-Kommando übermittelten Formular-Werte auf anderen ASPX-Seiten auszuwerten. Dies wird als *Cross Page Posting* bezeichnet. Dafür müssen die Steuerelemente, die ein Submit eines Formulars auslösen können, über die passende Eigenschaft verfügen. Mit *PostBackURL* können die Steuerelemente *Button*, *ImageButton* und *LinkButton* eine neue Seite aufrufen, in der dann die Post behandelt wird.

Cross Page Posting wurde im Kapitel 1 bereits beschrieben.

## UseSubmitBehavior

Die Eigenschaft *UseSubmitBehavior* wird nur vom ASP.NET-Button unterstützt. Je nachdem ob dieses Attribut auf *true* oder *false* gesetzt ist, wird das Formular per *Form Submit* gesendet oder JScript der Submit ausgeführt. Der voreingestellte Wert ist *true*. Damit sieht der Button im Quellcode des Browsers ungefähr so aus:

```
<input type="submit" value="Button" />
```

Wenn *UseSubmitBehavior* auf *false* gesetzt wird, gestaltet sich das Ergebnis wie folgt:

```
<input type="button" value="Button" onclick="javascript: __doPostBack('ct100$ContentPlaceHolder1$Button1','')">
```

## DefaultButton

Eines der kleinen, aber netten Dinge ist das Bestimmen des Buttons, der ausgelöst wird, wenn der Benutzer das Formular mit Eingabe bestätigt. Man spricht dann von der Standardschaltfläche (*Default Button*). Diese kann als Eigenschaft des Formulars gesetzt werden.

```
<form id="form1" runat="server" defaultbutton="button1">
```

Wenn der Button nicht existiert, wird eine Ausnahme ausgelöst.

Auch das *Panel*-Steuerelement besitzt die Eigenschaft *DefaultButton* mit dem dann der Standard Button innerhalb des Panels bestimmt wird.

Im HTML-Code erzeugt ASP.NET ein JScript um den Submit auszulösen.

```
<form method="post" action="DefaultButton.aspx" onkeypress="javascript:return  
WebForm_FireDefaultButton(event, this.elements['Button1'])" id="form1">
```

Ein kleines Problem ergibt sich im Zusammenhang mit Masterseiten. Die endgültige ID des Buttons ist nämlich nicht *Button1*, sondern ein hierarchisch zusammengesetzter Name. Wenn Sie den Default-Button auch hier setzen wollen, führt daran aber kein Weg vorbei. Dabei muss erst das Formular-Steuerelement per *FindControl* identifiziert werden. Da die Rückgabe vom Typ *Object* ist, folgt noch eine Typumwandlung in ein *HtmlForm*-Steuerelement. Darauf kann dann die Eigenschaft *DefaultButton* angesprochen werden.

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)  
    CType(Master.FindControl("Form1"), HtmlForm).DefaultButton = _  
        "ctl00$ContentPlaceHolder1$Button1"  
End Sub
```

**Listing 5.6** Masterseite Default Button setzen

Noch besser ist es, die Eigenschaft *UniqueID* zu verwenden, um zur Laufzeit per Code herauszufinden, welche ID der Button schlussendlich besitzt.

### TIPP

Wenn Sie absolut sicher gehen wollen, wie denn die genaue ID des gewünschten Buttons ist, empfehle ich *Page Tracing*. Dort werden alle Steuerelemente mit ihren erzeugten IDs aufgelistet.

## DefaultFocus

Ganz ähnlich wie *DefaultButton* wird *DefaultFocus* im Form-Element eingesetzt. Es geht dabei allerdings natürlich um Eingabe-Steuerelemente wie z.B. Textboxen.

```
<form id="Form1"  
    defaultbutton="SubmitButton"  
    defaultfocus="TextBox1"  
    runat="server">
```

**Listing 5.7** Eine Textbox erhält den Focus

Das Setzen des Fokus wird mit JScript erreicht. Sie können aber auch nach wie vor mit der JScript-Funktion *SetFocus* den Fokus setzen.

Auch die *Page*-Klasse besitzt eine zweifach überladene Funktion *SetFocus*. Mit dieser kann dann programmatisch in *Page\_Load* der Fokus auf ein spezifisches Steuerelement gesetzt werden. Sollten Sie das tun, überschreiben Sie damit den im *Form*-Element definierten Fokus-Status.

Darüber hinaus hat jedes Webserver Steuerelement die *Fokus*-Eigenschaft, mit der ebenfalls das Steuerelement ausgewählt wird.

```
TextBox2.Focus()
```

#### HINWEIS

Auch passive Steuerelemente wie Label können den Fokus haben.

### SubmitDisabledControls

Wer sich einmal genauer mit dem Versenden von HTML-Formularen beschäftigt hat, weiß, dass nur Werte von ausgefüllten Eingabeelementen per POST-Kommando gesendet werden. Werte, die in deaktivierten Steuerelementen stehen, gehen bei einem Postback verloren.

Damit das nicht passiert, wird das Attribut *submitDisabledControls* im *Form*-Element auf *true* gesetzt.

```
<form id="form1" submitdisabledcontrols=true runat="server">
```

## Eingabeprüfung

Salopp könnte man sagen: »Traue keinem Benutzer über 30. Auch keinem unter 30. Und erst recht nicht seinen Eingaben.« Also benötigen Sie auf allen Formularseiten eine Eingabeprüfung, die am besten am Client und am Server zum Tragen kommt. Genau das leisten die *Validations-Steuerelemente* schon seit ASP.NET 1.0. Es gibt für jede Art der Prüfung ein eigenes Steuerelement.

Steuerelement	Funktion
<i>RequiredFieldValidator</i>	Prüft, ob ein Wert in das Eingabefeld eingetragen ist.
<i>RangeValidator</i>	Prüft, ob der Wert in einem zulässigen Bereich liegt.
<i>RegularExpressionValidator</i>	Überprüft die Eingabe anhand eines regulären Ausdrucks.
<i>CompareValidator</i>	Vergleicht den Wert mit einem anderen Wert.
<i>CustomValidator</i>	Für eigene Prüfroutinen.
<i>ValidationSummary</i>	Zeigt das Ergebnis der Prüfung.

**Tabelle 5.1** Die sechs Validation-Steuerelemente

Ein oder auch mehrere Validations-Steuerelemente werden an eine Eingabemöglichkeit, z.B. eine Textbox, gebunden. Dies geschieht über die Eigenschaft *ControlToValidate* des Validations-Steuerelements. Für die Prüfung wird JScript Code erzeugt, sodass jedes Feld unmittelbar ohne Postback geprüft wird.

Ein häufig verwendetes Steuerelement zur Eingabeprüfung ist der *RegularExpressionValidator*. Dieser prüft anhand von regulären Ausdrücken.

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
runat="server" ControlToValidate="TextBox1"
ErrorMessage="Keine Telefonnummer"
ValidationExpression="(\\(0\\d\\d\\) |\\(0\\d{3}\\) )?\\d )?\\d\\d \\d\\d \\d\\d|\\(0\\d{4}\\) \\d \\d\\d-\\d\\d?">
</asp:RegularExpressionValidator>
```

**Listing 5.8** Telefonnummer wird anhand von regulären Ausdrücken geprüft

Wenn die Anwendung allerdings in den USA eingesetzt werden soll, ergibt sich daraus ein Problem – hier hat die Telefonnummer ein anderes Format. Dank ASP.NET-Expressions kann der reguläre Ausdruck allerdings leicht ausgelagert werden z.B. in die *web.config*.

```
ValidationExpression=<%$ appSettings:TelefonnummerFormat%>
```

Das gesamte Ergebnis der Prüfung wird per *ValidationSummary* Steuerelement ausgegeben. Dabei kann dieses Steuerelement per Attribut *ShowMessageBox* sogar eine MessageBox im Browser öffnen (was natürlich in Wirklichkeit per JScript geschieht).

```
<asp:ValidationSummary ID="ValidationSummary1" Runat="server" ShowMessageBox="True" />
```



**Abbildung 5.2** Fehlerhafte Eingaben werden von den Validator-Steuerelementen geprüft

## ValidationGroup

Die Validation-Steuerelemente lassen bisher wenige Wünsche offen. Einen Wunsch gibt es jedoch: Es wird immer das komplette Formular geprüft. Wenn das nicht erwünscht ist, weil z.B. entweder die Kreditkarten- oder Bankinformationen eingegeben werden müssen, war das bisher nicht möglich. Diese Eingaben können nun zu Gruppen zusammengefasst werden. Dies geschieht mit dem zusätzlichen Attribut *ValidationGroup*. Dabei werden in den Validatoren und in den Buttons gleich lautende Einträge im *ValidationGroup*-Attribut vorgenommen.

```

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
Kreditkartennummer
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server" ErrorMessage="Bitte füllen Textbox1"
SteuerelementToValidate="TextBox1" ValidationGroup="Kreditkarte">*
</asp:RequiredFieldValidator>
<asp:Button ID="Button1" Runat="server" Text="Button" ValidationGroup="Kreditkarte" /> <br />
Kontonummer
<asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server" ErrorMessage="bitte füllen Textbox2"
SteuerelementToValidate="TextBox2" ValidationGroup="Konto">*
</asp:RequiredFieldValidator>
<asp:Button ID="Button2" Runat="server" Text="Button" ValidationGroup="Konto" />
<asp:ValidationSummary ID="ValidationSummary1" Runat="server" ShowMessageBox="True"
ValidationGroup="Kreditkarte" />
<asp:ValidationSummary ID="ValidationSummary2" Runat="server" ValidationGroup="Konto" />
</asp:Content>

```

Listing 5.9 Ein Formular mit zwei Gruppen

**HINWEIS**

Pro ValidationGroup wird auch ein ValidationSummary Steuerelement benötigt.



Abbildung 5.3 Nun wird jede Gruppe individuell validiert

Auf der Serverseite wird dann üblicherweise aus Sicherheitsgründen mit der Funktion *Validate* nochmals geprüft, ob die Prüfung positiv ausfällt. Die Funktion *Validate* ermöglicht jetzt als Parameter den Namen der Validierungsgruppe. Mit *IsValid* kann dann im Code eine Verzweigung eingerichtet werden.

```

Page.Validate("Konto")
If Page.IsValid Then
End If

```

Listing 5.10 Serverseitige Prüfung der Eingaben

## SetFocusOnError

Wenn sich die Eingabe des Benutzers in das Formular nach der Prüfung als fehlerhaft erweist, muss dieser seine Eingaben korrigieren. Dazu sollte der Cursor im ersten Feld stehen, das fehlerhaft befüllt wurde. Dafür setzen Sie in den Validationssteuerelementen *SetFocusOnError* auf *True*. Die Standardeinstellung ist allerdings *false*.

## CausesValidation für Listen

Mit dem Attribut *CausesValidation* wird gesteuert, ob ein Element die Validierung auslöst. Ist dieses Attribut nicht gesetzt, wird automatisch validiert. Um die Validierung auszuschalten, muss *CausesValidation* explizit auf *false* gesetzt werden. Dieses Attribut ist nun auch in Listen-Steuerelementen verfügbar.

## ValidateEmptyText

Diese Eigenschaft ist nur für den *CustomValidator* verfügbar. Wenn sie auf *true* gesetzt wird, ist ein leerer Wert als Eingabe zulässig.

# Datenbindung

Obwohl Datenbindung eine sehr häufige Tätigkeit eines Webentwicklers ist, erforderte diese bisher viel Code. In den Kapiteln 7 und 8 werden dieses Thema und die dazu benötigten Steuerelemente im Detail behandelt.

Einige kleinere Änderungen werden bereits hier beschrieben:

## AppendDataBoundItems

Viele der datengebundenen Steuerelemente erlauben es, feste Daten und Daten aus der Datenbank zu mischen. Dafür gibt es das Attribut *AppendDataBoundItems*, mit dem auch nachträglich Daten hinzugefügt werden können. Ein Listen-Steuerelement kann im Entwurfsmodus bereits mit Daten gefüllt und zur Laufzeit mit weiteren Daten, z.B. aus einer Datenbank, ergänzt werden. Es muss nur das Attribut *AppendDataBoundItems* auf *true* gesetzt werden.

```
<%@ Page Language="VB" MasterPageFile="-/all.master" Title="Untitled Page" %>
<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myData As New Hashtable
        myData.Add(1, "Entwickler")
        myData.Add(2, "Autoren")
        myData.Add(3, "Sprecher")
        ListBox1.DataTextField = "VALUE"
        ListBox1.DataValueField = "KEY"
        ListBox1.DataSource = myData
        ListBox1.DataBind()
    End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <asp:ListBox ID="ListBox1" Runat="server" Width="84px" Height="80px" AppendDataBoundItems="True">
        <asp:ListItem Value="-1">bitte w&#228;hlen...</asp:ListItem>
    </asp:ListBox></asp:Content>
```

**Listing 5.11** Eine Liste erhält zusätzliche Einträge





Abbildung 5.4 Listbox mit Daten aus zwei Quellen

## Listbox Enabled

In einer *ListBox* sind die Einträge in Form von *ListItem*-Elementen aufgeführt. Hier kommt das neue Attribut *Enabled* hinzu.

```
<asp:ListItem Value="-1" Enabled="false">
```

Im Browser wird der Eintrag dann nicht angezeigt, ist aber in der Aufzählung vorhanden.

# Änderungen an bestehenden Steuerelementen

Es sind nicht nur jede Menge Steuerelemente hinzugekommen, auch die bestehenden Steuerelemente wurden erweitert. Einige Änderungen sind hier bereits beschrieben worden.

Im Folgenden werden für jedes Steuerelement die neuen Funktionalitäten vorgestellt. Weil das Thema mehr als umfangreich ist, wird auf Wiederholungen von Attributen und deren Beschreibung verzichtet.

## Button

Das einfache Button-Steuerelement war schon bisher ein sehr nützlicher Helfer. Damit wird ein Formular an den Webserver gesendet und somit ein Roundtrip erzeugt. Am Server kann dann die *Click*-Ereignismethode abgearbeitet und die neue Seite an den Browser gesendet werden.

## OnClientClick

Da die Ereignisbehandlung am Server durchgeführt wird, kann am Client nichts mehr gegen das Senden der Seite unternommen werden. So wünschen sich viele Entwickler, dass der Benutzer erst einen Dialog *Confirm* bestätigen muss, bevor beispielsweise ein Datensatz gelöscht wird. Der Dialog kann per JScript und der *Confirm*-Funktion erzeugt werden. Die Zuordnung zum ASP.NET-Button wird jetzt mit dem Attribut *OnClientClick* erzeugt. Diesem kann ein kurzes JScript oder der Aufruf einer JScript-Funktion mitgegeben werden. Wenn der Rückgabewert aus der JScript Funktion *false* ist, wird kein Submit durchgeführt und ein anstehender Löschvorgang abgebrochen.

```
<asp:Button ID="Button1" Runat="server" Text="Delete" OnClick="Button1_Click"
OnClientClick="Jscript:return confirm('wirklich');"/>
```

Listing 5.12 Sicherheitsabfrage vor dem Löschen von Daten

Diese Eigenschaft steht in den Steuerelementen *Button*, *ImageButton* und *LinkButton* zur Verfügung.

## PostBackUrl

Seit ASP.NET 2.0 kann der Inhalt eines Formulars, das per POST abgesendet wurde, auch auf einer anderen Seite behandelt werden. Um die Zielseite anzugeben, benötigt man das neue Attribut *PostBackUrl*.

Diese Eigenschaft steht in den Steuerelementen *Button*, *ImageButton* und *LinkButton* zur Verfügung.

Die Funktionalität des Cross Page Posting wird in Kapitel 12 genauer erläutert.

## UseSubmitBehavior

Ein Formular-Submit wird in der Standardeinstellung durch einen Klick auf einen Button ausgeführt. Mit dem neuen Attribut *UseSubmitBehavior* kann dieses Verhalten geändert werden. Wenn Sie den Wert auf *false* setzen, wird statt dem klassischen Postback ein JScript ausgeführt. Wenn ein Buttonklick dann per JScript simuliert werden soll, hilft die Funktion *GetPostBackEventReference*.

```
meinImage.Attributes.Add("onclick", Page.GetPostBackEventReference(Button1))
```

## LinkButton

Das *LinkButton*-Steuerelement enthält die neuen Attribute *OnClick* und *PostBackUrl*. Die Details sind im *Button*-Steuerelement beschrieben.

## ImageButton

Das *ImageButton*-Steuerelement enthält die neuen Attribute *OnClick* und *PostBackUrl*. Die Details sind im *Button*-Steuerelement beschrieben.

## CheckBox

Das *CheckBox*-Steuerelement besteht eigentlich aus zwei Steuerelementen: Eben aus der *CheckBox* und aus einem *Label*, um den beschreibenden Text anzuzeigen. Um die beiden Elemente mit zusätzlichen Attributen zu versehen, gibt es zwei neue Attributauflistungen im *CheckBox*-Steuerelement. Mit *InputAttributes* werden die Attribute des im HTML-Code generierten INPUT-Elements verwaltet und mit *LabelAttributes* die Collection des zugehörigen Labels. Beide sind nur zur Laufzeit und nicht zur Entwurfszeit verfügbar.

```
<script runat="server">
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    checkbox1.LabelAttributes.Add("myLabelKey", "LabelValue")
    CheckBox1.InputAttributes.Add("myInputKey", "InputValue")
End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <asp:CheckBox ID="CheckBox1" Runat="server" Text="Beschreibung" />
</asp:Content>
```

**Listing 5.13** Das Checkbox-Steuerelement verfügt über zusätzliche Attribute

In der praktischen Anwendung kann man so beispielsweise einen Hotkey (*Schnellzugriffstaste*) definieren und einen ToolTip erzeugen.

```
CheckBox1.LabelAttributes.Add("accesskey", "M")  
CheckBox1.LabelAttributes.Add("title", "ALT+M um Checkbox zu setzen")
```

**Listing 5.14** Erweiterte Checkbox-Attribute

In der HTML-Ausgabe wird dafür ein HTML-Label-Element erzeugt.

## Image-Steuerelement

Das Image-Steuerelement erzeugt ein *img*-HTML-Element. Zusätzliche Attribute beeinflussen die Darstellung des Bildes im Browsers. So wird über *style=>border-width:0px*; automatisch der Rahmen entfernt, auch wenn ein Hyperlink über dem Bild liegt. Zwei neue Attribute berücksichtigen vor allem behinderte Menschen. Mit *DescriptionURL* wird ein erweiterter Beschreibungstext eingebunden. *GenerateEmptyAlternatetext* erzeugt über das *alt*-Attribut eine leere Beschreibung.

Näheres zu diesem Thema findet sich bereits früher in diesem Kapitel.

## Label-Steuerelement

Mit dem Label-Steuerelement werden Ausgaben im Browserfenster platziert. Das Label dient damit als Platzhalter für den späteren Text. Zu ASP-Zeiten wurden solche Ausgaben oft mit *Response.Write* gemacht. Da Labels häufig eine Beschreibung für Eingabemöglichkeiten sind, lassen sich jetzt Label und Eingabesteuerelemente wie z.B. eine *TextBox* mit dem neuen Attribut *AssociatedControlID* verbinden. Mit dem Attribut *AccessKey* wird dann ein Hotkey definiert. Das Drücken des Hotkeys setzt dann den Cursor in das assoziierte Eingabe-Steuerelement.

```
<asp:Label ID="Label1" AccessKey="E" Runat="server" Text="<u>E</u>ingabe" _  
AssociatedControlID="TextBox2"></asp:Label>  
<asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox>
```

**Listing 5.15** Ein Label-Steuerelement wird einer Textbox zugeordnet

## TextBox-Steuerelement

Um Texte in ein Formular einzugeben, ist ein *TextBox*-Steuerelement geeignet. In der Eigenschaft *Text* befindet sich der eingegebene Text. Um die Eingaben komfortabel zu gestalten, kann auf die Profilinformaton des Browser zugegriffen werden. Dazu wird das Attribut *AutoCompleteType* verwendet. Über dieses wird das Feld aus dem Profil zugewiesen. Mit dem Attribut *CausesValidation* kann die Seitenvalidierung umgangen werden, wenn das Formular mit diesem Button übertragen wird.

## AdRotator

Das *AdRotator*-Steuerelement gehört zur Reihe der so genannten *Rich-Steuerelemente*. Das liegt an der umfangreichen Funktionalität dieses Steuerelements. Mit dem *AdRotator* werden Werbebanner (*Ads*) periodisch ausgetauscht. Ziel ist es, zufällig wechselnde Banner in der Webseite einzublenden. Das Steuerelement kann nun auch an eine Datenquelle gebunden werden. Mit ASP.NET 1.x war dies nur per XML Datei möglich.

## \$ Crashkurs

Die wichtige Frage lautet: »Wie baue ich in zwei Minuten eine Banner-Werbung in meine Webseite ein und verdiene Millionen damit?« Zumindest den ersten Teil können Sie hier kennen lernen:

Ziehen Sie aus der Werkzeugleiste der Entwicklungsumgebung das *AdRotator*-Steuerelement in die Webseite. Dann bestimmen Sie die Steuerdatei mit dem Attribut *AdvertisementFile*:

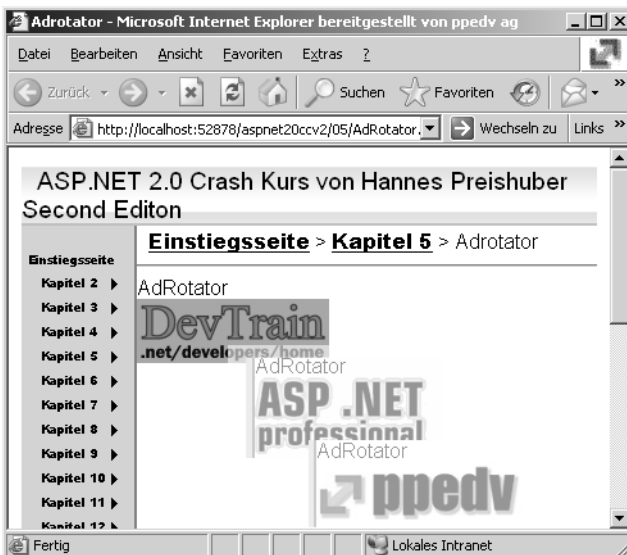
```
<asp:AdRotator ID="AdRotator1" Runat="server" AdvertisementFile="~/05/myad.xml" />
```

**Listing 5.16** Werbung mit einer Zeile ASPX-Code

Die Steuerdatei muss im XML-Format vorliegen. Pro Banner gibt es ein Element `<Ad>` unterhalb des Root-Elements `<Advertisements>`. Die Wahrscheinlichkeit der Einblendung ergibt sich aus dem *Impressions*-Wert. Je höher der Wert, desto häufiger wird das Banner eingeblendet.

```
<Advertisements>
  <Ad>
    <ImageUrl>banner/ppedv.jpg</ImageUrl>
    <NavigateUrl>http://www.ppedv.de</NavigateUrl>
    <AlternateText>ppedv ag</AlternateText>
    <Impressions>80</Impressions>
    <Keyword>Topic1</Keyword>
  </Ad>
  ...
</Advertisements>
```

Und das war's! Damit bleibt nur noch, die Werbekampagne im Browser zu betrachten.



**Abbildung 5.5** Drei abwechselnd angezeigte Werbungen

## Details

Auch das *AdRotator*-Steuerelement strotzt nur so vor verschiedenen Eigenschaften und Methoden. Nur einige der wichtigsten und interessantesten werden hier kurz erläutert:

AdType	Bedeutung
<i>Banner</i>	Ein Banner wird im Browserfenster angezeigt.
<i>PopUnder</i>	Das Banner wird in einem neuen Browserfenster angezeigt, das in den Hintergrund rückt. Auf der eigentlichen Seite wird kein Banner angezeigt.
<i>PopUp</i>	Das Banner wird in einem neuen Browserfenster angezeigt, das im Vordergrund steht.

**Tabelle 5.2** Drei Banner-Typen

Wenn mit dem *PopUp*-Typen gearbeitet wird, sind noch einige zusätzliche Attribute sinnvoll, um das Verhalten des Browserfensters zu steuern.

Attribut	Bedeutung
<i>PopFrequency</i>	Die Frequenz in Prozent, in der das PopUp-Browserfenster angezeigt wird. 50 bewirkt, dass statistisch bei jedem zweiten Aufruf das PopUp angezeigt wird.
<i>PopPositionLeft</i>	Die Startposition des neu geöffneten Browserfensters von links, angegeben in Pixel.
<i>PopPositionRight</i>	Die Y-Startkoordinate des neu geöffneten Browserfensters von oben, in Pixel.
<i>Target</i>	Das Ziel des PopUps. Dies kann der Name des Browserfensters, des Frames oder <i>_blank</i> sein.

**Tabelle 5.3** Die PopUp-Typen

## Datenbindung

Zu einem richtigen Rich-Steuerelement gehört auch die Möglichkeit, Daten zu binden – genauer gesagt: An die neuen DataSource-Steuerelemente zu binden. Die Daten kommen dann statt aus einer XML-Datei aus einer Datenbanktabelle.

Datenbindung wird im Kapitel 7 genauer beschrieben.

## Calendar-Steuerelement

Dieses Steuerelement dient dazu, einen Kalender anzuzeigen und dem Benutzer so die Navigation in Datumswerten zu erleichtern. Ein solches Steuerelement war auch bereits in ASP.NET 1.x vorhanden und mit ähnlicher Funktionalität ausgestattet.

Da im Browser der Kalender per HTML-Tabellen dargestellt wird, sind einige Attribute hinzugekommen, die auch bereits bei den Tabellen beschrieben worden sind. Dies sind *Caption* und *UseAccessibilityHeader*. Außerdem gibt es ein neues *SkinID*-Attribut, das in Zusammenarbeit mit der Verwendung von Designs zum Tragen kommt.

Eine weitere neue Eigenschaft findet sich in den *Expressions*. Damit kann der Kalender mehrsprachig ausgeführt werden.

## Literal-Steuerelement

Das Literal-Steuerelement dient als eine Art Platzhalter in der Webseite. Zur Laufzeit wird nur der reine Inhalt des Steuerelements gerendert, im Unterschied zu Steuerelementen wie dem *Label*-Steuerelement, das ein *<DIV>*-Element erzeugt. Die wesentliche Neuerung ist das zusätzliche Attribut *Mode*:

Mode	Bedeutung
<i>Transform</i>	Abhängig vom erkannten Zielsystem und Browser werden nicht unterstützte HTML-Sequenzen nicht an den Browser gesendet.
<i>PassThrough</i>	Alles, was in der Eigenschaft <i>Text</i> steht, wird einschließlich HTML-Steuerelementen direkt an den Browser gesendet.
<i>Encode</i>	Der Inhalt des Textes wird erst HTML-kodiert, bevor er an den Client-Browser gesendet wird.

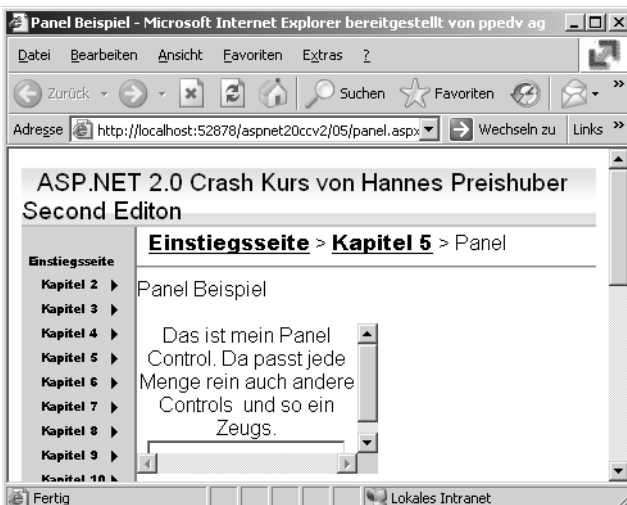
**Tabelle 5.4** Das neue Mode-Attribut und seine drei möglichen Werte

## Panel-Steuerelement

Das Panel-Steuerelement ist auch in der Windows-Entwicklung bekannt. Es gruppiert durch einen Rahmen optisch andere Steuerelemente. Oft steht in dem Rahmen auch noch eine Überschrift. Genau diese Art der Gruppierung erreichen Sie in ASP.NET mit dem Panel-Steuerelement.

### Scrollen

Neu hinzugekommen ist die Möglichkeit, das Panel scrollbar zu machen. In der Standardeinstellung wird das Panel per *<DIV>*-Element erzeugt. Wenn der Inhalt mehr Platz braucht, als das Panel groß ist, kann man über das Attribut *ScrollBars* die Laufleisten erzeugen. Die möglichen Werte sind: *None*, *Horizontal*, *Vertikal*, *Both* und *Auto*.



**Abbildung 5.6** Scrollbars schaffen Platz

## Direction

Wenn Sie den Text nicht von links nach rechts ausgerichtet haben wollen, können Sie mit dem Attribut *Direction* dieses Verhalten ändern. Mögliche Werte sind *NotSet*, *LeftToRight* und *RightToLeft*. Der hauptsächliche Einsatzzweck wird für Websites in bestimmten Sprachen sein, die – wie beispielsweise Hebräisch – von rechts nach links geschrieben werden. Im Browser wird der Eintrag *dir=>ltr<* erzeugt.

## Fieldset

Wenn die Gruppierung von Inhalten per Panel-Steuerelement im Rahmen auch noch eine Überschrift haben soll, kommt das Attribut *GroupingText* zum Einsatz. Allerdings wird dann komplett anderer HTML-Code für den Browser erzeugt. Der Bereich wird durch ein *HTML-<fieldset>*-Element eingegrenzt. Das Unterelement *<legend>* enthält dann die Überschrift.

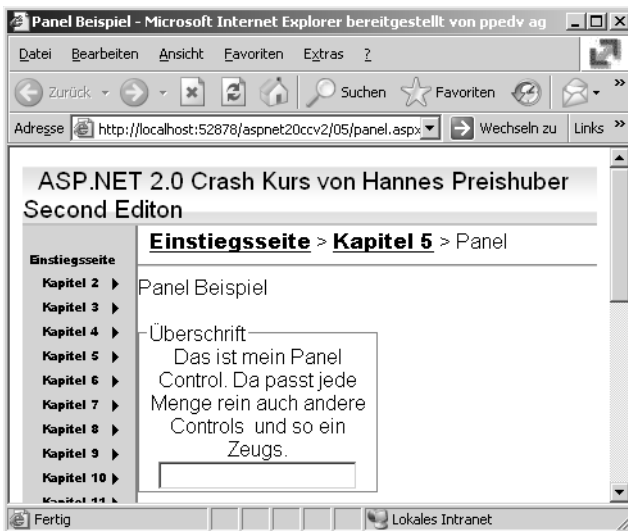


Abbildung 5.7 Ein Bereich wird beschriftet

Ein so erzeugtes Panel verfügt über keine Scrollbars mehr. Falls Sie Scrollbars benötigen, müssen Sie das Attribut *GroupingText* entfernen.

## Table-Steuerelement

Das *Table*-Steuerelement erzeugt eine Tabelle abhängig vom Zielsystem. Eine Tabelle besteht aus Reihen, den *TableRow*-Unterelementen. Diese wiederum beinhalten Zellen, die *TableCell*-Elemente.

```
<asp:Table ID="Table1" Runat="server" CaptionAlign="Bottom" GridLines="Both">
  <asp:TableRow Runat="server">
    <asp:TableCell Runat="server">
      Eins</asp:TableCell>
    <asp:TableCell Runat="server">
      Zwei</asp:TableCell>
  </asp:TableRow>
</asp:Table>
```

Listing 5.17 Eine Tabelle mit einer Zeile und zwei Spalten

In der Praxis kommt das Tabellen-Steuerelement hauptsächlich im Zusammenhang mit zur Laufzeit generierten Tabellen zum Einsatz. Dazu werden an eine Tabelle *TableRow*-Objekte angehängt, die wiederum mit *TableCell*-Elementen gefüllt sind.

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim nr As New TableRow()
    Dim i As Integer
    For i = 0 To 1
        Dim nc As New TableCell()
        nc.Steuerelements.Add(New LiteralControl("row neu"))
        nr.Cells.Add(nc)
    Next i
    Table1.Rows.Add(nr)
End Sub
```

**Listing 5.18** Manuell zur Tabelle hinzugefügte Zeile

### ACHTUNG

Nach einem Postback gehen die zur Laufzeit hinzugefügten Zeilen verloren.

Tabellen mit Daten als Inhalt sollten korrekterweise mit einem Table-Header statt einer normalen Zeile versehen werden. Dann wird für den Browser `<TH>` statt `<TD>` gerendert. Dies erleichtert Lesegeräten das Erkennen der Inhalte. Aber auch für die Formatierung z.B. per CSS kann das nützlich sein.



**Abbildung 5.8** Die Überschrift wird anders dargestellt

## Tabelle Caption

Tabellen können auch über Überschriften verfügen. Das kann man natürlich elegant mit einem *Label*-Steuerelement lösen. Die *Caption*-Eigenschaft hilft aber Lesegeräten, die Daten korrekt zu erkennen. Der Inhalt wird als *HTML*-`<CAPTION>`-Element für den Browser gerendert. Eigentlich ist die Bezeichnung *Überschrift* unrichtig, da diese über das Attribut *CaptionAlign* an jeder Seite der Tabelle dargestellt werden kann. Mögliche Werte sind: *Bottom*, *Top*, *Right* und *Left*.

```
<asp:Table ID="Table1" Runat="server" CaptionAlign="Bottom" Caption="Ihre Daten">
```



## XML-Steuerelement

Das *Xml*-Steuerelement wird verwendet, um XML-Daten anzuzeigen. Gleichzeitig kann eine XSL-Transformation durchgeführt und eine Auswahl per XPath-Attribut vorgenommen werden. Mit XSLT können XML-Daten in anders strukturierte XML-Daten umgewandelt werden. Diese Möglichkeit wird auch verwendet, um aus XML-Daten HTML zu erzeugen. Danach können diese Daten im Browser angezeigt werden.

```
<asp:Xml ID="Xml1" Runat="server" DocumentSource="~/05/people.xml"
  TransformSource="~/05/peopletable.xsl"></asp:Xml>
```

Beim *Xml*-Steuerelement ist die Eigenschaft *XPathNavigator* neu hinzugekommen. *XPath* ist eine pfadorientierte Abfragesprache zum Selektieren von XML-Datenmengen. *XPathNavigator* wird ein Objekt vom Typ *XPathNavigator* zugewiesen, mit dem XPath-Abfragen durchgeführt werden.

Über das Ereignis *Databinding* kann zur Laufzeit in den Konvertierungsprozess eingegriffen werden.

## Änderungen am HTML-Server-Steuerelement

Die HTML-Server-Steuerelemente werden eher selten verwendet, da die Webserver-Steuerelemente viel mächtiger und einfacher zu verwenden sind. In manchen Fällen muss man allerdings HTML-Steuerelemente verwenden, gerade dann, wenn man 100prozentige Kontrolle über den generierten HTML-Code haben möchte. Hier werden nun einige Attribute erklärt, die neu hinzugekommen sind und bei verschiedenen HTML-Steuerelementen funktionieren.

### ValidationGroup

Eingaben können mit Validations-Steuerelementen gruppiert werden. So werden beim Drücken eines bestimmten Buttons nur alle Eingaben in den Steuerelementen geprüft, die zu einer Gruppe gehören. Für *HtmlButton*, *HtmlInputButton* und *HtmlInputImage* steht dieses Attribut zur Verfügung. Andere HTML-Steuerelemente wie *Input* (Text) besitzen das Attribut *ValidationGroup* nicht.

### DefaultFocus

Wenn ein Formular angezeigt wird, sollte ein bestimmtes Eingabefeld den Fokus aufweisen. Speziell wenn ein Fehler aufgetreten ist, hilft es dem Benutzer, den Cursor gleich in das betroffene Feld zu setzen. Das Attribut *DefaultFocus* des HTML-Formulars zeigt auf das entsprechende Eingabe-Steuerelement.

```
<form id="form1" runat="server" defaultfocus="Textbox1">
```

Im Browser wird dafür ein JScript generiert, das den Fokus nach dem Laden der Seite setzt.

### SubmitDisabledControl

Wenn das Attribut *SubmitDisabledControl* im Form-Element auf *true* gesetzt ist, werden auch Steuerelemente mit dem Attribut *disabled* übertragen. Dies sind Steuerelemente, deren Attribut

`enabled=false` ist. Im Normalfall überträgt der Browser nur für auf diese Art aktivierte Steuerelemente die enthaltenen Werte. Damit verlieren diese aber nach dem Postback ihren Inhalt. Um dies zu ändern, wird das Attribut gesetzt.

```
<form id="form1" runat="server" submitdisabledControls="true">
```

## HtmlSelect

Das `<SELECT>`-Element wird durch den Zusatz `runat=server` zum HTML-Server-Steuerelement.

Schon seit ASP.NET 1.X kann dieses Steuerelement an Daten gebunden werden. Das neue Attribut `datasourceid` ermöglicht das Binden an eines der neuen DataSource-Steuerelemente.

```
<select name="Select1" runat="server" datasourceid="dsid">
  <option value="1">Wien</option>
  ..
</select>
```

**Listing 5.19** Ein HTML-Select wird an eine Datenquelle gebunden

### HINWEIS

Die Gruppierung innerhalb einer Select-Liste mit dem `<OPTGROUP>`-Unterelement ist als HTML-Server-Steuerelement nicht möglich. Entfernen Sie dann die `<OPTGROUP>`-Elemente.

## Neue Webserver-Steuerelemente

Visual Basic hat sicher auch deshalb so großen Erfolg gehabt, weil es eine unglaubliche Menge an Steuerelementen von Drittherstellern gibt. Mit Steuerelementen lassen sich immer wiederkehrende Visualisierungsaufgaben in Minuten erledigen. Ein solcher Steuerelement-Markt wie bei Visual Basic hat sich bei ASP.NET bislang noch nicht so richtig entwickelt, und wahrscheinlich ist auch das der Grund dafür, dass Microsoft selbst eine so große Anzahl neuer Steuerelemente bereithält. Diese decken neue Funktionen wie z.B. Web Parts ab oder erweitern vorhandene Funktionen.

## FileUpload

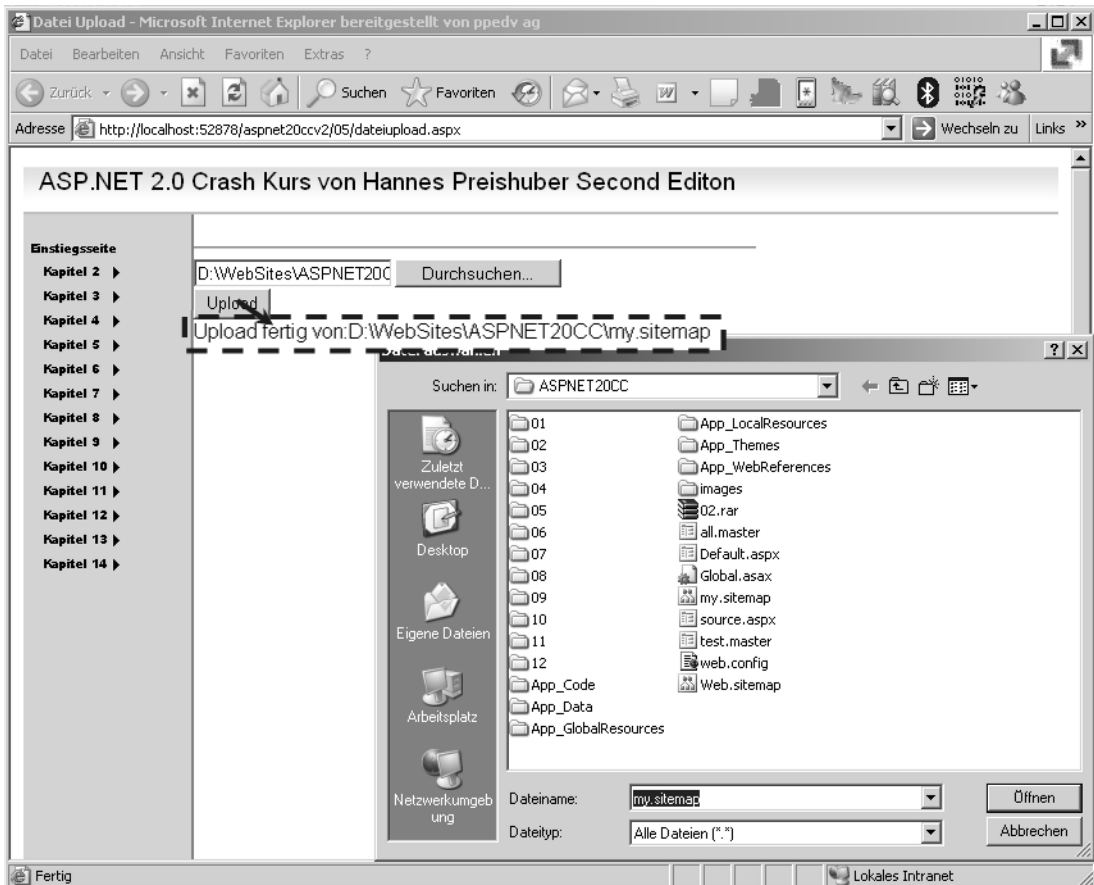
Das Hochladen von Daten war bisher auch schon mit dem HTML-Steuerelement möglich. Es gehört zu den gängigen Aufgaben, Dateien vom Browser zum Server zu bringen. In den Anfängen wurden dazu oft ActiveX-Steuerelemente verwendet. Dabei hat HTTP durchaus eine Upload-Möglichkeit eingebaut. Dazu muss das Formular das spezielle Attribut `enctype=»multipart/form-data«` aufweisen. Außerdem benötigt man zusätzlich noch ein HTML-Input-Element vom Typ `File`.

Genau das erzeugt das ASP.NET Web Steuerelement `FileUpload`. Allerdings braucht man für den eigentlichen Upload noch ein weiteres Steuerelement, das den Upload-Vorgang per Code anstößt.

```
<asp:FileUpload ID="FileUpload1" Runat="server" AlternateText="Ihr Browser unterstützt kein Upload" />
<br />
<asp:Button ID="Button1" Runat="server" Text="Upload" OnClick="Button1_Click" /><br />
<asp:Label ID="Label1" Runat="server" Text="Label"></asp:Label>
```

**Listing 5.20** *FileUpload* und *Button* Webserver-Steuerelement

Der Benutzer drückt den Button *Durchsuchen*. Dadurch erscheint ein Dateiauswahl-Dialog. Nach Auswahl einer Datei drückt er den Upload-Button. Nach erfolgreichem Upload erhält der Benutzer eine Bestätigungsanzeige im Browser.



**Abbildung 5.9** Datei auswählen, auf Upload klicken, fertig!

Leider muss der eigentliche Upload manuell programmiert werden. Da dabei viel schief gehen kann, sollten Sie eine umfangreiche Fehlerbehandlung einbauen. Beim Upload hilft das Objektmodell des *FileUpload*-Steuerelements. So prüft die Eigenschaft *HasFile* mit einer booleschen Rückgabe, ob überhaupt eine Datei ausgewählt wurde. Die Funktion *SaveAs* speichert dann die Datei unter dem angegebenen Namen an dem ausgewählten Ort – in diesem Fall in dem Unterverzeichnis *Upload*.

**ACHTUNG** Dateioperationen erfordern eine gewisse Sorgfalt, um auch bei Installationen auf anderen Systemen das Programm lauffähig zu halten. *Server.MapPath* gibt dazu den realen Pfad aus einer virtuellen Angabe aus. Da der Separator für Verzeichnisangaben (Stichwort Mono unter Linux) unterschiedlich sein kann, sollte die Hilfsfunktion *DirectorySeparatorChar* aus der *Path*-Klasse verwendet werden. Dafür muss der Namespace *System.IO* eingebunden werden.

Wenn der Upload abgeschlossen ist, können über das *PostedFile*-Objekt verschiedene Informationen ausgelesen werden.

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    If FileUpload1.HasFile Then
        FileUpload1.SaveAs(Server.MapPath("upload") & Path.DirectorySeparatorChar & FileUpload1.FileName)
        Label1.Text = "Upload fertig von:" & FileUpload1.PostedFile.FileName
    End If
End Sub
```

**Listing 5.21** Die eigentliche Datei-Upload-Routine

**ACHTUNG** Der Datei-Upload birgt viele Fehlerquellen. Meist scheitert es schon an den Zugriffs-Rechten. Unter Windows 2000/XP läuft der so genannte Worker Process (der Arbeitsprozess von ASP.NET) unter dem lokalen ASP.NET-Benutzerkonto. Dieses benötigt Schreibrechte auf das Verzeichnis. Achten Sie darauf, dass dadurch keine Sicherheitslöcher entstehen!

Außerdem ist die Upload-Größe per Standardeinstellung in der *machine.config* auf 4 MB begrenzt. Sie können diese Grenze dort oder auch in der *web.config* mit dem Eintrag *maxRequestLength* beliebig verändern.

## HiddenField-Steuerelement

Gerade im Zusammenspiel mit clientseitigem JScript werden oft HiddenFields verwendet, um Informationen unsichtbar vom Browser zum Server und eventuell auch wieder zurück zu transportieren. Viele ASP.NET-Funktionen machen davon Gebrauch.

Dafür gibt es jetzt das wenig spektakuläre Webserver *HiddenField*-Steuerelement. Eigentlich ist nur die Eigenschaft *Value* wichtig, um den Wert zu lesen oder zu schreiben. Aber es gibt auch noch das sehr nützliche Event *ValueChanged*. Dieses wird nach dem Postback gefeuert, wenn der Wert z.B. von einer JScript-Funktion im Browser verändert wurde.

```
<asp:HiddenField ID="HiddenField1" Runat="server" OnValueChanged="HiddenField1_ValueChanged"
    Value="10" /></asp:Content>
```

**Listing 5.22** HiddenField mit Ereignisprozedur

## ImageMap-Steuerelement

Wenn Grafiken in Bereiche unterteilt werden sollen, geht das mit dem *HTML-<map>*-Element bzw. der Bereichsunterteilung mit *<area>*-Elementen. Dabei kann der Mausklick des Benutzers auf eine Grafik je nach gewähltem Bereich unterschiedlich behandelt werden. Eine übliche Anwendung dafür ist eine geografische Auswahl auf Landkarten.

Mit dem *ImageMap*-Steuerelement bietet ASP.NET 2.0 nun diese Option samt dazugehörigem Objektmodell. Damit wird ein Bild auf einer Webseite angezeigt. Mit den so genannten *HotSpots* wird die Grafik dann

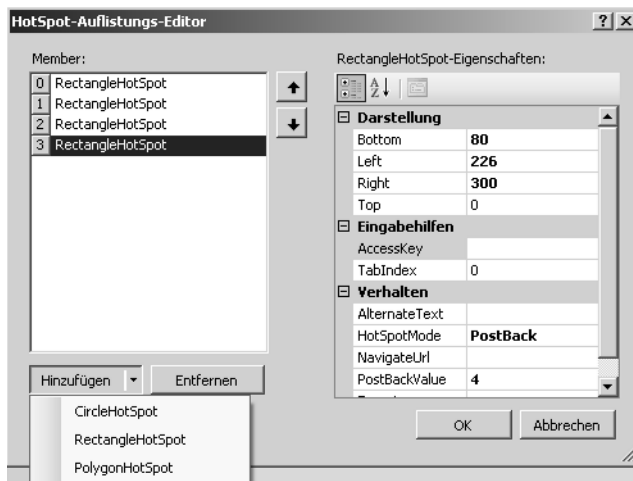
in Bereiche unterteilt. In den Eigenschaften des *ImageMap*-Steuerelements gibt es in den *HotSpots*-Eigenschaften die Möglichkeit, solche Bereiche und die dazugehörige Aktion anzulegen. Dabei hilft einer von drei möglichen Assistenten.

### HotSpot-Typen

Je nach Typ der benötigten Fläche kommen die HotSpot-Typen zum Einsatz.

HotSpot Typ	Beschreibung
<i>CircleHotSpot</i>	HotSpots werden kreisförmig angelegt. Dabei wird der zentrale Punkt in X- und Y-Koordinaten sowie der Radius angegeben.
<i>RectangleHotSpot</i>	Rechteckiger HotSpot. Es werden die Bereichskordinaten mit <i>Top</i> und <i>Bottom</i> sowie <i>Left</i> und <i>Right</i> begrenzt.
<i>PolygonHotSpot</i>	Ein Bereich, der mit vielen X- und Y-Koordinaten definiert wird. Ideal für Landkartenauswahl.

**Tabelle 5.5** Die drei Arten von HotSpots



**Abbildung 5.10** HotSpots definieren

### HotSpotMode

Jedes HotSpot-Element aus dem *ImageMap*-Steuerelement kann ein anderes Verhalten für den Event *Click* an den Tag legen. Im *HotSpotMode*-Attribut wird eine der folgenden vier Optionen festgelegt.

HotSpotMode	Bedeutung
<i>NotSet</i>	Nicht gesetzt.
<i>Navigate</i>	Wenn der Hotspot angeklickt wird, wird eine Navigation auf eine andere Seite durchgeführt. Dafür muss die Zielseite in der Eigenschaft <i>NavigateURL</i> angegeben werden. Im HTML-Code wird der Seitenaufruf direkt ohne JScript durchgeführt, wie dies auch bei einem Hyperlink geschieht. Mit dem <i>Target</i> -Attribut kann damit auch ein neues Fenster geöffnet werden.
<i>PostBack</i>	Ein Postback wird ausgeführt, sodass die Daten am Server verarbeitet werden können. Dafür muss die Eigenschaft <i>PostBackValue</i> gesetzt werden.

**Tabelle 5.6** Die vier möglichen HotSpot-Modi

HotSpotMode	Bedeutung
Inactive	Der Bereich ist inaktiv.

**Tabelle 5.6** Die vier möglichen HotSpot-Modi (Fortsetzung)

## HotSpot-Beispiel

Nachdem die nötigen Grundlagen erörtert worden sind, kann nun zum ersten Beispiel übergegangen werden. Dabei wird eine Grafik in vier rechteckige Bereiche aufgeteilt, die durch den Benutzer per Mausclick ausgewählt werden können.

```
<asp:ImageMap ID="ImageMap1" Runat="server"
  ImageUrl="-/images/imagemap.gif" OnClick="ImageMap1_Click">
  <asp:RectangleHotSpot hotspotmode="PostBack" Bottom="80" Right="75"
    PostBackValue="1" ></asp:RectangleHotSpot>
  <asp:RectangleHotSpot hotspotmode="PostBack" left=76 Bottom="80" Right="150"
    PostBackValue="2"></asp:RectangleHotSpot>
  <asp:RectangleHotSpot hotspotmode="PostBack" left=151 Bottom="80" Right="225"
    PostBackValue="3"></asp:RectangleHotSpot>
  <asp:RectangleHotSpot hotspotmode="PostBack" left=226 Bottom="80" Right="300"
    PostBackValue="4"></asp:RectangleHotSpot>
  <asp:RectangleHotSpot NavigateUrl="-/Default.aspx"></asp:RectangleHotSpot>
</asp:ImageMap>
```

**Listing 5.23** Eins, zwei, drei oder auch vier?



**Abbildung 5.11** Mit einer ImageMap trifft der Benutzer die Auswahl

Um im Programm-Code auf Basis der Benutzerwahl eine Entscheidung treffen zu können, muss das *Click*-Ereignis des *ImageMap*-Steuerelements behandelt werden. Die dabei aufgerufene Prozedur erhält das Objekt *ImageMapEventArgs* als Parameter. Daraus kann die Eigenschaft *PostBackValue* ausgelesen werden. In der ASPX-Seite wurde im HotSpot-Element dieser Wert vorher gesetzt.

```
Sub ImageMap1_Click(ByVal sender As Object, ByVal e As System.Web.UI.WebControls.ImageMapEventArgs)
  Label1.Text = e.PostBackValue
End Sub
```

**Listing 5.24** Einem Label wird die Auswahl zugewiesen

## MultiView-Steuerelement

Wenn ein Formular sehr viele Eingaben erfordert, oder große und komplexe Datenmengen visualisiert werden sollen, bleibt oft nur das Scrollen im Browser übrig. Aus ergonomischen oder auch anderen Gründen ist dies oft unerwünscht. Also teilt man das Formular auf mehrere Webseiten auf. Dabei müssen aber sehr aufwändig die Eingaben des Benutzers zwischengespeichert werden. Das *MultiView* Steuerelement löst diese Probleme.

Das Formular wird zerlegt und in mehreren Sichten (Views) dargestellt. Dazu dienen die *View*-Elemente als ein Unterelement eines *MultiView*-Steuerelements. Zur Laufzeit wird immer nur der Inhalt eines *View*-Elements angezeigt. Dabei wird mit dem Attribut bzw. der Eigenschaft *ActiveViewIndex* das anzuzeigende *View*-Element bestimmt. Der Index ist wie üblich nullbasiert. Wenn kein *ActiveIndex* im *MultiView*-Steuerelement angegeben wird, ist dieser -1 und damit keine *View* sichtbar. Natürlich kann auch per Code, z.B. in der Ereignisprozedur eines »weiter« Buttons, auf den angezeigten Index Einfluss genommen werden.

Im Browser ist immer nur die aktivierte *View* zu sehen und vorhanden. Die anderen Felder werden auch nicht unsichtbar übertragen. Deshalb ist für einen Wechsel des Views unbedingt ein Postback erforderlich.



**Abbildung 5.12** Das MultiView-Steuerelement in der Entwicklungsumgebung

Bisher mag dies alles ein wenig an den Einsatz von Panels erinnern, allerdings unterscheidet sich die Navigation deutlich. Diese ist auch ohne Code rein per Deklaration im ASPX-Code möglich.

Es wird dafür einfach ein übliches Webserver-Steuerelement wie beispielsweise der *Button* verwendet. Einzig das Attribut *CommandName* und eventuell ein zusätzliches Attribut müssen stimmen. Die Buttons müssen in den jeweiligen Views platziert werden.

### SwitchViewByID-Button

Die erste Möglichkeit erlaubt es, einzelne Views direkt anzuzeigen. Dazu wird aus der Toolbox ein Button in die View gezogen und mit dem *CommandName SwitchViewByID* versehen. Die ID der View dient zur Steuerung und wird dem Attribut *CommandArgument* übergeben. Im Beispiel 5.9 wird in der zweiten View per

Button auf die erste View gewechselt. Weil *ActiveViewIndex* 1 ist, wird beim ersten Aufruf der Webseite die zweite View (*ID=VIEW2*) angezeigt.

```
<asp:MultiView ID="MultiView1" Runat="server" ActiveViewIndex="1">
<asp:View ID="View1" Runat="server" >
  <b>Ansicht 1</b>
</asp:View>
<asp:View ID="View2" Runat="server">
  <b>Ansicht 2</b><br />
    <asp:Button id="Button1"
      Text = "erster Teil"
      commandname="SwitchViewByID"
      commandargument="View1"
      runat= "Server"/>
</asp:View>
</asp:MultiView>
```

**Listing 5.25** Ein MultiView-Beispiel

### SwitchViewByIndex-Button

Relatives Wechseln ist die zweite Möglichkeit mit Views zu navigieren. Immer ausgehend von der aktuellen View wird eine View davor oder danach angezeigt. Der *CommandName* der Buttons muss dann *NextView* oder *PrevView* lauten. Sie benötigen dann in der Regel zwei Buttons pro View.

```
<asp:Button id="Button2"
  Text = "Vorher"
  CommandName="PrevView"
  runat= "Server"/>
<asp:Button id="Button3"
  Text = "Nächstes"
  CommandName="NextView"
  runat= "Server"/>
```

**Listing 5.26** Zwei Buttons für View-Navigation

### Ereignisse

Auch beim Wechseln zwischen Views verlieren diese dank ViewState nicht ihre vom Benutzer in die Steuerelemente eingegeben Werte. Der Zugriff im Server-Code ist wie gewohnt dauerhaft auf alle Steuerelemente möglich. Sowohl MultiView als auch View haben Ereignisse, die den Statuswechsel signalisieren

Ereignis	Steuerelement	Bedeutung
<i>OnActiveViewChanged</i>	MultiView	Das Ereignis wird ausgelöst, wenn die View gewechselt wurde. Mit <i>GetActiveView</i> lässt sich der Name der neuen View auslesen.
<i>OnDeactivate</i>	View	Das Ereignis wird ausgelöst, wenn die aktuelle View ihre Darstellung verliert.
<i>OnActivate</i>	View	Das Ereignis wird ausgelöst, wenn die aktuelle View ihre Darstellung bekommt.

**Tabelle 5.7** Ausgewählte Ereignisse beim Wechseln von Views



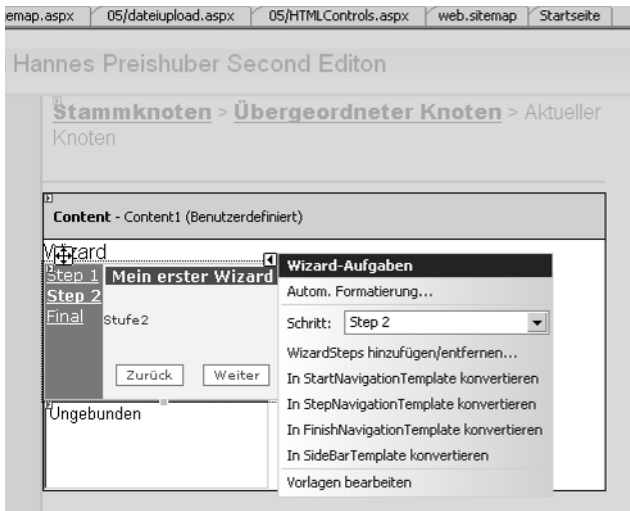
## Wizard-Steuerelement

Benutzer müssen speziell bei komplizierten Vorgängen Schritt für Schritt ans Ziel geführt werden. Ein adäquates Mittel dazu sind Assistenten (*Wizards*). Auch dabei wird ein großes komplexes Formular in mehrere kleine Schritte zerlegt. In Webanwendungen ist dies sehr aufwändig zu programmieren. Da Seiten kein Gedächtnis haben, muss der komplette Status verwaltet werden.

Eine weitere wichtige Eigenschaft ist die Benutzerführung. So kann genau bestimmt werden, ob ein Benutzer einen Schritt zurück machen darf oder eben nicht. Das *Wizard*-Steuerelement bietet dies und viel mehr. Die umfangreich enthaltenen Funktionen reihen dieses Steuerelement in die Reihe der Rich-Steuerelemente ein.

### Wizard-Einstieg

Trotz der Funktionsvielfalt genügt es, das *Wizard*-Steuerelement von der Werkzeugleiste auf das Web-Formular zu ziehen. Dabei werden zwei Schritte im Wizard angelegt, die auch sofort funktionieren.



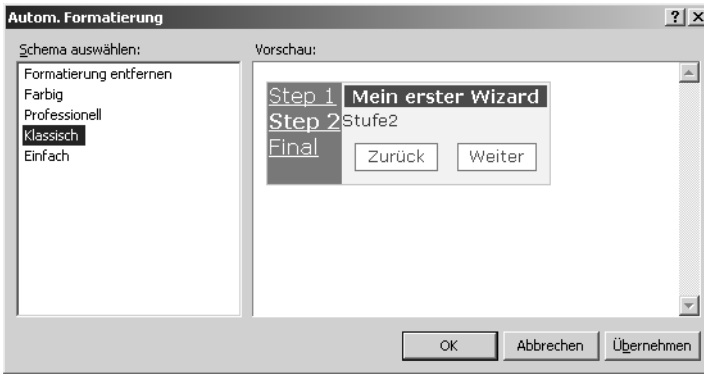
**Abbildung 5.13** Das Wizard-Steuerelement in der Entwicklungsumgebung

Das *Wizard*-Steuerelement besitzt eines der neuen Aufgabenmenüs. Die erstaunlichste Eigenschaft ist aber das visuelle Umschalten zwischen den Wizard-Schritten über die so genannte *Sidebar* des Wizard-Steuerelements direkt in der Entwicklungsumgebung. So können im echten WYSIWIG-Modus die einzelnen Schritte mit Steuerelementen und Text gefüllt werden.

```
<asp:Wizard ID="Wizard1" Runat="server" DisplaySideBar="true" ActiveStepIndex="0">
  <WizardSteps>
    <asp:WizardStep Title="Step 1" Runat="server">
      Stufe1</asp:WizardStep>
    <asp:WizardStep Title="Step 2" Runat="server">
      Stufe2</asp:WizardStep>
  </WizardSteps>
</asp:Wizard>
```

**Listing 5.27** Minimaler Wizard mit zwei Schritten

Beinahe schon selbstverständlich dabei ist der Assistent für die *Automatische Formatierung*. Damit können Sie sich ein ansprechendes Design für Ihren Wizard aussuchen. Besonders gelungen ist dabei die Vorschau. Diese zeigt Ihren Wizard mit den echten Inhalten und kein Pseudo-Template.

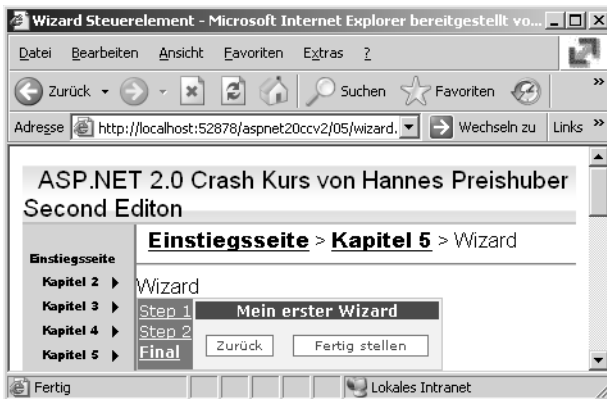


**Abbildung 5.14** Die automatische Formatierung zeigt schon mal die Vorschau

Dadurch werden aus der Vorlage die Style-Elemente den einzelnen Unterelementen zugewiesen, wie hier beispielhaft am *NavigationButtonStyle* gezeigt.

```
<NavigationButtonStyle Font-Names="Verdana" Font-Size="0.8em" BorderStyle="Solid"
  BorderWidth="1px" BorderColor="#507CD1" BackColor="White" ForeColor="#284E98"></NavigationButtonStyle>
```

Mit dem Attribut *HeaderText* kann der Wizard auch noch beschriftet werden.



**Abbildung 5.15** Dieser Wizard wurde mit Autoformat gestaltet

Obwohl eigentlich alles bereits definiert ist, kommt doch der Wunsch nach weiterer Individualisierung auf. Die Beschriftung der Buttons im Bereich Navigation lassen sich mit den Attributen *NextButtonText* bzw. *PreviousButtonText* anpassen. Darüber hinaus lässt sich auch der Typ auf *Button*, *Link* oder *Image* einstellen. Die Sidebar kann im Style weit gehend geändert oder auch komplett abgeschaltet werden (*DisplaySideBar*). Mit Hilfe des Kontextmenüeintrags *Wizard-Aufgaben* lassen sich alle Bereiche in Templates umwandeln.

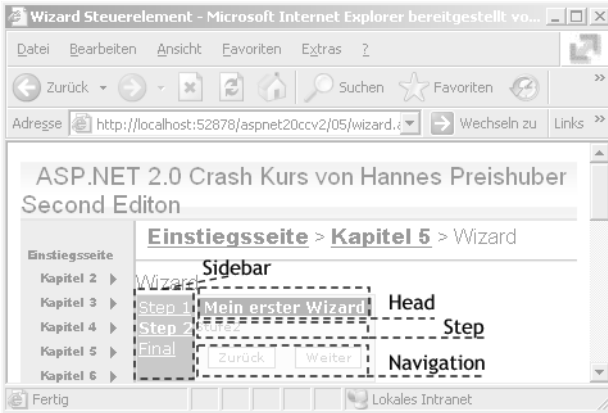


Abbildung 5.16 Die Bereiche des Wizard-Steuerelements

Es obliegt dem Entwickler, wie viele Möglichkeiten er dem Benutzer einräumt – ob beliebig zwischen den einzelnen Schritten hin und her gesprungen werden kann, oder ob nur Schritte nach vorne möglich sein sollen. Alles ist machbar und individuell konfigurierbar.

### Schritte definieren

Für jeden Schritt Ihres Assistenten legen Sie ein *WizardStep*-Element an. Dies geht natürlich am einfachsten über einen Assistenten, den *WizardStep-Auflistungs-Editor*. Um diesen zu starten, wählen Sie aus dem Kontextmenü *Wizard Aufgaben* des *Wizard*-Steuerelements den Menüpunkt *Wizard Steps hinzufügen/entfernen*. Damit können neue Schritte angelegt bzw. entfernt werden. Außerdem lassen sich in den *Step-Eigenschaften* weitere Eigenschaften zum Verhalten der Schritte festlegen.

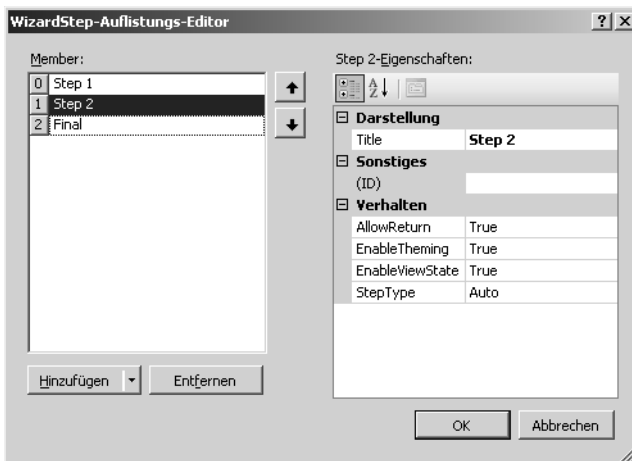


Abbildung 5.17 Der Assistent für das Anlegen von Wizard Steps

In den Eigenschaften kann mit *AllowReturn* der Rückschritt zu einem vorhergehenden Schritt unterbunden werden. Besondere Beachtung verdient die Eigenschaft *StepType*, in der die verschiedenen Typen definiert werden können.

StepType	Bedeutung
<i>Auto</i>	Je nach Reihenfolge der Wizard Steps werden die Buttons automatisch angezeigt.
<i>Complete</i>	Dient zum Anzeigen einer Erfolgs- oder Dankesmeldung. Es werden keine Buttons mehr angezeigt.
<i>Finish</i>	Der letzte Schritt, in dem Benutzereingaben erforderlich sind. Der <i>Finish</i> Button (Schaltfläche <i>Fertig stellen</i> ) wird angezeigt.
<i>Start</i>	Das ist der erste Schritt des Wizards. Es wird deshalb kein <i>Previous</i> Button ( <i>Zurück</i> -Schaltfläche) angezeigt.
<i>Step</i>	Ein gewöhnlicher Schritt des Wizards. Es werden <i>Next</i> - und <i>Previous</i> Buttons angezeigt.

**Tabelle 5.8** Mögliche Attribute von StepType

## Wizard Ereignisprozeduren

Das *Wizard*-Webserver-Steuerelement besitzt einige Ereignisse, die in der Praxis von Bedeutung sind.

Events	Bedeutung
<i>ActiveStepChanged</i>	Dieses Ereignis wird ausgelöst, wenn die Seite des Wizards gewechselt wird.
<i>CancelButtonClick</i>	Dieses Ereignis tritt ein, wenn der Wizard mit dem <i>Cancel</i> Button abgebrochen wird.
<i>FinishButtonClick</i>	Dieses Ereignis tritt ein, wenn der Wizard die letzte Seite erreicht hat und dort der <i>Finish</i> Button gedrückt wurde.
<i>NextButtonClick</i>	Dieses Ereignis tritt ein, wenn der <i>Next</i> Button im Wizard gedrückt wird.
<i>PreviousButtonClick</i>	Dieses Ereignis tritt ein, wenn der <i>Previous</i> Button im Wizard gedrückt wird.
<i>SideBarButtonClick</i>	Dieses Ereignis tritt bei einem Klick auf einen der Punkte in der Sidebar ein.

**Tabelle 5.9** Wizard-Steuerelement-Ereignisse

Die Zuweisung der Prozedurnamen wird über Attribute des *Wizard*-Steuerelements durchgeführt.

```
OnFinishButtonClick="Wizard1_FinishButtonClick">
```

## Auswerten der Historie

Natürlich können Sie per *Validator*-Steuerelement Eingaben prüfen. Aber was ist, wenn ein Benutzer einen Schritt des Wizards einfach überspringt? Keine Sorge, das *Wizard*-Steuerelement zeichnet genau auf, in welcher Reihenfolge, auch mehrmals, welche Schritte aufgerufen wurden.



**Abbildung 5.18** Die Schrittreihenfolge bei der Wizard-Verwendung durch den Benutzer

Dazu dient die Funktion *GetHistory*. Die darin enthaltene Auflistung kann man dann auslesen und über die Eigenschaften *ID* oder *Title* prüfen. In diesem Beispiel wird zur Kontrolle die durchgeführte Schrittreihenfolge in einer *ListBox* ausgegeben.

```
Sub Wizard1_FinishButton_Click(ByVal sender As Object, ByVal e As _
    System.Web.UI.WebControls.WizardNavigationEventArgs)
    Dim steps As ArrayList = CType(Wizard1.GetHistory, ArrayList)
    For Each wstep As WizardStep In steps
        ListBox1.Items.Add(wstep.ID & " Title: " & wstep.Title)
    Next
End Sub
```

**Listing 5.28** Die Auflistung der Wizard Steps

Die eigentlichen Eingabewerte aus den Steuerelementen werden wie in normalen Webformularen üblich ausgelesen.

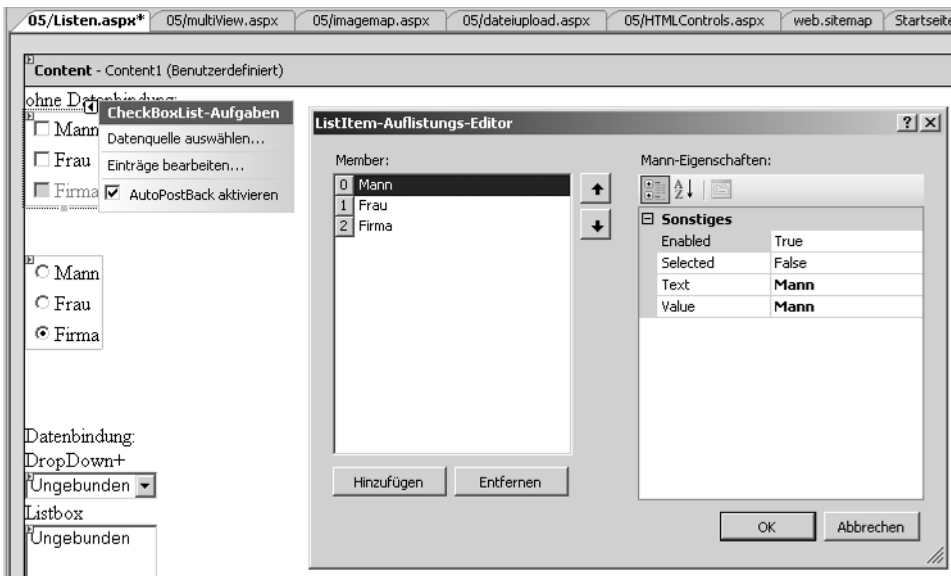
## CheckBoxList-Steuerelement

HTML-Input-Elemente vom Typ *CheckBox* werden durch das *CheckBox* WebServer-Steuerelement erzeugt. Das *CheckBoxList*-Steuerelement fasst mehrere Checkboxes zusammen. So kann direkt eine Bindung der so erzeugten Liste an eine Datenquelle hergestellt werden. Aber auch generell ist dieses Steuerelement recht praktisch zu verwenden. Wenn ohne Datenbindung gearbeitet wird, werden in den *ListItem*-Unterelementen die *CheckBox*-Einträge definiert. Wenn ein Eintrag das Attribut *Enabled=False* besitzt, wird er im Browser deaktiviert (grau) dargestellt. Um bei jedem Klick auf eine der Checkboxes ein RoundTrip zum Server durchzuführen, muss das Attribut *Autopostback* auf *true* gesetzt werden.

```
<asp:CheckBoxList ID="CheckBoxList1" Runat="server" AutoPostBack="True">
  <asp:ListItem>Mann</asp:ListItem>
  <asp:ListItem>Frau</asp:ListItem>
  <asp:ListItem Enabled="False">Firma</asp:ListItem>
</asp:CheckBoxList>
```

**Listing 5.29** Die Checkboxliste

Auch die Möglichkeiten in der Entwicklungsumgebung mit dem Kontextmenü *CheckBox-Aufgaben* und vollständiger visueller WYSIWIG-Darstellung kennzeichnen ein ASP.NET-Web-Steuerelement der zweiten Generation.



**Abbildung 5.19** Checkboxliste mit CheckBocList Aufgaben Kontext Menü

Ein weiterer Assistent hilft beim Erstellen der statischen Einträge in die *CheckBox*-Liste. Der Benutzer kann zur Laufzeit im Browser mehrere dieser Einträge auswählen.

Die Datenbindung dieser und anderer Steuerelemente wird in Kapitel 7 im Detail behandelt.

## RadioButtonList-Steuerelement

Die *RadioButtonList* wird genauso wie die *CheckBoxList* gehandhabt. Einziger Unterschied ist, dass nur ein einzelner Eintrag vom Benutzer zur Laufzeit ausgewählt werden kann.

```
<asp:RadioButtonList ID="RadioButtonList1" Runat="server" AutoPostBack="True">
  <asp:ListItem Value="1">Mann</asp:ListItem>
  <asp:ListItem Value="2">Frau</asp:ListItem>
  <asp:ListItem Value="3" Selected="True">Firma</asp:ListItem>
</asp:RadioButtonList>
```

**Listing 5.30** Das *RadioButtonList*-Web-Steuerelement im Einsatz

## BulletedList-Steuerelement

Auch das *BulletedList*-Steuerelement ist auf den ersten Blick eine Art Liste, die an eine Datenquelle gebunden werden kann. Wenn man etwas näher hinschaut, entdeckt man einige interessante Eigenschaften. Grundsätzlich handelt es sich dabei um eine Aufzählungsliste. Diese kann nummeriert oder mit Punkten gestaltet sein. Die HTML-Darstellung erfolgt dann wie üblich mit `<ul>`- bzw. `<li>`-Elementen.

Der *AuflistungsEditor* und die *ListItems* sind wie beim *OptionList*-Steuerelement zu behandeln. Allerdings hat die *BulletedList* eine Menge weiterer Attribute.

### Listendarstellung

In den Eigenschaften der Liste kann mit *BulletStyle* das generelle Erscheinungsbild der Liste definiert werden.

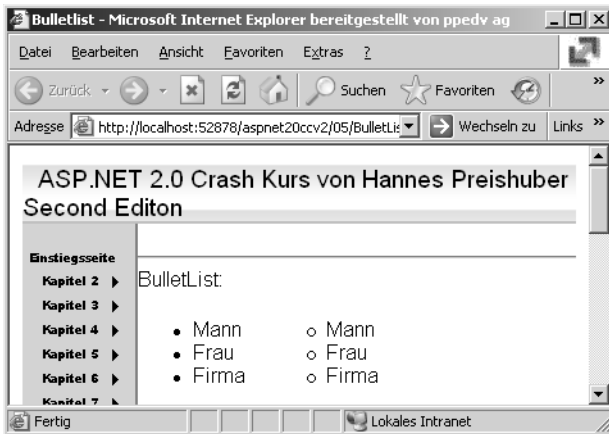


Abbildung 5.20 *BulletStyle Circle* und *Disc* im Einsatz

Je nach verwendetem Wert im *BulletStyle* kann es nötig sein, mit zusätzlichen Attributen in der *BulletedList* zu arbeiten.

BulletStyle	Bedeutung
<i>Circle</i>	Die Bullets werden als Kreise ohne Füllung dargestellt.
<i>CustomImage</i>	Es wird eine eigene Grafik für die Bullet-Symbole verwendet. Der Speicherort wird über <i>BulletImageUrl</i> angegeben.
<i>Disc</i>	Bullets werden als voller Kreis dargestellt, der etwas kleiner als bei <i>Circle</i> ist. Dies ist auch die Voreinstellung, wenn <i>BulletStyle</i> nicht definiert ist.
<i>LowerAlpha</i>	Die Liste wird mit Kleinbuchstaben beginnend bei a »durchnummeriert«.
<i>LowerRoman</i>	Die Liste wird mit römischen Zahlen durchnummeriert. Dabei werden lateinische Kleinbuchstaben verwendet, z.B. <i>xii</i> .
<i>NotSet</i>	<i>NotSet</i> ist identisch mit <i>Disc</i> oder Weglassen des Attributes und erzeugt einen kleinen gefüllten Kreis.
<i>Numbered</i>	Die Liste wird mit 1 beginnend durchnummeriert. Abweichend davon kann mit dem Attribut <i>FirstBulletNumber</i> auch ein anderer Startwert angegeben werden.

Tabelle 5.10 Die verschiedenen Möglichkeiten von Bullet-Typen

BulletStyle	Bedeutung
<i>Square</i>	Die Bullets der Liste werden als kleine gefüllte schwarze Quadrate dargestellt.
<i>UpperAlpha</i>	Die Liste wird mit Großbuchstaben beginnend mit A »durchnummeriert«.
<i>UpperRoman</i>	Die Liste wird mit römischen Zahlen durchnummeriert. Dabei werden lateinische Großbuchstaben verwendet, z.B. <i>XII</i> .

**Tabelle 5.10** Die verschiedenen Möglichkeiten von Bullet-Typen (*Fortsetzung*)

## Auswahl

Die *BulletedList* kann auch als Auswahlliste verwendet werden. Dazu wird das Attribut *DisplayMode* verwendet.

DisplayMode	Bedeutung
<i>Text</i>	Die Einträge der Liste werden als reiner Text angezeigt und sind nicht wählbar.
<i>Hyperlink</i>	Die Einträge der Liste werden als Hyperlinks angezeigt. Die Zieladresse wird im Attribut <i>Value</i> jedes <i>ListItem</i> -Unterelements angegeben.
<i>LinkButton</i>	Die Einträge der Liste werden als <i>LinkButton</i> angelegt. Der Benutzer sieht einen Hyperlink im Browser. Es wird JScript für den Postback erzeugt.

**Tabelle 5.11** Die *BulletedList* kann mit *Displaymode* zur interaktiven Auswahlliste werden

Mit der Ereignisprozedur wird die Behandlung der Benutzerauswahl durchgeführt. Das Ereignis *Click* tritt immer dann auf, wenn ein Eintrag vom Benutzer selektiert wird. Der gewählte Eintrag lässt sich aus dem Prozedur-Argument *e.item* ermitteln. Beachten Sie, dass dieser Index wie üblich bei 0 beginnt.

```
Sub BulletedList4_Click(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.BulletedListEventArgs)
    Label1.Text = "gewählt:" & e.Index.ToString
End Sub
```

**Listing 5.31** Ereignisbehandlung in der *BulletedList*

Mit der ebenfalls möglichen Ereignisprozedur *SelectedIndexChanged* funktioniert das allerdings nicht. Die Eigenschaften *SelectedIndex* oder *SelectedItem* fehlen der *BulletedList*.

Die Zuweisung des Prozedurnamens wird im Attribut *OnClick* der *BulletedList* festgelegt. Achten Sie darauf, dass Attribute die Groß-/Kleinschreibung berücksichtigen. Wenn Sie sich hier vertippen, wird das Ereignis schlicht nicht ausgelöst.

```
<asp:BulletedList ID="BulletedList4" Runat="server"
    BulletStyle="Numbered" DisplayMode="LinkButton"
    OnClick="BulletedList4_Click" >
    <asp:ListItem Value="1">Mann</asp:ListItem>
    <asp:ListItem Value="2">Frau</asp:ListItem>
    <asp:ListItem Value="3">Firma</asp:ListItem>
</asp:BulletedList>
```

**Listing 5.32** Die Deklaration der *BulletedList* in der ASPX-Seite



Das fertige Beispiel kann dann auch schon im Browser benutzt werden. Wie an Abbildung 5.16 in der Statusleiste des Browser zu erkennen ist, wird der Postback per Client-JScript-Funktion ausgeführt (`__doPostBack`).



**Abbildung 5.21** Linke Liste Auswahl per Hyperlink, rechts per PostBack JScript

## Substitution-Steuerelement

Das *Substitution*-Steuerelement ist in seiner Funktion eher untypisch für Web-Steuerelemente. Es dient als Platzhalter für dynamische Inhalte, allerdings nur im Zusammenspiel mit Page Caching. Wenn eine Seite aus dem Cache kommt, wird an der Stelle, an der sich das *Substitution*-Steuerelement befindet, dynamisch HTML-Code eingesetzt. Die Seite ist also nur zum Teil gecacht. Der HTML-Code wird von einer Methode erzeugt, die über das Attribut *MethodName* definiert wird.

```
<asp:Substitution runat="server" MethodName="liveFunktion">
```

Caching und damit auch das *Substitution*-Steuerelement werden im Kapitel 3 genauer erklärt.

# Neue HTML-Server-Steuerelemente

Die Webserver-Steuerelemente sind sehr mächtig und haben einen wesentlich größeren Funktionsumfang als HTML-Steuerelemente. Trotzdem spendiert uns ASP.NET 2.0 auch einige neue HTML-Steuerelemente. Dabei finden sich nicht einmal alle in der Werkzeugleiste von Visual Studio. Sie finden alle HTML-Steuerelemente im *HTMLControl*-Namensraum.

Ein möglicher Einsatzbereich sind von Web-Designern entworfene Seiten. Diese enthalten eben nur INPUT-Elemente und müssen eventuell weiter verwendet werden. Dabei hat ein HTML-Server-Steuerelement genau wie alle anderen Server-Steuerelemente den Zusatz `runat=server`. Wenn ein übliches HTML-Element diesen Zusatz und das *ID*-Attribut erhält wird daraus automatisch ein *HtmlGenericControl*. So lässt sich z.B. ein Meta-Element im Kopf der HTML-Seite zur Laufzeit dynamisch verändern. Aber zunächst beginnen wir mit dem neuen HTML-Head-Steuerelement.

## HTML-Head-Steuerelement

Eigentlich ist das HTML-Head-Steuerelement kein übliches Steuerelement, das mit Drag&Drop in der ASPX-Seite platziert wird. Dem HTML-Element eine ID und den Zusatz `runat=Server` zu geben, ist alles, was zu tun ist. Das funktioniert auch in Masterseiten.

```
<head runat="server" id="Kopf1">
  <title>Untitled Page</title>
</head>
```

**Listing 5.33** Ein Head-Element wird zum Steuerelement

Das Ganze geschieht mit dem Zweck, per Code jederzeit einfach auf den Inhalt des *Head*-Elements zugreifen zu können. So kann z.B. ein zusätzliches Unterelement eingefügt werden – wie ein Link auf ein Stylesheet. Der gewünschte HTML-Code dazu sieht wie folgt aus.

```
<link type="text/css" rel="stylesheet" href="Neu.css" />
```

Um diese Aufgabe zu lösen, wird an die Steuerelements-Auflistung des Header-Steuerelements mit dem Namen *Kopf1* ein neues Steuerelement angehängt. Dieses neue Steuerelement wird als *HtmlLink* erzeugt und dann mit den zusätzlichen Attributen *rel* und *add* versehen.

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myLink As New HtmlLink()
    myLink.Href = "~/05/Neu.css"
    myLink.Attributes.Add("rel", "stylesheet")
    myLink.Attributes.Add("type", "text/css")
    Dim myHead As HtmlHead
    myHead = CType(Master.FindControl("Kopf1"), HtmlHead)
    myHead.Controls.Add(myLink)
End Sub
```

**Listing 5.34** Stylesheet dynamisch einbinden

Über die Eigenschaft *Header* der *Page*-Klasse kann der Header beeinflusst bzw. ausgelesen werden.

```
Page.Header.Title = "Test"
```

**Listing 5.35** Programmatischer Zugriff auf den Header-Bereich

Das funktioniert allerdings nur dann, wenn der Header mit dem Zusatz *runat=server* versehen worden ist. Falls Sie mit Masterseiten arbeiten, muss dementsprechend *Master.Page* verwendet werden.

## HtmlMeta-Element

Um Meta-Informationen in den Kopfbereich der erzeugten HTML-Seite zu schreiben bietet sich das *HtmlMeta*-Steuerelement an. Damit lassen sich nicht nur Meta-Tags für die Suchmaschine, sondern auch Kommandos via http-equiv senden. Ein üblicher Anwendungsfall ist das automatische Neuladen der Seite nach einem definierten Zeitraum. Mit dem Kommando *Refresh* im Attribut *HttpEquiv* und der Intervalldauer in Sekunden im Attribut *Content* veranlassen Sie den Browser, die Seite regelmäßig neu zu laden.

Aufbauend auf Listing 5.32 wird dazu ein *HTMLMeta*-Steuerelement erstellt.

```
Dim tmpMeta As New HtmlMeta()  
tmpMeta.HttpEquiv = "Refresh"  
tmpMeta.Content = "2"  
myHead.Controls.Add(tmpMeta)
```

**Listing 5.36** Automatischer Seitenrefresh alle 2 Sekunden

## HtmlLink

Der Einsatz des *HtmlLink*-Steuerelements wurde in Listing 5.18 bereits gezeigt. Das *HtmlLink*-Steuerelement ist für das Einbinden von Stylesheets in den *Head*-Bereich eines HTML-Dokuments gedacht.

## HtmlTitle

Das *HtmlTitle*-Steuerelement repräsentiert das *Titel*-Element einer Webseite. Der Titel kann auch über die Eigenschaft *Title* der *Page*- oder *Master*-Klasse gesetzt werden.

## HTMLInputPassword

Aus einem HTML-Input-Element wird durch den Zusatz *runat=Server* ein HTML-Server-Steuerelement. Über das Attribut *Type* des Elements wird aus einer Textbox ein Passwortfeld oder ein Submit-Button. Allerdings ist beim HTML-Input (*Text*)-Steuerelement die Option *Type* im Eigenschaftsdialog abgeblendet (grau). Wenn Sie nun im Quellcode der ASPX-Seite *Type* auf *Password* setzen, wird daraus ein *HTMLInputPassword*-Steuerelement. Einfacher geht es in der Entwicklungsumgebung per Drag & Drop aus der Werkzeugleiste. Dort findet sich das Steuerelement.

```
<input id="Password1" type="password" runat="server" />
```

Mit dem HTML-Element *Input (Passwort)* wird ein HTML-Input-Element vom Typ *Password* erzeugt.

Wenn sich der Inhalt des Passwortfeldes ändert, wird das *ServerChange*-Ereignis gefeuert.

```
Sub Password1_ServerChange(ByVal sender As Object, ByVal e As System.EventArgs)  
    Label1.Text = "Passwort geändert"  
End Sub
```

**Listing 5.37** Das *ServerChange*-Ereignis des *HTMLInputPassword*-Steuerelements

Bemerkenswert dabei ist, dass ein Passwortfeld nach einem Submit seinen Inhalt aus Sicherheitsgründen verliert. Trotzdem erkennt das Steuerelement dank *ViewState* eine Änderung der Eingabe. Wenn also das Passwort nach einem Submit verändert wird, wird das Ereignis *ServerChange* gefeuert. Allerdings führt das *Password*-Steuerelement keinen Postback durch; dieser muss von einem anderen Steuerelement wie z.B. Submit ausgelöst werden.

## HTMLInputSubmit

Mit dem HTML-Element *Input (Submit)* wird ein HTML-Input-Element vom Typ *Submit* erzeugt. Normalerweise wird die Seite einfach per Post an den Server gesendet und in *page\_load* behandelt. Es wird aber auch ein *ServerClick*-Ereignis angeboten:

```
Sub Submit1_ServerClick(ByVal sender As Object, ByVal e As System.EventArgs)
    Label1.Text = "ServerClick"
End Sub
```

**Listing 5.38** Das Ereignis des HTMLInputSubmit-Steuerelements

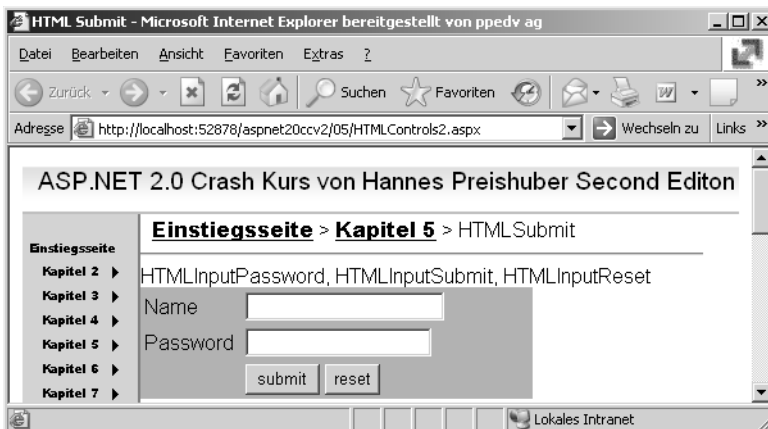
Die HTML-Steuerelemente können auch programmatisch erzeugt werden. Am einfachsten hängen Sie Ihr neu erzeugtes Steuerelement in ein in der ASPX-Seite vorhandenes *PlaceHolder*-Steuerelement:

```
Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim submitButton As HtmlInputSubmit = New HtmlInputSubmit
    submitButton.Value = "Submit"
    Placeholder1.Steuerelemente.Add(submitButton)
End Sub
```

**Listing 5.39** Dynamisch erzeugtes Submit-Element

## HTMLInputReset

Das letzte Steuerelement in der Reihe ist das Element *Input (Reset)*. Mit ihm wird ein HTML-Input-Element vom Typ *Reset* erzeugt. Bei einem Klick auf den so im Beispiel erzeugten *Reset*-Button werden alle Eingabefelder geleert. Es wird dabei kein RoundTrip zum Server durchgeführt.



**Abbildung 5.22** Ein Formular mit HTML-Server-Steuerelementen

## Kapitel 6

# Sicherheit

### **In diesem Kapitel:**

Sicherheit unter ASP.NET	142
Security-Konzept	145
Membership-Objekt	148
Role Manager	156
Anmelden-Steuerelemente	161
Personalisierung mit Profilen	173

# Sicherheit unter ASP.NET

Wenn man von Sicherheit spricht, sind oft die Benutzeranmeldung und die damit verbundenen Rechte gemeint. Gerade bei der Entwicklung eines Sicherheitskonzepts muss ASP.NET den Entwickler bestmöglich unterstützen.

Um eine Benutzeranmeldung durchzuführen, stehen drei Grundkonzepte zur Verfügung.

- Windows-Authentifizierung
- Passport-Authentifizierung
- Forms-Authentifizierung

Bei der Windows-Authentifizierung übernimmt der Browser in Zusammenarbeit mit dem IIS die Anmeldung und die Kontrolle der Rechte. Dazu sind allerdings Einstellungen direkt im IIS und oft auch im Verzeichnisdienst (Active Directory) notwendig. Dies ist für interne Anwendungen manchmal nützlich und auch gewünscht. Benutzerdaten aus dem Firmennetzwerk können so in der Webanwendung automatisch weiterverwendet werden.

Im Interneteinsatz ist das jedoch eher hinderlich. Dann kommt ein zentrales Anmeldesystem wie Microsoft Passport oder eben Ihr eigenes System unter Verwendung der Forms-Authentifizierung zum Einsatz.

Bei der Forms-Authentifizierung wacht ASP.NET über die Benutzeranmeldung und die benötigten Rechte. Anhand eines Benutzer-Tokens erkennt der Webserver, ob es sich um einen angemeldeten Benutzer handelt und erlaubt den Zugriff auf Ressourcen. Bei ASP.NET 1.x wird dieser Token in einem Cookie gespeichert.

---

**ACHTUNG** Da ASP.NET sich um die Sicherheit kümmert, werden mit der Forms-Authentifizierung auch nur solche Dateitypen gesichert, die ASP.NET zugeordnet (gemappt) sind. Das sind z.B. *.aspx*- oder *.asmx*-Dateien, aber **nicht** *.gif*-, *.htm*- oder *.txt*-Dateien.

---

Dabei muss der Entwickler die Benutzeranmeldung generell nicht am Anfang berücksichtigen. Zu jeder Zeit kann das Verhalten der Anwendung über die *web.config* gesteuert werden.

Es genügt zunächst, im Bereich *Authentication* aus den möglichen Modi (*Windows*, *Forms*, *Passport*, *None*) den passenden auszuwählen. Im folgenden Beispiel wird Forms benötigt. Im Unterelement *Forms* wird dann der Name der Login-Seite angegeben, zu der automatisch umgeleitet wird, wenn ein nicht angemeldeter Benutzer eine Seite aufruft. Nach erfolgreichem Login wird der Benutzer dann wieder auf die ursprünglich angeforderte Seite umgeleitet.

Allerdings reicht das alleine als Maßnahme nicht aus. Es muss noch festgelegt werden, dass anonyme, also nicht angemeldete Benutzer, vom Zugriff ausgeschlossen werden. Dies wird per *web.config* im Bereich *authorization* definiert. Sie können Zugriffe pro Benutzer oder Gruppe erlauben (*allow*) oder verbieten (*deny*). Das Fragezeichen »?« steht für anonyme Benutzer, der Stern »\*« für alle.

```
<authentication mode="Forms" >
<forms loginUrl="login.aspx"></forms>
</authentication>
<authorization>
<deny users="?" />
</authorization>
```

**Listing 6.1** Sicherheitseinstellungen in *web.config*

**ACHTUNG** Die Einstellungen für Forms-Authentifizierung sind nur in der Datei *web.config* zulässig, die sich im Webanwendungsstamm befindet. Es ist zwar möglich *web.config*-Dateien in Unterverzeichnissen anzulegen. Dort darf es aber nie den Bereich *Authentication* geben. Im Code-Beispiel, das Sie aus dem Internet laden können, sind deshalb die Einstellungen in ein *<location>*-Element eingebettet.

Die Anmeldung kann dauerhaft (permanent) in einem Cookie gespeichert werden, sodass der Benutzer auch nach Schließen des Browsers beim nächsten Besuch automatisch angemeldet wird. Aus Sicherheitsgründen sollte die Lebensdauer allerdings begrenzt werden. Vermutlich deshalb hat Microsoft auch in letzter Minute diese Einstellung von 50 Jahren auf 30 Minuten geändert.

## Cookieless Forms Authentication

ASP.NET unterstützte seit der ersten Version *SessionIDs*, ohne dabei auf Cookies zurückgreifen zu müssen. Das wurde notwendig, da Cookies einen sehr schlechten Ruf bei den Endanwendern genießen und ihre Verwendung deshalb oft gesperrt ist. Das Problem des Webentwicklers ist, dass er davon eigentlich nichts merkt und deshalb die Anwendung fehlerhaft arbeiten kann.

So wurde ein Lösungsansatz ohne Cookies (*cookieless*) entwickelt. Dabei wird die *SessionID* einfach im URL von Seite zu Seite gereicht. Damit diese ID nicht mit den QueryStrings kollidiert, wird die ID in Klammern in den URL eingebaut, so wie hier konzeptionell gezeigt:

```
/DOTNETCC/(Sessionid1212)/default.aspx
```

Eine eigentlich tolle Sache – sie funktionierte leider bisher nicht im Zusammenhang mit der Forms-Authentifizierung, obwohl dort auch nur ein Session Token weitergereicht wird. Doch wieder geht ein ASP.NET-Entwicklerwunsch mit der neuen ASP.NET 2.0-Version in Erfüllung:

Mit Hilfe des Attributs *cookieless* lassen sich nun verschiedene Modi wählen. Wenn *UseUri* gewählt wurde, ist die *SessionId* im Querystring vorhanden.

```
<forms name=".ASPXAUTH"
loginUrl="06/login.aspx"
defaultUrl="~/default.aspx"
slidingExpiration="true"
cookieless="UseUri"
domain="meindomain"
protection="All"
requireSSL="false"
timeout="50000">
</forms>
```

**Listing 6.2** Aktivierte Forms-Authentifizierung in der Datei *web.config*

Es gibt vier mögliche Werte, die dem Attribut *cookieless* zugewiesen werden können.

Wert	Bedeutung
<i>AutoDetect</i>	Versucht, Cookies zu verwenden und wechselt im Fehlerfall zum URL-Verfahren.
<i>UseCookies</i>	Es werden nur Cookies verwendet.
<i>UseDeviceProfile</i>	Entsprechend der Browser-Konfigurationsdatei wird die Cookie- oder URL-Methode gewählt. Eine weitere Prüfung entfällt.
<i>UseUri</i>	Verwendet immer den URL zum Transport der SessionID.

**Tabelle 6.1** Werte für das Cookieless-Attribut

Wie schon ausgeführt ist die Methode `Cookieless` nicht grundsätzlich neu. Im Zusammenhang mit SessionIDs gibt es diese schon länger. Nach wie vor kann auch eine SessionID in der URL übertragen werden. Damit die beiden nicht kollidieren, werden diese in eigene Klammern gepackt. Ein vorangestellter Buchstabe kennzeichnet den Typ der ID: S für Session und F für Forms.

Session IDs sind übrigens wesentlich kürzer als Forms IDs. Ein kombinierter URL kann dann also so aussehen.

```
http://localhost:47113/ASPNET20CC/(S(gs35ii3pij5mitzc5siahr55)F(sI*taJqdTlkQvC5ocXi-19PdN88Fdau*ukGbu3QTNyEo evCKdfVrpjJE3ZmEs0Q5fId-82W0GRnXik9vdScJLp0JHczezLzBomy9HdAUxpqX1d-n3vw2))/04/default.aspx
```

Eine nicht ganz unbedeutende Neuerung in ASP.NET 2.0 ist, dass der Name der Login-Seite nicht mehr hardcodiert `default.aspx` lautet. In der `web.config` kann durch das zusätzliche Attribut `DefaultUrl` im Forms Element der Name dieser Seite definiert werden.

#### HINWEIS

Gesendete Daten wie Benutzername, Passwort oder SessionID werden im Klartext übertragen. Es sollte deshalb für die Forms-Anmeldung unbedingt eine verschlüsselte Verbindung per HTTPS verwendet werden.

## Benutzer prüfen

Auf der Login-Seite gibt der Benutzer seine Informationen ein, die ihn eindeutig und sicher identifizieren. Meist sind dies ein Benutzername und das dazugehörige Passwort. Diese Daten werden mit den Daten einer Benutzerdatenbank verglichen. Sie können die Benutzer aber auch an anderen Stellen speichern. Sogar in der `web.config` können Benutzerkonten abgelegt werden. Von dieser Methode ist aber generell abzuraten, weil es aufwändig und nicht sicher ist.

Nach einer erfolgreichen Prüfung wird mit der Funktion `RedirectFromLoginPage` der Benutzer angemeldet. Dabei wird mit dem ersten Parameter der Name des Benutzers definiert. Der zweite Parameter bestimmt, ob die Anmeldung dauerhaft sein soll:

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    FormsAuthentication.RedirectFromLoginPage("HannesUser", True)
End Sub
```

**Listing 6.3** Der Benutzer wird zur ursprünglich angeforderten Seite umgeleitet

Im Standardfall wird dann ein Cookie angelegt, das in diesem Beispiel dauerhaft auf der Festplatte gespeichert wird. Wenn der Benutzer das nächste Mal die Webseite aufruft, ist er automatisch angemeldet.



# Security-Konzept

Eines der vielen Highlights von ASP.NET 2.0 sind die Steuerelemente, die sich in der Werkzeugleiste unter dem Abschnitt *Anmelden* befinden. Da sich in den meisten Anwendungen ein Dialog zum An- und Abmelden findet, macht es Sinn, diese wenig aufregende und sich wiederholende Design- und Programmierarbeit zu erleichtern. Die »Anmelden« Webserver-Steuerelemente erlauben es, ohne eine Zeile Code alle üblichen Aufgaben zur Benutzerverwaltung abzubilden. Dennoch sind die Steuerelemente dank ihrer Flexibilität für jedes Szenario geeignet. Die Namen der Steuerelemente sind selbsterklärend.

- CreateUserWizard
- Login
- LoginView
- PasswordRecovery
- LoginStatus
- LoginName
- ChangePassword

All diese Steuerelemente besitzen ein Aufgaben-Kontextmenü. Per *Autom. Formatierung* aus dem *Aufgabenmenü* der Steuerelemente können Sie aus fünf verschiedenen Designvorlagen wählen.

Zusätzliche Attribute stehen in den Eigenschaften zur Verfügung. Mit dem Attribut *Orientation* wird zwischen horizontaler und vertikaler Darstellung gewählt. Über die zahlreichen anderen Attribute können Beschriftungen und Funktionen gewählt werden. Wem das alles nicht reicht, kann sich das Steuerelement im Aufgabenmenü mit dem Menüpunkt *in Vorlage konvertieren* in ein Template und damit in seine einzelnen Bestandteile zerlegen lassen.

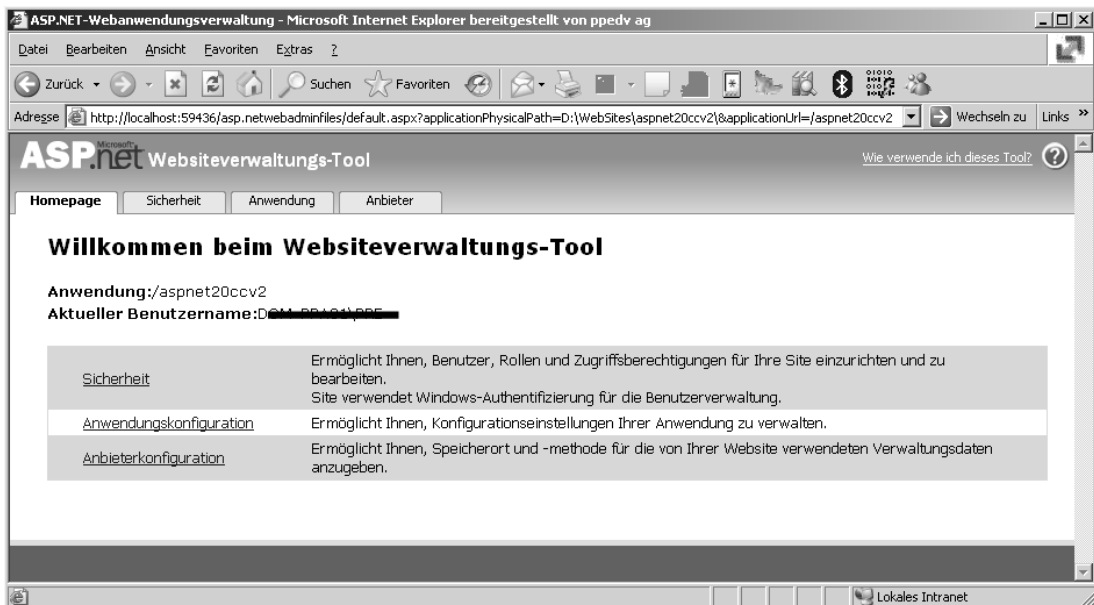


Abbildung 6.1 ASP.NET-Administration aus der Entwicklungsumgebung

Selbst die Verwaltung von Benutzern oder anderen Security-Einstellungen funktioniert in der Entwicklungsumgebung von Visual Studio per Kontextmenü. Zum Starten des ASP.NET Web Site Verwaltungs-Tools wählen Sie im Kontextmenü der Login-Steuerelemente *Website verwalten* oder im Menü *Website- ASP.NET Konfiguration*. Es wird dann eine neue Instanz des Webservers gestartet und diese angezeigt.

Über die Managementkonsole des IIS lassen sich dieselben Einstellungen vornehmen.

## Security-Grundlagen

Wie funktioniert im Hintergrund das Zusammenspiel von Security-Web-Steuerelementen und der Datenbank, in der die Sicherheitsinformationen gespeichert sind, und das ganz ohne Code?

Dafür ist das Provider-Entwurfsmodell für Membership verantwortlich. So ähnlich wie Sie das von OLE DB-Providern kennen, kann so durch Tauschen des Providers auch der Datenbanktyp gewechselt werden. Provider sind ein Kernkonzept von ASP.NET 2.0.

### HINWEIS

In den deutschsprachigen Dialogen von Visual Studio 2005 wird *Provider* mit *Anbieter* bezeichnet.

```
<providers>
  <add name="AspNetSqlMembershipProvider"
    type="System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"
    connectionStringName="LocalSqlServer"
    enablePasswordRetrieval="false"
    enablePasswordReset="true"
    requiresQuestionAndAnswer="true"
    applicationName="/"
    requiresUniqueEmail="false"
    passwordFormat="Hashed"
    maxInvalidPasswordAttempts="5"
    passwordAttemptWindow="10"
    minRequiredPasswordLength="7"
    minRequiredNonalphanumericCharacters="1"
    passwordStrengthRegularExpression="" />
</providers>
</membership>
```

**Listing 6.4** Auszug aus der Datei *machine.config*

In einem weiteren Bereich der *machine.config* sind die Einstellungen für den Connection String zur Datenbank in der die Membership Daten gespeichert sind hinterlegt. Für den Entwickler bleibt nichts weiter zu tun und es funktioniert trotzdem.

```
<connectionStrings>
  <add name="LocalSqlExServer"
    connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=
      |DataDirectory|aspnetdb.mdf;User instance=true"
    providerName="System.Data.SqlClient"/>
</connectionStrings>
```

**Listing 6.5** Vordefinierte Connection Strings in der *machine.config*

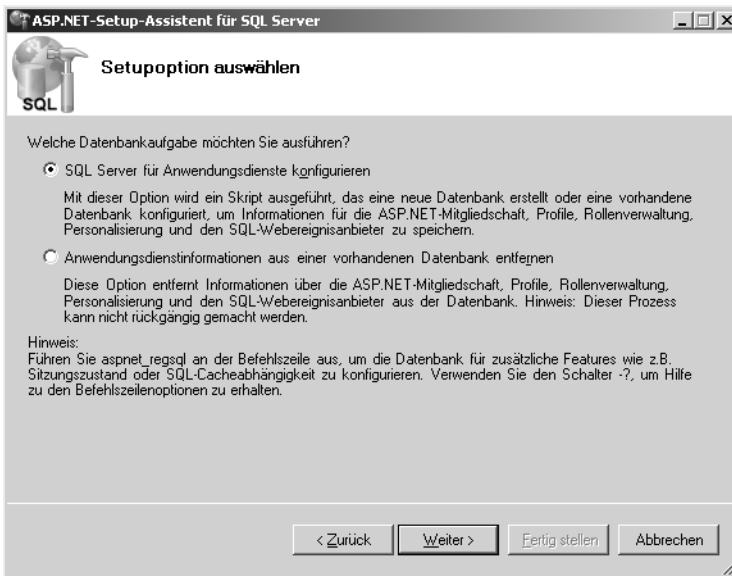
**HINWEIS**

In früheren Builds von Visual Studio 2005 kam ein Access-Provider zum Einsatz. Dieser wurde entfernt und durch einen SQL-Provider ersetzt. Dies erfordert zumindest die Installation von SQL 2005 Express Edition, um die Membership-Funktionen verwenden zu können. Das Entwickler-Team beabsichtigt, einen Access-Provider später nachzuliefern. Zur Verwendung von SQL Express finden sich im Kapitel noch einige Hinweise und Tipps.

Die Benutzer- und Sicherheitsdaten werden damit im *app\_Data*-Verzeichnis der Webanwendung in einer SQL Express-MDF-Datei gespeichert. Sollte die Datenbank nicht vorhanden sein, erzeugt sie der Provider automatisch. Natürlich können Sie diese Einstellungen in der *web.config*-Datei der Anwendung überschreiben und so die Datei z.B. anders benennen. In der Datei *ASPNETDB.MDF* sind nicht nur die reinen Benutzerdaten gespeichert, sondern in mehreren Tabellen auch Informationen zu Membership oder Personalisierung. Je nach verwendetem Provider und Einstellungen werden die Benutzerdaten in unterschiedlichen Datenbanken gespeichert. Nicht alle Provider erzeugen aber ihre benötigten Datenstrukturen komplett selbst.

## SQL Server-Datenbanken vorbereiten

Falls kein SQL Express 2005 vorhanden ist oder mehrere Webanwendungen dieselbe Membership Datenbank verwenden sollen, kann dies manuell konfiguriert werden. Dies ist z.B. auch mit dem SQL Server 2000 möglich. Für den SQL Server muss die Datenbank mit dem Kommandozeilen-Tool *aspnet\_regsql* erzeugt werden. Sie finden die ausführbare Exe-Datei im Verzeichnis *\WINDOWS\Microsoft.NET\Framework\v2.0.50727*. Wenn dieses Programm ohne Parameter gestartet wird, startet der Assistent.



**Abbildung 6.2** Zweiter Schritt des *aspnet\_regsql*-Assistenten

Damit wird eine Datenbank *aspNetDB* samt Tabellen und Stored Procedures angelegt, in der später Benutzer-, Rollen-, Membership- und weitere Informationen gespeichert werden.

Der Connection String in der *web.config* der Anwendung muss anschließend noch entsprechend angepasst werden. Der Membership Provider verwendet gemäß der Konfiguration in der *machine.config* die Zeichenkette mit dem Namen *LocalSqlServer*.

```
<remove name="LocalSqlServer"/>
<add name="LocalSqlServer" connectionString="Server=localhost;User ID=sa;password=;Database=aspnetdb;
Persist Security Info=True"
providerName="System.Data.SqlClient" />
```

**Listing 6.6** Connection String für SQL Server als Membership-Datenbank

## Membership-Objekt

Mit dem Membership-Objekt wird das Verwalten von Benutzern um einiges einfacher. Da alle Funktionen statisch implementiert sind, muss keine Instanz erzeugt werden. Sie können die Membership-Klasse ohne weitere Voraussetzungen verwenden. Die Verbindung zum Datenspeicher wird über einen SecurityProvider hergestellt. Und da die Provider in der *machine.config* sogar schon vorkonfiguriert sind, kann man sofort ohne zusätzlichen Code oder Konfiguration alles verwenden.

### Benutzer anlegen

Wenn noch keine Benutzer vorhanden sind, können diese komfortabel über das WebSite-Verwaltungstool angelegt werden. Auch die Rollen und sogar die Rechte auf die Verzeichnisse können so festgelegt werden.

Das komplette Website-Verwaltungs Tool ist in Kapitel 12 detailliert erläutert.

**HINWEIS** Leider hat das Tool einen Bug. Es ergänzt das *Configuration*-Element in der *web.config* um ein *Namespace*-Attribut. Dann funktioniert in der Entwicklungsumgebung in der *web.config* IntelliSense nicht mehr. Entfernen Sie dieses Attribut, sodass nunmehr ein leeres *<configuration>* Element stehen bleibt: *<configuration xmlns="http://schemas.microsoft.com/.Net-Configuration/v2.0">*



**Abbildung 6.3** Der erste Login-Dialog

Im ersten Schritt wird der Provider festgelegt. Im folgenden Beispiel wird der SQL Server Express 2005 als Datenbank verwendet.



Abbildung 6.4 Websiteverwaltung-Provider-Auswahl

Dadurch wird im SQL Server eine neue Datenbank mit dem Namen *aspnetdb* angelegt. Diese enthält eine Menge Tabellen, die später die Daten zu Benutzern, Rollen, Profilen und Personalisierung enthalten werden.

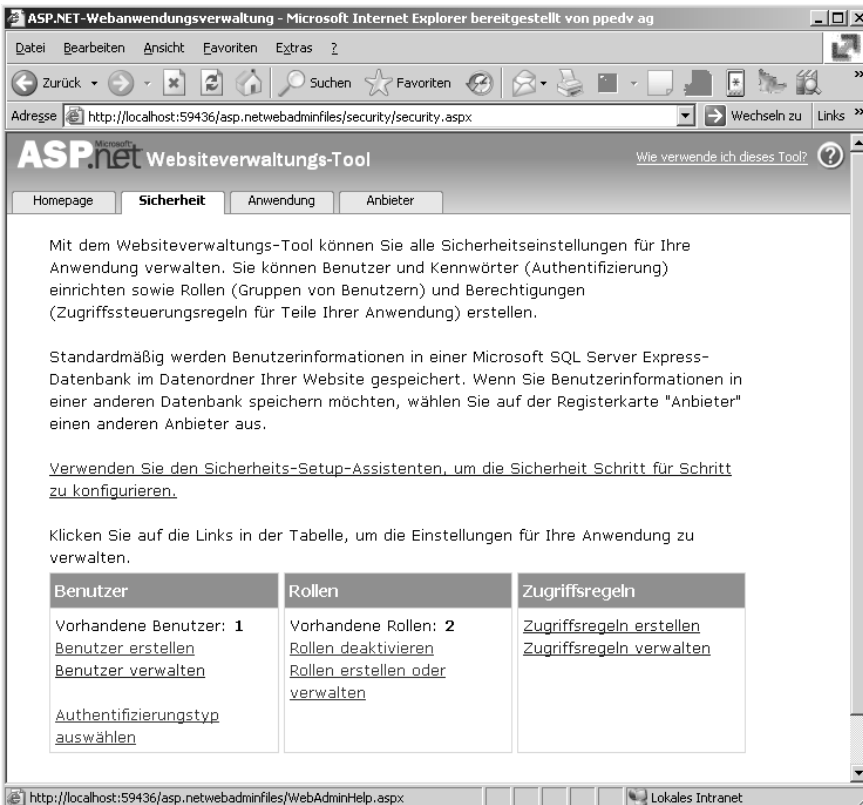


Abbildung 6.5 Benutzerverwaltung mit dem Web Site Administration Tool

Anschließend können unter dem Reiter *Sicherheit* Benutzer angelegt, Rollen zugeordnet und sogar die Rechte auf Verzeichnisse eingerichtet werden. Falls noch nicht geschehen, muss vorher der Authentifizierungstyp auf *aus dem Internet gestellt* werden.

Für die Verwaltung weniger Benutzer ist dieses Werkzeug sehr hilfreich. Wenn es sich um mehr als hundert Benutzer handelt, wird es ein wenig unübersichtlich und auch unpraktisch. Schließlich legen sich die Benutzer z.B. oft auch selbst an (beispielsweise bei Online-Shops oder Auktionsites, um dort mitwirken zu können). Aus Sicherheitsgründen kann aber der Benutzer nicht auf das Websiteverwaltungs-Tool zugreifen. Also wird mit der *MemberShip-API* die Webanwendung um die nötigen Funktionen erweitert.

## Membership konfigurieren

Der Membership-Provider ist in der Datei *machine.config* für den gesamten Webserver bereits vorkonfiguriert. Sie können diese Werte in der *machine.config* direkt für alle Webanwendungen ändern oder dies in der *web.config* einer Anwendung individuell tun.

```
<providers>
<add name=" "
type="System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"
connectionStringName="LocalSqlServer"
requiresUniqueEmail="false"
passwordFormat="Hashed"/>
```

**Listing 6.7** Auszug aus providers Element

**ACHTUNG** Es ist nicht möglich, den Provider einfach in der *web.config* zu überschreiben. Entweder man legt mit *add* einen neuen Namen an oder man entfernt ihn erst mit *Remove*, um ihn dann wieder mit *add* hinzuzufügen.

Der Standardname aus der *machine.config* ist *AspNetSqlMembershipProvider*, der dann auch in der *web.config* weiter verwendet werden kann.

```
<membership defaultProvider="AspNetSqlMembershipProvider" userIsOnlineTimeWindow="15" >
```

In der Standard Konfiguration wird von Microsoft ein Passwort mit mindestens sieben Zeichen Länge, wovon eines ein Sonderzeichen sein muss, vorgegeben. Erfahrungsgemäß haben die Benutzer damit erhebliche Probleme. Wenn es nicht um sicherheitsrelevante Daten geht, oder auch nur für Testzwecke, kann diese Regel in den Providereinstellungen verändert werden. Die beiden Attribute *minRequiredPasswordLength* und *minRequiredNonalphanumericCharacters* werden dazu benötigt.

```
minRequiredPasswordLength="3" minRequiredNonalphanumericCharacters="0"
```

## Benutzer anmelden

Erst nachdem ein Benutzer in der Membership-Datenbank erzeugt wurde, kann sich dieser auch anmelden. Es wird nun mithilfe des *Membership* Providers und im Detail mit der Funktion *Validate* geprüft, ob der Benutzer existiert und das Passwort stimmt. Nach erfolgreicher Prüfung wird die eigentliche Anmeldung, also das Erzeugen des Security Tokens, mit der Funktion *RedirectFromLoginPage* ausgeführt.

```

<%@ Page Language="VB" MasterPageFile="-/masterpage2.master" Title="Login" %>
<script runat="server">
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    If Membership.ValidateUser(name.Text, passwort.Text) Then
        FormsAuthentication.RedirectFromLoginPage(name.Text, True)
    End If
End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">Loginseite
<br />Name<asp:TextBox ID="name" Runat="server"></asp:TextBox><br />Passwort<asp:TextBox
ID="passwort" Runat="server" TextMode="Password"></asp:TextBox>
<asp:Button ID="Button1" OnClick="Button1_Click" Runat="server" Text="Login" /></asp:Content>

```

**Listing 6.8** Login mit Membership Provider

Gemäß den Einstellungen in der *web.config* wird dann nach dem erfolgreichen Login auf eine andere Seite umgeleitet.

**HINWEIS** In Beispiel 6.7 wurde keine Prüfung der Benutzerdaten implementiert. Jede Eingabe wird als erfolgreiche Anmeldung betrachtet.



**Abbildung 6.6** Der erste Login-Dialog

Wenn der Benutzer erfolgreich authentifiziert ist, kann wieder über das *Membership*-Objekt auf Informationen wie den vollen Namen des Benutzers zugegriffen werden. Die Funktion dazu lautet *GetUser.Username*. Mit dem *RoleManager*, der später genauer beschrieben wird, kann darüber hinaus auf die zugeordneten Rollen des Benutzers zugegriffen werden. Über den Index kann dann eine bestimmte Rolle, hier die erste (0) ausgelesen werden.

```

Label1.Text = Membership.GetUser().Username
Label2.Text = Roles.GetRolesForUser()(0)

```

**Listing 6.9** Membership und Roles verwenden



**Abbildung 6.7** Der authentifizierte Benutzer und seine Rollenzugehörigkeit

Wenn der Benutzer in keiner Rolle oder gar nicht authentifziert ist, wird zur Laufzeit ein Fehler ausgelöst. Es empfiehlt sich also, eine entsprechende Fehlerbehandlung mit *Try Catch* zu implementieren.

## Benutzer erstellen

Eine der ersten verwendeten Funktionen des *Membership*-Objekts ist die Funktion *CreateUser*. Diese existiert in vier Überladungen. Die erste Variante erwartet beim Aufruf mindestens Name und Passwort als Parameter und liefert ein *MembershipUser*-Objekt zurück.

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim neuBuerger As MembershipUser = Membership.CreateUser(name.Text, passwort.Text)
End Sub
```

**Listing 6.10** Ein neuer Benutzer mit Passwort wird angelegt

Je nach verwendeter Überladung kommen folgende Parameter zum Einsatz.

Parameter	Verwendung
<i>username</i>	Der Benutzername als String.
<i>password</i>	Das Passwort als String.
<i>Email</i>	Die E-Mail-Adresse als String.
<i>passwordQuestion</i>	Die Passwortfrage als String.
<i>passwordAnswer</i>	Die Passwortantwort als String.
<i>isApproved</i>	Ein boolescher Wert, der angibt, ob der Benutzer überprüft wurde.
<i>Status</i>	Dieser Parameter wird per Referenz übergeben und gibt einen Wert zurück, der verrät, ob der Benutzer erfolgreich angelegt wurde. Der Rückgabetypp ist <i>MembershipCreateStatus</i> .
<i>ProviderUserKey</i>	Der eindeutige Bezeichner für die externe Datenbank, wie Active Directory.

**Tabelle 6.2** Mögliche Parameter der *CreateUser*-Funktion

Nach Ausführen des Kommandos wird der Status per Referenz zurückgeliefert. Die *MembershipCreateStatus*-Auflistung wird dann verwendet, um die Rückgabe zu bearbeiten. Folgende Status-Codes sind möglich.



Statuscode	Bedeutung
<i>DuplicateEmail</i>	E-Mail-Adresse ist bereits vorhanden. Dies tritt nur auf, wenn es in der Provider-Konfiguration entsprechend gefordert wird.
<i>DuplicateProviderUserKey</i>	Der Schlüssel des Benutzers ist in der Datenbank (z.B. Active Directory) bereits vorhanden.
<i>DuplicateUserName</i>	Der Benutzername darf natürlich nur einmal verwendet werden.
<i>InvalidAnswer</i>	Ungültige Formatierung der Antwort.
<i>InvalidEmail</i>	Entsprechend der Formatierungsregeln für E-Mails ungültiger Wert.
<i>InvalidPassword</i>	Ungültiges formatiertes Passwort.
<i>InvalidProviderUserKey</i>	Der Schlüssel für den Benutzer ist ungültig vom Type oder Format.
<i>InvalidQuestion</i>	Ungültige Frage.
<i>InvalidUserName</i>	Der Benutzername ist ungültig und verletzt die Regeln.
<i>ProviderError</i>	Der Provider meldet einen anderen Fehler, z.B. bei der Kommunikation mit der Datenbank.
<i>Success</i>	Der Benutzer wurde erfolgreich angelegt.
<i>UserRejected</i>	Benutzer nicht authentifiziert.

**Tabelle 6.3** *MembershipCreateStatus*-Auflistung

## Benutzer suchen

Wenn Benutzer per Code oder Websiteverwaltung angelegt worden sind, möchte man auch wieder auf diese Daten zugreifen. Eine mögliche Funktion dazu ist *GetUser*. Damit erhält man Zugriff auf das *MembershipUserObjekt*. Mit diesem kann der aktuelle oder ein per Parameter gewählter Benutzer ausgelesen werden. Außerdem kann die Eigenschaft *TimeStamp*, die Auskunft über die letzte Aktivität des Benutzers gibt, gelesen oder aktualisiert werden.

Aus dem *User*-Objekt kann dann die Eigenschaft *UserName* den Namen des Benutzers auslesen. Die Funktion *GetUser* gibt es in sechs Überladungen.

```
Dim benutzer As String = Membership.GetUser().UserName
'Membership.GetUser(True)
'Membership.GetUser("hannes")
'Membership.GetUser("hannes", True)
```

**Listing 6.11** Überladungen von *GetUser*

Eine weitere und einfachere Möglichkeit besteht in der Suche nach einem Benutzer unter Angabe seiner genauen E-Mail-Adresse. Die Funktion lautet *GetUserNameByEmail*. Dabei erhält man im Erfolgsfall genau einen Benutzer zurück.

```
Label1.Text = Membership.GetUserNameByEmail("hannes@ppedv.de")
```

Es gibt drei weitere Methoden zum Finden von Benutzern, die dann eine Auflistung der gefundenen Benutzer zurückliefern. Das Schöne daran ist, dass man das Ergebnis mit ein wenig Geschick direkt an ein Webserver-Steuerelement binden kann. Die Funktion *FindUsersByEmail* sucht mithilfe der E-Mail-Adressen.

```

ListBox1.DataSource = Membership.FindUsersByEmail (TextBox1.Text)
ListBox1.DataTextField = "UserName"
ListBox1.DataBind()

```

**Listing 6.12** Benutzer werden gesucht und an eine Listbox gebunden

Als Parameter wird die E-Mail-Adresse oder ein Match-String übergeben. Das Wildcard-Zeichen ist hier, ähnlich der SQL-Syntax, ein Prozentzeichen.



**Abbildung 6.8** Alle Benutzer werden in einer Listbox angezeigt

Natürlich kann auch per Name in der Benutzerliste gesucht werden. Dabei kommt die Funktion *FindUsersByName* zum Einsatz.

```
Membership.FindUsersByName()
```

Möchte man alle Benutzer ermitteln, verwendet man die Funktion *GetAllUsers*.

```
Membership.GetAllUsers()
```

In der folgenden Tabelle werden die Eigenschaften der *MembershipUser*-Klasse beschrieben.

Eigenschaft	Verwendung
<i>Comment</i>	Kommentarinformation zum Benutzer.
<i>CreationDate</i>	Liest oder schreibt das Datum, wann der Benutzer im Membership-Verzeichnis angelegt wurde.
<i>Email</i>	Liest oder schreibt die E-Mail-Adresse des Benutzers.
<i>IsApproved</i>	Liest oder schreibt einen booleschen Wert, der angibt, ob sich der Benutzer anmelden darf.
<i>IsOnline</i>	Zeigt, ob ein Benutzer online ist.
<i>IsLockedOut</i>	Prüft, ob der Benutzer gesperrt ist z.B. durch zu häufige Fehleingabe des Passwortes.

**Tabelle 6.4** Eigenschaften von *MembershipUser*

Eigenschaft	Verwendung
<i>LastActivityDate</i>	Liest oder schreibt das Datum des letzten Zugriffs.
<i>LastLoginDate</i>	Liest oder schreibt, wann sich der Benutzer das letzte Mal angemeldet hat.
<i>LastLockOutDate</i>	Zeigt an, wann der Benutzer gesperrt wurde.
<i>LastPasswordChangedDate</i>	Liest oder schreibt, wann das Passwort das letzte Mal geändert worden ist.
<i>PasswordQuestion</i>	Damit erhält man die Passwortfrage des Benutzers.
<i>Provider</i>	Damit erhält man eine Referenz auf den Membership Provider.
<i>ProviderUserKey</i>	Liest die eindeutige ID vom Datentyp-Objekt, die einen Benutzer kennzeichnet.
<i>UserName</i>	Liest den Benutzernamen aus dem Membership Store.

**Tabelle 6.4** Eigenschaften von *MembershipUser* (Fortsetzung)

## Benutzer-Update und -Delete

Mit dem *Membership*-Objekt kann man auf die Benutzerdaten zugreifen, diese auslesen und wieder zurück schreiben. Dazu muss man eine Instanz des aktuellen Benutzerobjektes ermitteln. In diesem Objekt können dann die Attribute gelesen, dargestellt und wieder geschrieben werden. Einzig die Eigenschaft *UserName* lässt sich nicht ändern.

```
<script runat="server">
Dim mu As MembershipUser
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    'nicht möglich!
    mu.UserName = txtName.Text
    mu.Email = txtemail.Text
    Membership.UpdateUser(mu)
End Sub
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    mu = Membership.GetUser("hannes")
    If Not IsPostBack Then
        txtemail.Text = mu.Email
        txtName.Text = mu.UserName
    End If
End Sub
</script>
```

**Listing 6.13** Lesen und Schreiben der Membership-Daten

Das Löschen des Benutzers funktioniert mit der *DeleteUser*-Funktion des *Membership*-Objektes. Als Parameter wird der Benutzername angegeben. In einer Überladung existiert ein zweiter Parameter, der angibt, ob verknüpfte Daten gelöscht werden sollen.

```
Membership.DeleteUser("hannes", True)
```

**Listing 6.14** Das war's wohl für »hannes«

## Passwörter

Die Handhabung der Passwörter erfolgt auch aus Sicherheitsgründen anders als der Änderungsvorgang der übrigen Benutzerdaten. Dazu wird direkt das Provider-Objekt des Membership-Objektes verwendet. Je nach

Einstellung kann dies auf unterschiedliche Arten passieren. Eine einfache Änderung des Passwortes erfolgt mit der Funktion *ChangePassword*, die per boolescher Rückgabe den Erfolg meldet.

```
Membership.Provider.ChangePassword("hannes", "altes pwd", "neues pwd")
```

Wenn die Sicherheitsfrage und -antwort geändert werden soll, verwendet man die Funktion *ChangePasswordQuestionAndAnswer*.

```
Membership.Provider.ChangePasswordQuestionAndAnswer(User.Identity.Name, _
    txtPassword.Text, txtQuestion.Text, txtAnswer.Text)
```

Die im Membership-Provider vorhandenen Funktionen sind in Tabelle 6.5 aufgeführt.

Funktion	Verwendung
<i>ChangePassword</i>	Ändert das Passwort eines Benutzers.
<i>ChangePasswordQuestionAndAnswer</i>	Ändert die Passwortfrage des Benutzers.
<i>GetHashCode</i>	Damit erhält man den HashCode.
<i>GetPassword</i>	Damit erhält man das Passwort des Benutzers aus dem <i>Membership</i> -Objekt.
<i>ResetPassword</i>	Setzt das Passwort zurück und erzeugt dadurch ein neues.
<i>GetNumberOfUsersOnline</i>	Mit dieser Funktion erhält man die Anzahl der angemeldeten Membership Benutzer.
<i>UnlockUser</i>	Entsperrt den als Parameter angeführten Benutzer.

**Tabelle 6.5** Auszug aus den Funktionen des MembershipUser-Provider-Objektes

## Role Manager

Der Role Manager arbeitet ganz ähnlich wie *Membership*. Allerdings ist die Verwendung von Rollen in der Voreinstellung per *machine.config* deaktiviert und muss folglich in der *web.config* der Anwendung bei Bedarf aktiviert werden.

```
<roleManager enabled="false" ...
```

Benutzer können einer oder mehreren Rollen zugewiesen werden. Zugriffe und Rechte können dann anhand von Rollen gesteuert werden. Der einfachste Weg, um Rollen anzulegen und Benutzer hinzuzufügen, bietet die integrierte Website-Administration.

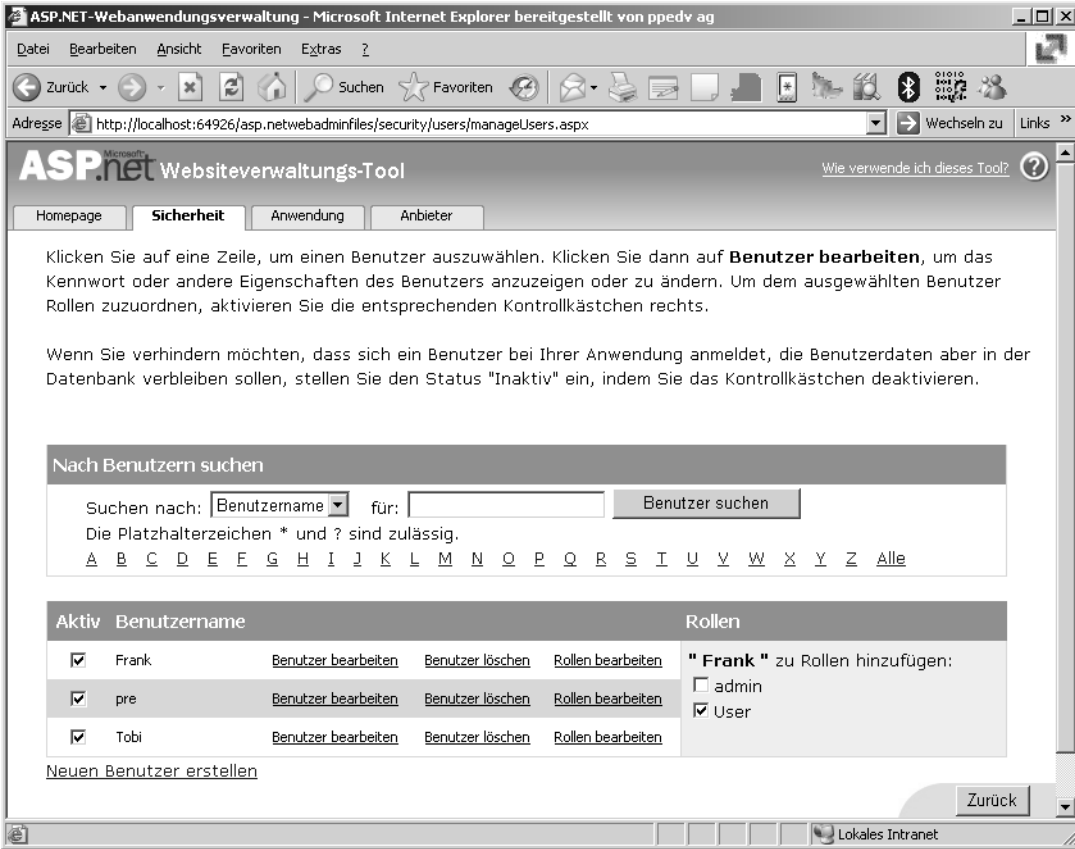


Abbildung 6.9 Rollen mit dem Websiteverwaltungs-Tool verwalten

Grundsätzlich ist es möglich, auch Benutzer zu Rollen hinzuzufügen, die im Membership-System nicht vorhanden sind.

## Rollen aktivieren

Um den Rollen-Manager zu aktivieren, muss in der *web.config* mindestens das Element *roleManager* mit dem Attribut *enabled=true* versehen und ein Provider definiert werden. Der *AspNetSqlRoleProvider* ist standardmäßig per *machine.config* eingerichtet.

```
<roleManager enabled="true" defaultProvider="AspNetSqlRoleProvider" />
```

Daneben gibt es noch folgende weitere Attribute:

Attribut	Verwendung
<i>cacheRolesInCookie</i>	Mit diesem booleschen Wert wird definiert, ob die Rollen des Benutzers in einem Cookie gecacht (zwischen gespeichert) werden dürfen. Dadurch werden Anfragen an den Role Provider reduziert.
<i>cookieName</i>	Name des erstellten Cookies.
<i>cookiePath</i>	Setzt den Pfad, in dem der Cookie gültig ist.
<i>cookieRequireSSL</i>	Mit diesem booleschen Wert wird definiert, ob zum Übertragen des Cookies ein geschützter SSL-Kanal verwendet werden muss.
<i>cookieTimeout</i>	Setzt die Lebensdauer des Cookies in Minuten.
<i>createPersistentCookie</i>	Mit diesem booleschen Wert wird der Cookie dauerhaft gesetzt.
<i>cookieName</i>	Mit diesem String wird der Name des Cookies definiert, in dem die gecachten Rollen gespeichert werden. Dazu muss <i>enabled</i> auf <i>true</i> gesetzt sein.
<i>cookieProtection</i>	Definiert, wie der Inhalt des Cookies geschützt werden soll. Mögliche Werte sind: <i>All</i> , <i>None</i> , <i>Encryption</i> und <i>Validation</i> .
<i>cookieSlidingExpiration</i>	Dieser boolesche Wert definiert, ob die Cookie-Lebensdauer mit jedem Cookie-Zugriff neu beginnt.
<i>defaultProvider</i>	Dieser String definiert den verwendeten Provider.
<i>Enabled</i>	Definiert als boolescher Wert, ob der Role Manager aktiviert ist.
<i>Domain</i>	Erzeugt das Domain-Attribut im Cookie.
<i>maxCachedResults</i>	Maximale Anzahl (vom Typ <i>Integer</i> ) von Rollen, die in einem Cookie gespeichert werden.

**Tabelle 6.6** Attribute des Role Managers in der *web.config*

## Rollen zuweisen und entfernen

Im nächsten Schritt können nun Benutzern Rollen zugewiesen werden. Verwendet werden das *Roles*-Objekt und eine ihrer Funktionen. Es gibt vier passende Methoden, die eine oder mehrere Rollen einem oder mehreren Benutzern zuweisen. Die Namen sind *AddUserToRole*, *AddUserToRoles*, *AddUsersToRole* und *AddUsersToRoles*. Folgendes Code-Schnipsel weist dem Benutzer eine Rolle zu, die aus einer Listbox selektiert wurde.

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Roles.AddUserToRole(TextBox1.Text, ListBox1.SelectedValues)
End Sub
```

**Listing 6.15** Benutzer zu Rolle hinzufügen

Zum Entfernen von Rollen gibt es vier ähnliche Funktionen. Sie nennen sich *RemoveUserFromRole*, *RemoveUserFromRoles*, *RemoveUsersFromRole* und *RemoveUsersFromRoles*.

```
Dim users(2) As String
users(0) = "hannes"
users(1) = "tobi"
users(2) = "shinja"
Roles.RemoveUsersFromRole(users, "test")
```

**Listing 6.16** Mehrere Benutzer aus einer Rolle entfernen

## Rollen auslesen

Weitere Funktionen aus dem *Role*-Objekt erlauben verschiedene Leseoperationen im Zusammenhang mit den Rollen. Mit der Funktion *RoleExists* kann geprüft werden, ob eine Rolle vorhanden ist. Soll geprüft werden, ob ein spezieller Benutzer einer Rolle zugeordnet ist, kommt *IsUserInRole* zur Anwendung. Schließlich gibt es noch zwei weitere Funktionen, die entweder alle Rollen eines Benutzers oder alle Benutzer in einer Rolle zurückgeben (*GetRolesForUser*, *GetUsersInRole*). Die Rückgabe in beiden Fällen erfolgt per String-Array – genau wie bei *GetAllRows*, durch das alle Rollen zurückgeliefert werden.

Ein solches Array kann dann auch an ein Anzeigeelement wie eine Listbox gebunden werden. In diesem Beispiel werden alle Rollen in einer Listbox angezeigt.

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    If Not IsPostBack() Then
        ListBox1.DataSource = Roles.GetAllRoles()
        ListBox1.DataBind()
    End If
End Sub
```

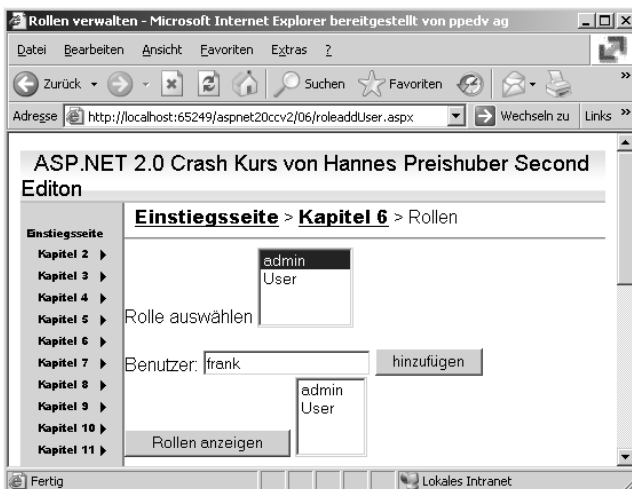
**Listing 6.17** Anzeigen aller definierten Rollen

In einer Textbox wird dann ein Benutzer angegeben und der ausgewählten Rolle hinzugefügt. Die gleiche Textbox dient auch als Quelle zur Anzeige der Benutzerrollen. Zur Anzeige wird eine zweite Listbox verwendet.

```
Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    ListBox2.DataSource = Roles.GetRolesForUser(TextBox1.Text)
    ListBox2.DataBind()
End Sub
```

**Listing 6.18** Benutzerrollen anzeigen

Die selbst erstellte Rollenverwaltung steht dann dem Administrator zur Verfügung.



**Abbildung 6.10** Rollenverwaltung

## Rollen erstellen und löschen

Ein weiterer Aufgabenbereich stellt das Erstellen und Löschen von Rollen dar. Auch dieser Job gestaltet sich vergleichsweise einfach. Die entsprechenden Funktionen im Role-Objekt heißen *DeleteRole* und *CreateRole*. Im Beispiel zeigt eine Listbox alle vorhandenen Rollen an. Per Eingabe des Namens der neuen Rolle in eine Textbox kann eine solche angelegt werden.



Abbildung 6.11 Erstellen einer neuen Rolle

Es kommt hier eine selbst erstellte Hilfsfunktion *Bind* zum Einsatz, um das *Listbox*-Steuerelement an die Rollendaten zu binden. Da zuerst die Methode *Page\_Load* und dann erst die Ereignisbehandlung des Buttons ausgeführt wird, geht die Auswahl des Benutzers aus der Liste verloren, wenn die Datenbindung nun nur in *Page\_Load* vorgenommen wird. Deshalb wird nach dem Ändern der Rollen umgehend eine neue Bindung der Auflistung durchgeführt.

```
<script runat="server">
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Roles.DeleteRole(ListBox1.SelectedValue)
    bind()
End Sub
Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Roles.CreateRole(TextBox1.Text)
    bind()
End Sub
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    If Not IsPostBack Then
        bind()
    End If
End Sub
Private Sub bind()
    ListBox1.DataSource = Roles.GetAllRoles()
    ListBox1.DataBind()
End Sub
</script>
```

Listing 6.19 Rollen anlegen und löschen



## Programmsteuerung mit Rollen

Auch die Menüführung kann anhand der Rollen bzw. der Rollenzugehörigkeiten gesteuert werden. Das heißt ein Benutzer sieht nur den Menüpunkt, auf den seine Rolle zugreifen kann. Zunächst sieht jeder Benutzer jeden Menüpunkt.

Um das zu ändern, muss in der *web.config* beim SitemapProvider das Attribut *securityTrimmingEnabled* auf *true* gesetzt werden. Dies ermöglicht erst die Sicherheitsfunktion.

```
<providers>
<add name="meinSiteMapProvider" securityTrimmingEnabled="true" ></add>
```

**Listing 6.20** Rollensteuerung einschalten in *web.config*

In der Sitemap-Datei wird in jedem *siteMapNode* mit dem Attribut *roles* die Rolle zugewiesen, die der Benutzer haben muss. Wenn mehrere Rollen zutreffen sollen, werden diese mit Kommas getrennt angegeben.

```
<siteMapNode url="admin.aspx" title="Help" roles="SuperAdmin">
  <siteMapNode url="subadmin.aspx" title="irgendwas" />
</siteMapNode>
...
```

**Listing 6.21** Rollen-Definitionen in der Sitemap-Datei

Wenn kein Benutzer die Rolle *SuperAdmin* besitzt, ist der Menü-Eintrag nicht mehr sichtbar.

---

**HINWEIS** Dies unterbindet lediglich die Anzeige des Menüpunktes. Die Zugriffsrechte auf Dateien oder Pfade müssen in der *web.config* extra geregelt werden. Das kann per *<location path="Admin.aspx">* geregelt werden.

---

## Anmelden-Steuerelemente

Ganze sieben Steuerelemente im Bereich Anmelden (Login) der Werkzeugleiste von Visual Studio decken die Laufzeitfunktionalität des Membership Providers ab. Mit einer visuellen Oberfläche lassen sich zur Entwurfszeit die Einstellungen vornehmen. Die Steuerelemente sind aus weiteren Steuerelementen wie Textboxen oder Buttons zusammengesetzt. Design-Vorlagen erlauben ein schnelles und einheitliches Layout. Wenn das nicht reicht, kann alternativ mit Benutzer-Templates jedes Detail im Design verändert werden. Allen Steuerelementen ist gemeinsam, dass sie ohne Code und nur per Konfiguration in der *web.config* auskommen.

### CreateUserWizard

Ein neuer Benutzer soll eventuell die Möglichkeit haben, sich selbst anzulegen. Dazu wird ein mehr oder minder aufwändiges Formular verwendet, das dem Benutzer diese Funktionalität zur Verfügung stellt. Es beinhaltet die Prüfung der Benutzereingaben und das abschließende Speichern der Daten. Die Anzahl der Optionen und Dialoge ist so umfangreich, dass hier keine vollständige Beschreibung erfolgen kann. Das *Cre-*

*ateUserWizard*-Steuerelement wird als einziges ausführlicher beschrieben. Die weiteren Anmelde-Steuerelemente nur im Wesentlichen. Alleine das *CreateUserWizard*-Steuerelement beinhaltet 144 Eigenschaften, 17 geschützte Eigenschaften, 27 Methoden, 59 geschützte Methoden und 22 Ereignisse. Aus diesem Grund soll es ausreichen, hier nur einige ausgewählte Highlights zu beschreiben.

Ziehen Sie aus der Werkzeugleiste das *CreateUserWizard*-Steuerelement auf ein leeres Web-Formular.

Dadurch erzeugt die Entwicklungsumgebung eigentlich nur wenige wichtige Zeilen in der *aspx*-Seite, die allerdings schon die komplette Funktionalität abdecken.

```
<asp:CreateUserWizard ID="CreateUserWizard1" Runat="server">
  </asp:CreateUserWizard>
```

**Listing 6.22** Die Source-Ansicht der ASPX-Seite

Der Einfachheit halber wurden die *WizardSteps*-Unterelemente hier nicht aufgeführt, da sie auch für diese Funktion nicht nötig sind.

Um diese Seite für den normalen Benutzer ohne Anmeldung aufrufbar zu machen, muss diese von der Authentifizierung ausgenommen werden. Dafür ist das *<location>*-Element in der *web.config* der Anwendung am besten geeignet.

```
...
</system.web>
<location path="06/createuser.aspx">
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</location>
</configuration>
```

**Listing 6.23** Zugriffsrechte per *web.config* definieren

Der Assistent verfügt in der Standardeinstellung über zwei Assistentenschritte – der Anmeldung und der Erfolgsmeldung. Es können bei Bedarf noch mehrere Schritte eingefügt werden, um z.B. komplexere Registrierungsvorgänge über mehrere Schritte hinweg durchzuführen.

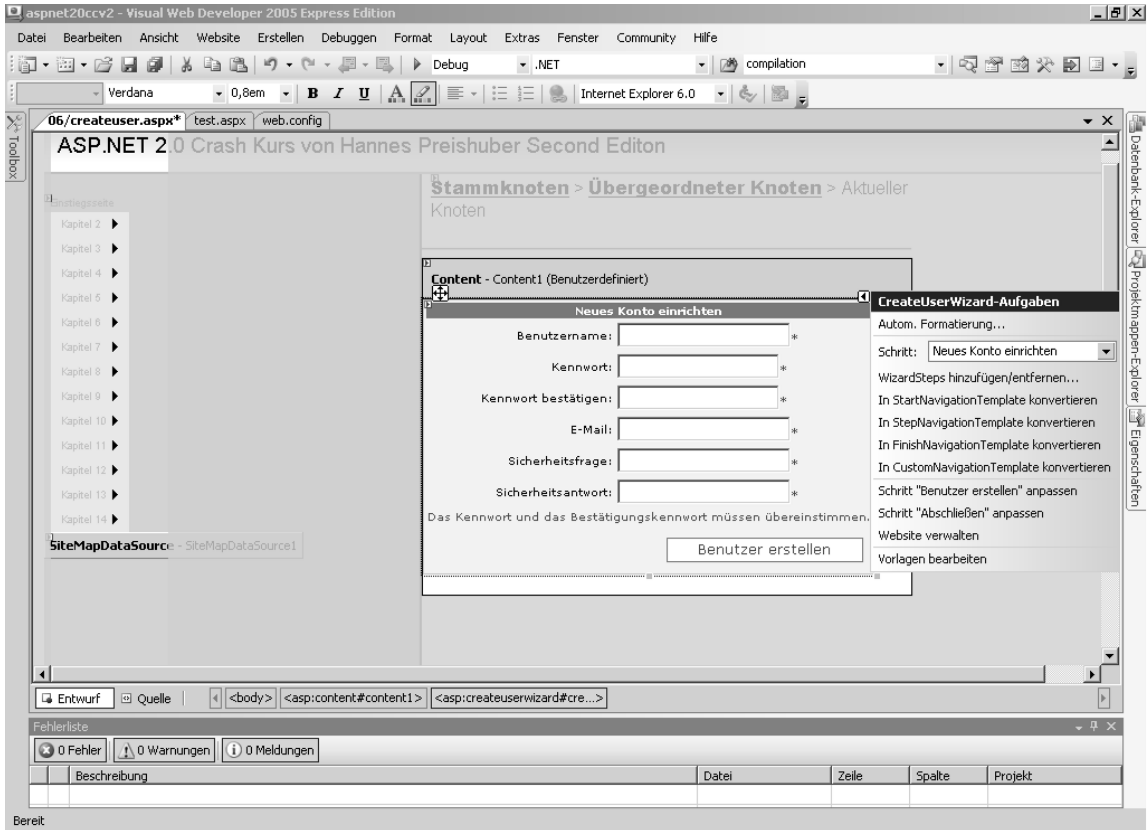


Abbildung 6.12 Das CreateUserWizard-Web-Steuerelement

Jeder Schritt und jeder Bereich lässt sich über Vorlagen (Templates) bis ins Detail steuern. Die einzelnen Dialoge lassen sich in ein *Custom Template* umwandeln. Dazu verwenden Sie das Aufgabenmenü.

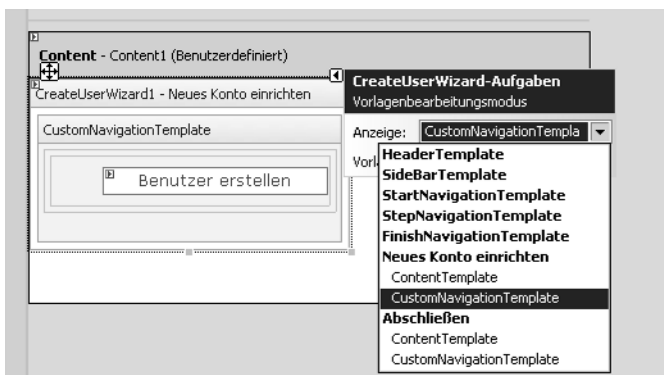


Abbildung 6.13 Der CreateUserWizard wird angepasst

So wird recht umfangreicher HTML-Code ergänzend zu Listing 6.23 in der ASPX-Seite erzeugt.

```

<asp:CreateUserWizardStep runat="server">
  <CustomNavigationTemplate>
    <table border="0" cellspacing="5" style="width: 100%; height: 100%">
      <tr align="right">
        <td align="right" colspan="0">
          <asp:Button ID="StepNextButton" runat="server" BackColor="White" BorderColor="#507CD1"
            BorderStyle="Solid" BorderWidth="1px" CommandName="MoveNext" Font-Names="Verdana"
            ForeColor="#284E98" Text="Benutzer erstellen" ValidationGroup="CreateUserWizard1" />
        </td>
      </tr>
    </table>
  </CustomNavigationTemplate>
</asp:CreateUserWizardStep>

```

**Listing 6.24** Ein Template innerhalb des CreateUserWizards

Wichtig ist, dass die IDs der Steuerelemente gleich bleiben, denn dies steuert die ausgeführte Aktion, wie z.B. hier *StepNextButton*.

Durch Hinzufügen weiterer Wizard-Schritte kann dann ein kompletter Registrierungsprozess abgehandelt werden. Weitere Daten wie z.B. Adresse oder Bankverbindung werden darin erfasst.

Die Speicherung dieser zusätzlichen Daten kann dann ebenfalls über die ASP.NET 2.0-Profile in der Membership-Datenbank mit wenig Aufwand erfolgen. Profile werden ein wenig später in diesem Kapitel genauer beschrieben.

Es lässt sich auch eine E-Mail als Bestätigung versenden. Die Einstellungen zum Versand erfolgen im Element *<MailDefinition >*. Dies wird später in diesem Kapitel beim Steuerelement *PasswordRecovery* beschrieben.

Im Folgenden werden die weiteren Steuerelemente vorgestellt, die Security-Aufgaben ohne Code verwirklichen. Leider fehlt der Platz für eine detaillierte Beschreibung aller Features; aus diesem Grund werden nur die wichtigsten vorgestellt.

## Login-Steuerelement

Mit dem *Login*-Steuerelement kann eine Benutzeranmeldung erfolgen. Bereits mit den Standard Attributen können die meisten Einstellungen vorgenommen werden. Das hier gezeigte Beispiel verwendet einige der Attribute exemplarisch. Mit *DisplayRememberMe* wird gesteuert, ob die Checkbox angezeigt wird, mit der es möglich ist, eine dauerhafte Anmeldung zu erzeugen. Die angezeigten Texte werden über Attribute wie *FailureText*, *LoginButtonText* oder *InstructionText* festgelegt. Auch kann die horizontale oder vertikale Ausgabe mit dem *Orientation*-Attribut erzeugt werden. Mit *UserName* kann der Name für die Anmeldung vorgebelegt werden. Soll der Login-Dialog nach erfolgreicher Authentifizierung weiter angezeigt werden, setzt man *VisibleWhenLoginID* auf *true*.

```

<asp:Login ID="Login1" Runat="server" BorderWidth="1px" BorderColor="#B5C7DE" BorderStyle="Solid"
  BackColor="#EFF3FB" Font-Names="Verdana" Font-Size="0.8em"
  FailureText="Anmeldung fehlgeschlagen"
  InstructionText="Hier bitte anmelden<br>mit Benutzernamen"
  LoginButtonText="anmelden"
  TitleText="Benutzer anmeldung"
  DisplayRememberMe="true"

```

**Listing 6.25** Login-User-Steuerelement im HTML View

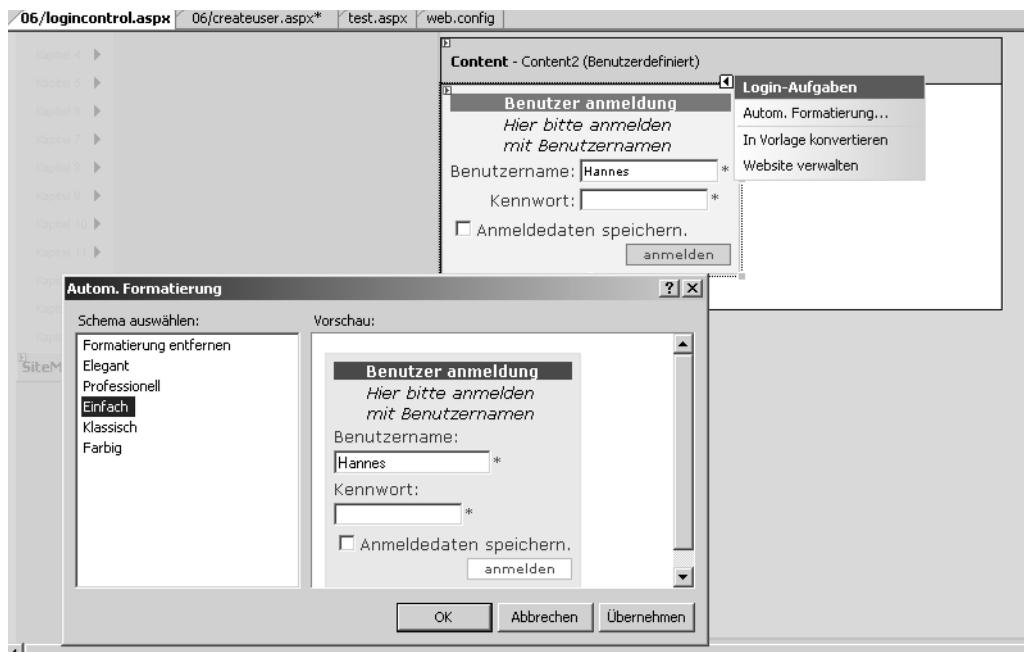
```

MembershipProvider="AspNetSqlMembershipProvider"
UserName="Hannes"
DestinationPageUrl="-/06/loginnameSteuerelement.aspx"
BorderPadding="4" ForeColor="#333333">
<TitleTextStyle Font-Bold="True" BackColor="#507CD1" ForeColor="White"
    Font-Size="0.9em"></TitleTextStyle>
<LoginButtonStyle BackColor="Pink" BorderColor="#507CD1" BorderStyle="Solid" BorderWidth="1px"
    Font-Names="Verdana" Font-Size="0.8em" ForeColor="#284E98"/>
<TextBoxStyle Font-Size="0.8em" />
<InstructionTextStyle Font-Italic="True" ForeColor="Black" />
</asp:Login>

```

**Listing 6.25** Login-User-Steuerelement im HTML View (Fortsetzung)

Sehr schön ist die Vorschau in der Entwicklungsumgebung von Visual Web Developer 2005 auf das Webserver-Steuerelement. Änderungen bezüglich Farbe und Text sind sofort zu sehen.



**Abbildung 6.14** Login-Steuerelement

Weitere Möglichkeiten, auf Gestaltung und Design des Login-Formulars einzuwirken, bieten sich mit den Unterelementen des Login-Steuerelements.

Element	Verwendung
<i>CheckBoxStyle</i>	Der Stil für die Checkbox.
<i>FailureTextStyle</i>	Der Stil für Fehlermeldungen.
<i>HyperLinkStyle</i>	Der Stil für Hyperlinks.

**Tabelle 6.7** Elemente des Login-Webserver-Steuerelements

Element	Verwendung
<i>InstructionTextStyle</i>	Der Stil für den Anleitungstext.
<i>LabelStyle</i>	Der Stil für die Labels.
<i>LayoutTemplate</i>	Mit dem Layout-Template wird der Inhalt manuell zusammengestellt.
<i>LoginButtonStyle</i>	Der Stil des Buttons zum Absenden des Formulars.
<i>TextBoxStyle</i>	Der Stil der Textboxen.
<i>TitleTextStyle</i>	Der Stil des Titels.
<i>ValidatorTextStyle</i>	Der Stil der Eingabeprüfungs-Steuerelemente.

**Tabelle 6.7** Elemente des Login-Webserver-Steuerelements (*Fortsetzung*)

Wenn man das Steuerelement in ein Template konvertiert, kann man auf jedes Detail einzeln ohne diese Elemente Einfluss nehmen, egal ob es sich um Beschriftungen von Labels, Buttons oder die Platzierung derselben handelt.

## LoginView-Steuerelement

Das *LoginView*-Steuerelement erlaubt es, die Anzeige im Browser abhängig vom angemeldeten Benutzer zu gestalten. Dazu gibt es unterschiedliche Vorlagen. Im *AnonymousTemplate* werden die Web-Steuerelemente und HTML-Elemente platziert, die angezeigt werden, wenn keine Benutzerauthentifizierung vorliegt.

Wenn der Benutzer angemeldet ist, wird der Inhalt des *LoggedInTemplate*-Bereichs angezeigt.

Die dritte Art von Template definiert mit der Gruppe *rolegroups* die Anzeige je nach Rollenzugehörigkeit des Benutzers. Dabei wird, wenn der Benutzer keine oder keine passende Rolle besitzt, das *LoggedInTemplate* verwendet. Ist der Benutzer Mitglied einer Rolle, wird die zuerst passende *RoleGroup* zur Steuerung der Anzeige verwendet. Nur Admins sehen den Inhalt der Rollengruppe Admin.

Es wird immer nur ein Template zur Anzeige verwendet.

```
<asp:LoginView ID="LoginView1" Runat="server">
<RoleGroups>
  <asp:RoleGroup Roles="Admin"><ContentTemplate>das sieht nur der Admin</ContentTemplate>
  </asp:RoleGroup>
</RoleGroups>
<LoggedInTemplate>
  Sie sind angemeldet
</LoggedInTemplate>
<AnonymousTemplate>
  Sie sind nicht angemeldet<br />
  können das aber <a href="loginSteuerelement.aspx"> hier</a> tun
</AnonymousTemplate>
</asp:LoginView>
```

**Listing 6.26** LoginView mit Vorlagen

Die Ausgabe im Browser für einen nicht angemeldeten Benutzer gibt diesem im Beispiel dann die Option zur Benutzeranmeldung.



Abbildung 6.15 Anzeige des Anonymous Templates

## PasswordRecovery-Steuerelement

Früher war vieles einfacher – man musste nur seinen Netzwerk Benutzernamen kennen. Heute ist das anders. Jede Webanwendung fordert ihre eigene Anmeldung. Der Ansatz von Microsoft, mit Passport das Problem zu beheben, kann aufgrund der geringen Benutzerakzeptanz als gescheitert betrachtet werden.

In der Praxis vergessen die Benutzer regelmäßig ihre Benutzerdaten. Um zu verhindern, dass bei jedem Besuch ein neuer Account angelegt wird, muss ein möglichst komfortabler Mechanismus bereit stehen, um vergessene Passwörter wiederzuerlangen. Dabei gibt es zwei Grundansätze: Ein vergessenes Passwort wird einfach automatisch auf Anforderung an die hinterlegte E-Mail-Adresse gesendet. Das ist zwar relativ komfortabel, aber auch unsicher. Die Passwörter müssen in der Datenbank hinterlegt sein. Dadurch erfolgt die Kommunikation zwangsweise im Klartext, und sie lässt sich relativ einfach abhören. Ein weiteres Problem ergibt sich, wenn der Benutzer nun eine neue und damit andere E-Mail-Adresse hat, weil er z.B. seinen Arbeitgeber gewechselt hat. Die Alternative ist da nur die Passwörter eben nicht zu speichern.

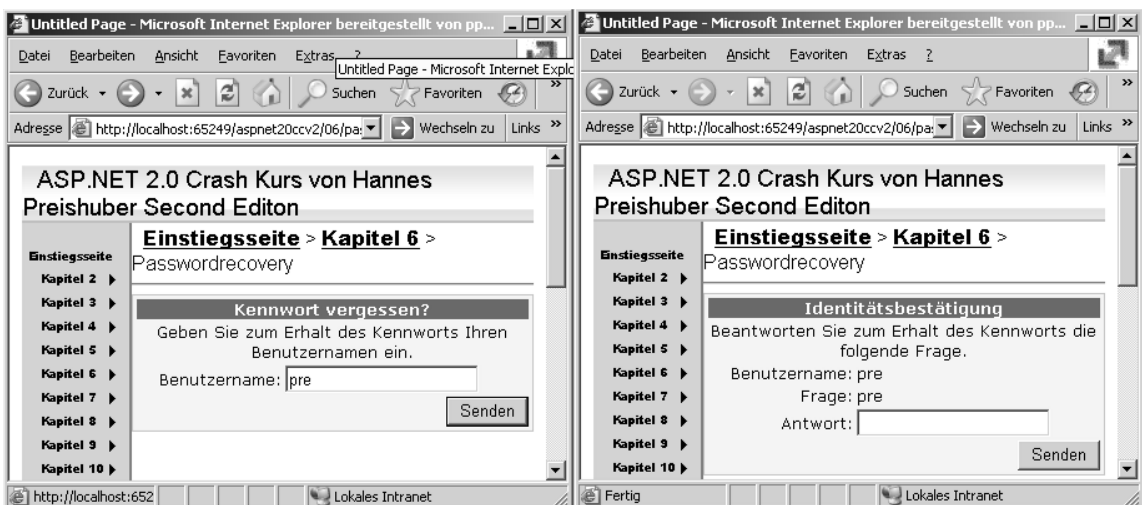


Abbildung 6.16 Zweistufiges PasswordRecovery

Das *PasswordRecovery*-Steuerelement bietet einen zweistufigen Weg an. In der ersten Stufe wird der Benutzername abgefragt. Befindet sich dieser in der Datenbank, wird dazu die geheime Frage angezeigt, die man beantworten muss. Erst wenn die Antwort identisch mit der Antwort in der Datenbank ist, wird das Passwort per E-Mail zugesandt.

Obwohl diese Vorgehensweise auf den ersten Blick sehr einfach aussieht, gibt es einiges zu beachten bzw. zu konfigurieren.

```
<asp:PasswordRecovery ID="PasswordRecovery1" Runat="server"
    SuccessText="Ihr Passwort">
</asp:PasswordRecovery>
```

**Listing 6.27** Das PasswordRecovery-Steuerelement

In der *machine.config* ist standardmäßig das Empfangen von Passwörtern unterbunden. So müssen also in der Definition des Providers noch einige Einstellungen vorgenommen werden. Wenn *enablePasswordRetrieval* auf *false* gesetzt ist, wird ein neues Passwort erzeugt. Dieses Attribut hängt mit dem *passwordFormat* zusammen (*Clear*, *Encrypted*, *Hashed*). Ein gehashtes und damit verschlüsseltes Passwort kann nicht angefordert werden. Dazu müsste für Format *Clear* eingestellt sein. Damit werden die Passwörter im Klartext in der Datenbank gespeichert. Mit *enablePasswordReset* wird definiert, ob das Passwort überhaupt zurückgesetzt werden darf. Das Attribut *requiresQuestionAndAnswer* definiert, ob die »Passwort-Vergessen-Frage« angezeigt wird.

```
<providers >
<remove name="AspNetSqlMembershipProvider"/>
<add name="AspNetSqlMembershipProvider"
type="System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"
    connectionStringName="LocalSqlServer"
    enablePasswordRetrieval="false"
    enablePasswordReset="true"
    requiresQuestionAndAnswer="true"
    applicationName="/"
    requiresUniqueEmail="false"
    passwordFormat="Hashed"
    description="speichert und liest Benutzer Accounts aus dem lokalen SQL Server" />
</providers>
</membership>
```

**Listing 6.28** Membership-Bereich aus *web.config*

Darüber hinaus setzt das Steuerelement noch zwingend einen installierten SMTP-Server voraus. Achten Sie darauf, dass auch das Weiterleiten von E-Mails erlaubt sein muss. In Kapitel 12 können Sie Details über die SMTP-Konfiguration nachlesen.

Weiterhin muss im Steuerelement das *MailDefinition*-Unterelement vorhanden sein. Dort werden über Attribute die Sender-Informationen eingestellt. Da die Attributnamen selbsterklärend sind, wird hier auf eine Detailbeschreibung verzichtet.



```
<MailDefinition
  IsBodyHtml="false"
  From="Passwordservice@devtrain.de"
  Priority="High"
  Subject="Ihr angefordertes Passwort">
</MailDefinition>
```

Bevor Sie ans Versenden denken, muss noch der Mail-Server konfiguriert werden. Das geschieht wie immer in der *web.config*, allerdings im Bereich *System.Net*. Eine genauere Beschreibung des E-Mail-Versands in ASP.NET 2.0 erfolgt in Kapitel 12.

```
<mailSettings >
<smtp deliveryMethod="PickupDirectoryFromIis">
<network host="localhost" port="25"/>
</smtp>
</mailSettings>
```

**Listing 6.29** Konfiguration des Sende-Servers in der *web.config*

Der Bodytext der E-Mail wird automatisch erzeugt. Sie können im Verzeichnis des SMTP Servers *C:\inetpub\mailroot\Queue* E-Mails direkt per Doppelklick öffnen, solange diese noch nicht versendet worden sind. Die E-Mail wird dann mit Outlook Express geöffnet und angezeigt.



**Abbildung 6.17** Eine Beispiel-E-Mail mit Passwörterinnerung

Natürlich kann auch der Body-Text geändert werden. Dazu gibt es das Attribut *BodyFileName*, dem als Parameter ein Dateiname übergeben wird. Ist kein Pfad angegeben, wird der aktuelle Pfad, in dem sich die ASPX-Datei befindet, verwendet.

```
<MailDefinition BodyFileName="mail.txt"..
```

In der Textdatei können Sie sich dann kreativ betätigen. Als Platzhalter für Benutzername und Passwort werden `<%UserName%>` und `<%Password%>` verwendet.



Abbildung 6.18 Das Passwort wurde versandt

Es kann auch über die Ereignismethoden auf die möglichen Ereignisfälle reagiert werden. Beispielhaft wird hier *SendMailError* genannt. Damit wird keine Fehlermeldung im Browser angezeigt, sondern durch den Code in dieser Prozedur behandelt.

## LoginStatus-Steuerelement

Das LoginStatus-Steuerelement hat eine ziemlich begrenzte Aufgabe. Es soll nichts anderes tun, als den Login-Status anzuzeigen.

Die Standardtexte können lokalisiert werden mit den Attributen *LoginText* und *LogoutText*.

```
<asp:LoginStatus ID="LoginStatus1" Runat="server" LoginText="Bitte anmelden" LogoutText="Sie sind abgemeldet"/>
```

Listing 6.30 Das LoginStatus-Steuerelement

Alternativ zu den Texten lassen sich mit den Attributen *LoginImageUrl* und *LogOutImageUrl* auch Bilder anzeigen. Für den ersten Versuch mit obigem Code wird auf Bilder verzichtet. In der Entwicklungsumgebung kann zwischen den beiden Vorlagen umgeschaltet werden.

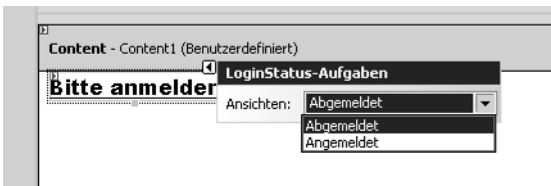


Abbildung 6.19 Login-Status mit Hyperlink zum Login

Die Seite, die nach der Abmeldung angezeigt werden soll, wird mit dem Attribut *LogoutUrl* definiert. Der Link zur Anmeldung zeigt statisch auf die *Login.aspx* des aktuellen Verzeichnisses. Im Browser wird folgender HTML-Code erzeugt.

```
<a id="ctl100_ContentPlaceHolder1_LoginStatus1" href="/ASPNET20CC/04/login.aspx?ReturnUrl=%2fASPNET20CC%2f06%2floginStatusSteuerelement.aspx">Bitte anmelden</a>
```

Ein besonders hinterhältiger Trick das zu umgehen, ist, einfach HTML-Code in den Text zu schleusen.

```
LoginText="<a href=index.aspx>Bitte anmelden</a>"
```

Im Folgenden sind die Attribute des LoginStatus-Steuerelements beschrieben, die die Logout-Aktion beeinflussen.

Attribut	Verwendung
<i>LogoutAction</i>	Damit wird die Aktion festgelegt, die beim Drücken des Logout-Buttons erfolgen soll. Mit <i>Refresh</i> wird die gleiche Seite nochmals aufgerufen. Mit <i>RedirectToLoginPage</i> wird auf die in der <i>web.config</i> festgelegte Login Page umgeleitet. Um eine definierte andere Seite aufzurufen, wird das Attribut <i>Redirect</i> verwendet.
<i>LogoutPageUrl</i>	Dieses Attribut wird im Zusammenhang mit <i>LogoutAction=Redirect</i> verwendet und definiert die Zielseite.

**Tabelle 6.8** Attribute des LoginStatus-Steuerelements

## LoginName-Steuerelement

Das einfachste Steuerelement ist wohl das *LoginName*-Steuerelement. Damit wird der aktuelle Benutzername angezeigt. Wenn keine Anmeldung erfolgt ist, wird nichts ausgegeben.

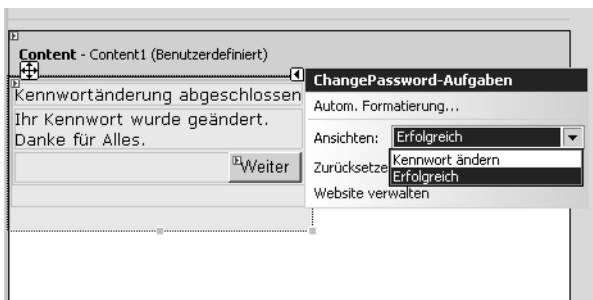
```
<asp:LoginName ID="LoginName1" Runat="server" />
```

Sinnvolles Einsatzgebiet ist z.B. in Kombination mit Masterseiten die dauerhafte Anzeige des aktuellen Benutzer-Accounts. Mit dem zusätzlichen Attribut *FormatString* lässt sich auch noch die Anzeige verschönern.

```
<asp:LoginName id="LoginName1" runat="server" FormatString="Willkommen, {0}" />
```

## ChangePassword-Steuerelement

Das letzte der Login-Steuerelemente übernimmt den Job der Passwortänderung. Es gibt zwei Anzeigevorlagen (*Erfolgreich* und *Kennwort ändern*). Über das Aufgabenmenü kann zwischen diesen gewechselt werden und bei Bedarf eine Vorlage *<SuccessTemplate>* daraus erzeugt werden.



**Abbildung 6.20** Das Success Template wird bearbeitet

Zur Formatierung der Templates werden HTML-Tabellen eingesetzt. Die IDs der *TextBox*-Steuerelemente und Buttons dürfen nicht geändert werden, um die Funktion nicht zu beeinträchtigen. Dafür können Sie hinsichtlich des Designs Ihrer Fantasie freien Lauf lassen.

```
<asp:ChangePassword ID="ChangePassword1" Runat="server">
<ChangePasswordTemplate>
<table cellpadding="1" border="0"><tr><td>
  <table cellpadding="0" border="0"><tr>
    <td align="center" colspan="2">
      <tr><td align="right">
        <asp:Label Runat="server" AssociatedControlID="CurrentPassword"
          ID="CurrentPasswordLabel">Passwort:</asp:Label></td>
        <td><asp:TextBox Runat="server" TextMode="Password" ID="CurrentPassword"></asp:TextBox>
        <asp:RequiredFieldValidator Runat="server" SteuerelementToValidate="CurrentPassword"
          ValidationGroup="ChangePassword1"
          ErrorMessage="Passwort erforderlich." ToolTip="Passwort ist erforderlich."
          ID="CurrentPasswordRequired">*</asp:RequiredFieldValidator>
        </td></tr>
      <tr><td align="right">
        <asp:Label Runat="server"
          AssociatedControlID="NewPassword" ID="NewPasswordLabel">Neues Passwort:</asp:Label></td>
        <td><asp:TextBox Runat="server" TextMode="Password" ID="NewPassword"></asp:TextBox>
        <asp:RequiredFieldValidator Runat="server"
          SteuerelementToValidate="NewPassword" ValidationGroup="ChangePassword1"
          ErrorMessage="Passwort ist erforderlich." ToolTip="Passwort ist erforderlich."
          ID="NewPasswordRequired">*</asp:RequiredFieldValidator>
        </td></tr>
      <tr><td align="right">
        <asp:Label Runat="server" AssociatedControlID="ConfirmNewPassword"
          ID="ConfirmNewPasswordLabel">nochmal neues Passwort:</asp:Label></td>
        <td><asp:TextBox Runat="server" TextMode="Password" ID="ConfirmNewPassword"></asp:TextBox>
        <asp:RequiredFieldValidator Runat="server" SteuerelementToValidate="ConfirmNewPassword"
          ValidationGroup="ChangePassword1" ErrorMessage="Passwort wiederholen."
          ToolTip="Passwort wiederholen zur Bestätigung."
          ID="ConfirmNewPasswordRequired">*</asp:RequiredFieldValidator>
        </td></tr>
      <tr><td align="center" colspan="2">
        <asp:CompareValidator Runat="server" SteuerelementToValidate="ConfirmNewPassword"
          ValidationGroup="ChangePassword1"
          ID="NewPasswordCompare" Display="Dynamic"
          ErrorMessage="Die Eingaben für das Neue Passwort sind nicht identisch."
          SteuerelementToCompare="NewPassword">
        </asp:CompareValidator>
        </td></tr>
      <tr><td align="right">
        <asp:Button Runat="server" Text="Passwort speichern" ValidationGroup="ChangePassword1"
          CommandName="ChangePassword" ID="ChangePasswordPushButton" />
        </td><td align="right"><asp:Button Runat="server" CausesValidation="False" Text="Abbruch"
          CommandName="Cancel" ID="CancelPushButton"/>
        </td></tr>
      <tr><td colspan="2" style="color: red;">
        <asp:Literal Runat="server" ID="FailureText" EnableViewState="False"></asp:Literal>
        </td></tr></table>
    </td>
  </tr></table>
</ChangePasswordTemplate>
</SuccessTemplate>
```

**Listing 6.31** ChangePassword mit Templates

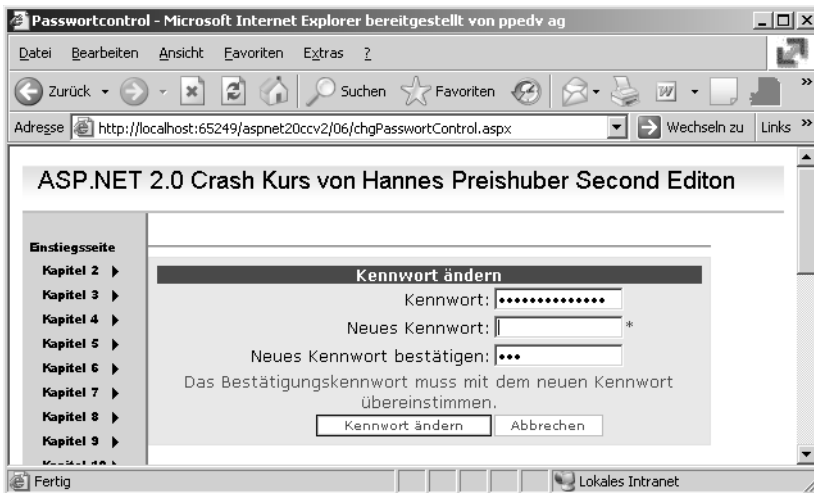
```

<table cellpadding="1" border="0"><tr><td>
  <table cellpadding="0" border="0">
    <tr><td align="center" colspan="2">
      Passwort erfolgreich geändert</td>
    </tr><tr><td colspan="2">Passwort geändert!</td>
    </tr><tr><td align="right" colspan="2">
      <asp:Button Runat="server" CausesValidation="False" Text="Continue" CommandName="Continue"
      ID="ContinuePushButton" />
    </td></tr></table>
  </td></tr></table>
</SuccessTemplate>
</asp:ChangePassword>

```

**Listing 6.31** ChangePassword mit Templates (Fortsetzung)

ASP.NET erzeugt dann aus dem Steuerelement eine HTML-Seite mit allen möglichen Feinheiten. Selbst die Validierung der Eingabe ist bereits enthalten. Es wird nun Zeit das unschöne Passwort, das von ASP.NET automatisch versandt wurde, aus Bild 6.17 wieder zu ändern.



**Abbildung 6.21** Die Passwortänderung ist fehlgeschlagen

## Personalisierung mit Profilen

Viele Funktionen von ASP.NET 2.0 erlauben es, Websites zu personalisieren. Was in ASP.NET 1.x noch fehlt, ist eine einfache typisierte Speicherung von Benutzerdaten. Das ist zwar bisher auch schon möglich, aber mit erheblichem Aufwand verbunden. Nehmen wir als Beispiel den Inhalt eines Warenkorbes. Während des gesamten Shopping-Vorgangs muss jederzeit auf den Inhalt des Warenkorbes zugegriffen werden können. Am besten sollen die Daten dafür auch im Arbeitsspeicher gehalten werden, um die Datenbank nicht zu belasten. Bisher würde man dies wahrscheinlich mit einem Warenkorb-Objekt in einer Session-Variablen realisieren. Die Werte sind aber nicht typisiert und müssen erst immer in den richtigen Typ umgewandelt werden, bevor das Objekt verwendet werden kann.

ASP.NET 2.0 bringt nun mit Benutzerprofilen die dafür genau passende Lösung. Dabei integrieren Profile sich in das Membership System. Es wird so ein personalisiertes Profil mit typisierten Eigenschaften erstellt. Diese Daten werden dauerhaft in der Membership Datenbank gespeichert. Für nicht authentifizierte Benutzer werden auch anonyme Profile erstellt. So kann man einen Warenkorb füllen ohne angemeldet zu sein. Je nach Einstellung wird dann mithilfe eines Cookies die Zuordnung zum Profil erstellt, so dass auch mit dem Schließen des Browsers keine Daten verloren gehen.

Die Struktur des Profils ist völlig flexibel. Das Schema wird in der *web.config* deklariert. IntelliSense stellt die passenden Eigenschaften in der Entwicklungsumgebung zur Verfügung. Zusammengehörnde Bereiche können sogar gruppiert werden.

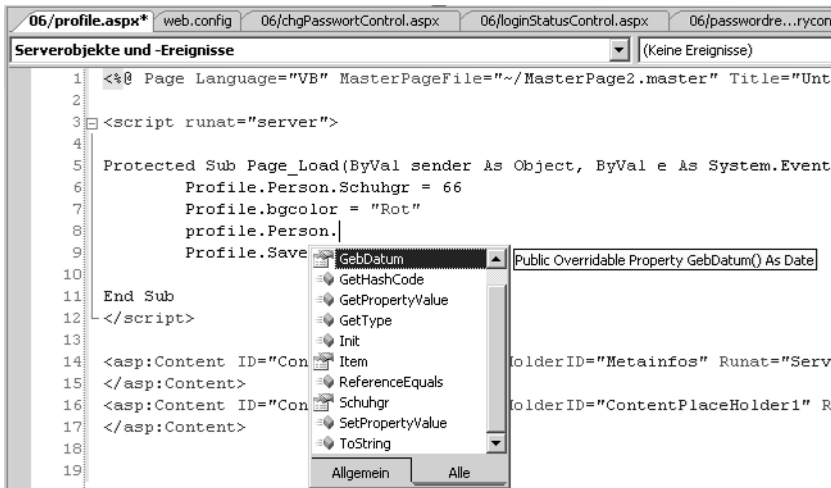


Abbildung 6.22 IntelliSense kennt alle Eigenschaften und Typ des Profils

Die Änderungen der Profildaten können explizit per *Save*-Kommando gespeichert werden. Die Eigenschaft *IsDirty* gibt an, ob die Profildaten gespeichert sind. So kann ein überflüssiger *Save*-Befehl verhindert werden. Der SQL Profile Provider sichert ganz automatisch die Profilinformatoren, spätestens dann, wenn die Seite abgearbeitet ist. Die kann mit dem Attribut *AutomaticSaveEnabled* allerdings auch im Profile-Element der *web.config* deaktiviert werden

Wo und wie die Profildaten letztendlich abgelegt werden, ist Sache des Profil-Providers und seiner Konfiguration.

## Konfiguration

Das Erstellen von der Profilstruktur wird durch Einträge in der *web.config* vorgenommen. Dies kann man natürlich auch direkt in der Entwicklungsumgebung oder mit einem Texteditor realisieren. Es wird dabei der Bereich *Profile* bearbeitet. Im Element *properties* werden dann die Eigenschaften samt Datentypen erstellt. Wenn auch anonyme Benutzer dieses Attribut verwenden können sollen, muss das Attribut *allowAnonymous* gesetzt werden. Mit dem *Group*-Element können Attribute gruppiert werden. Im folgenden Beispiel wird gestattet, die Hintergrundfarbe (*bgcolor*) einer Webseite durch den Benutzer zu setzen. Eine Profil-Variable *bgColor* speichert diesen Wert. Dazu wird der Standardwert über *defaultValue* vorbelegt und die Profileigenschaft auch nicht authentifizierte Benutzern zur Verfügung gestellt.

```
<profile defaultProvider="AspNetSqlMembershipProvider" enabled="true" >
  <properties>
    <add name="Name" type="System.String" allowAnonymous="true"/>
    <add name="bgcolor" type="System.String" defaultValue="white" allowAnonymous="true"/>
    <group name="Person" >
      <add name="Schuhgr" type="System.Int32" allowAnonymous="true"/>
      <add name="GebDatum" type="System.DateTime" allowAnonymous="true" />
    </group>
  </properties>
</profile>
```

**Listing 6.32** Profildefinition in *web.config*

## Anonyme Personalisierung

Diese Überschrift klingt wie ein Gegensatz in sich, macht aber durchaus Sinn. Bevor man als nicht authentifizierter Benutzer von den Vorteilen des Profilmanagements profitieren kann, muss erst in der *web.config* die Möglichkeit dazu geschaffen werden. Standardmäßig ist das nicht möglich. Es müssen das Element *anonymousIdentification* vorhanden und das Attribut *enabled* auf *true* gesetzt sein.

```
<anonymousIdentification
  enabled="true"
  cookieName=".ASPXANONYMOUS"
  cookieTimeout="43200"
  cookiePath="/"
  cookieRequireSSL="false"
  cookieSlidingExpiration="true"
  cookieProtection="All"
  cookieless="UseDeviceProfile"/>
```

**Listing 6.33** Ausschnitt aus *web.config*

Mit den weiteren Attributen lässt sich das Verhalten des Cookies steuern, der verwendet wird, um die Profilzuordnung zu steuern.

Außerdem muss in den definierten Profilen für jede Eigenschaft, die auch anonym verfügbar sein soll, das Attribut *allowAnonymous="true"* gesetzt sein.

## Profiles

Mit dem Profile-Manager und der Klasse *Profile* kann ohne Instanziierung gearbeitet werden. Die per *web.config* definierten Attribute stehen auch direkt zur Verfügung, egal ob diese gelesen oder geschrieben werden sollen.

```
Profile.Name = Name.Text
```

Im folgenden Beispiel soll neben einigen anderen Parametern insbesondere die Hintergrundfarbe im Profil abgeleitet werden. Die Eigenschaft dazu wurde *bgcolor* genannt, wie man aus Listing 6.33 erkennen kann.

```

Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Page.Form.Style.Add(HtmlTextWriterStyle.BackgroundColor, Profile.bgcolor)
End Sub
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Profile.bgcolor = drpBackColor.SelectedColor
    Profile.Name = Name.Text
    Profile.Person.GebDatum = CDate(txtGebDat.Text)
    Profile.Person.Schuhgr = CInt(txtSchuhgr.Text)
    Profile.Save()
End Sub

```

**Listing 6.34** Hintergrundfarbe mit Profile-Info setzen

Das Setzen der Hintergrundfarbe funktioniert im Browser auch ohne Benutzeranmeldung.



**Abbildung 6.23** Profil bearbeiten

## ProfileManager

Natürlich sammelt sich im Laufe der Zeit durch Profile einiges an Daten in der Datenbank an. Profile sollte man demzufolge auch wieder löschen können. Der *ProfileManager* hilft beim Verwalten der gespeicherten Profile. Mit der Funktion *GetAllProfile* werden alle gespeicherten Profile ausgelesen. Mit dem Parameter kann man zwischen *All*, *Anonymous* und *Authenticated* wählen. Wie fast immer lässt sich die Rückgabe direkt an ein *GridView*-Steuerelement binden, sodass man ohne großen Aufwand eine Listendarstellung realisieren kann.

```

<%@ Page Language="VB" MasterPageFile="-/all.master" Title="Untitled Page" %>
<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        GridView1.DataSource = ProfileManager.GetAllProfiles(ProfileAuthenticationOption.All)
        GridView1.DataBind()
    End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">&nbsp;&nbsp;<asp:GridView

```

**Listing 6.35** Alle gespeicherten Profile anzeigen



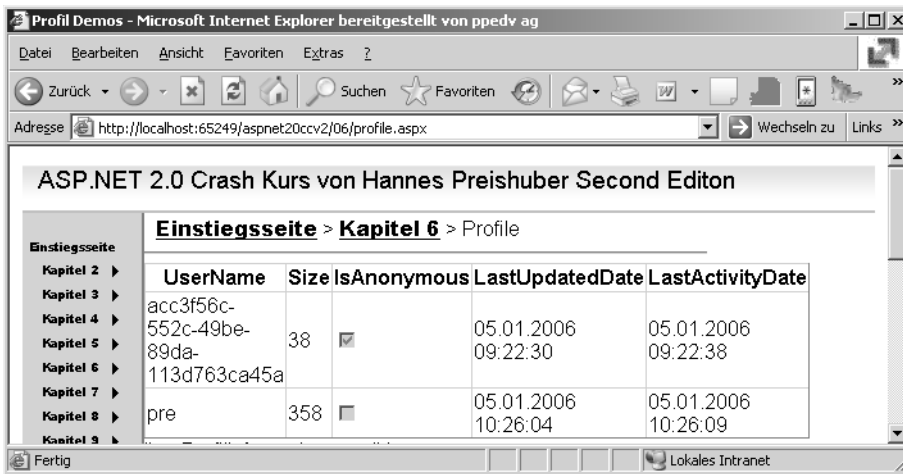
```
ID="GridView1" Runat="server">
</asp:GridView>
</asp:Content>
```

**Listing 6.35** Alle gespeicherten Profile anzeigen (Fortsetzung)

Wenn nun ein Profil gelöscht werden soll, kann dies ebenfalls mit dem ProfileManager erfolgen:

```
ProfileManager.DeleteProfile(User.Identity.Name)
```

Das *GridView*-Steuerelement zeigt automatisch alle vorhandenen Felder an.



**Abbildung 6.24** Liste der Profile wird angezeigt

Wenn ein Benutzer nicht authentifiziert ist, handelt es sich um ein anonymes Profil. In einem Online-Shop kann dies z.B. für das Füllen des Warenkorbs verwendet werden. Wenn der Benutzer zum Bezahlen geht, muss die Information trotzdem erhalten bleiben. Dies geschieht automatisch, wenn sich der Benutzer anmeldet. Dabei wird das Ereignis *MigrateAnonymous* ausgelöst, in dem man noch Änderungen an den Profildaten vornehmen kann.

```
Sub Profile MigrateAnonymous(ByVal sender As Object, ByVal e As ProfileMigrateEventArgs)
Dim anoProfile As HttpProfile = Profile.GetProfile(e.AnonymousId)
ProfileManager.DeleteProfile(e.AnonymousId)
AnonymousIdentificationModule.ClearAnonymousIdentifier()
End Sub
```

**Listing 6.36** Anonymes Profil wird persönlich

In diesem Beispiel wurde die Profildatenbank aufgeräumt, indem die Daten zu einem Profil physikalisch gelöscht wurden.



## Kapitel 7

# Datenzugriff

### **In diesem Kapitel:**

Crashkurs Datenzugriff	180
Datenbindung ohne Code	188
Datenquellen-Steuer-elemente	195
Neues in ADO.NET 2.0	209

Bei der Softwareentwicklung dreht sich fast alles um Daten. Diese müssen erfasst, geprüft und wieder dargestellt werden. So gut wie jede Anwendung braucht Daten. Der Großteil der benötigten Codezeilen entfällt dabei auf diese Aufgabe.

Insofern ist es nur logisch, genau in diesem Bereich zu versuchen, die benötigte Menge an Code zu reduzieren, um eine Datenanwendung zu realisieren.

Genau das versucht die Technologie ADO.NET, nunmehr schon in ihrer zweiten Version. Mit ASP.NET Webserver-Steuerelementen und ADO.NET ist es jetzt möglich viele Datenbankaufgaben gänzlich ohne Programmcode zu erledigen.

ADO.NET ist Datenzugriffsschicht und Code Bibliothek in einem. Durch das Provider-Konzept kann damit jede denkbare Datenquelle angesprochen werden. Dabei ist es nebensächlich, ob es sich um eine relationale oder hierarchische Quelle handelt. Ein großes Problem vieler Entwickler ist, dass ADO.NET ein entkoppeltes System ist. Die Konsequenz daraus ist, dass keine Cursor oder Satzsperrern zur Verfügung stehen.

Die Daten werden von der Datenbank in ein Container-Objekt geladen, das man *DataSet* nennt. Die Verbindung zur Datenbank wird daraufhin wieder getrennt. Das *DataSet* ist dann eine Kopie der Daten und wird oft auch »In-Memory-Datenbank« genannt. Dazu gehört ebenfalls, dass mehr als eine Tabelle, Relationen und andere datenbanktypischen Eigenschaften im *DataSet* gespeichert werden können. Der Nachteil ist, dass die Daten sehr viel Platz im Arbeitsspeicher benötigen und man erst mit den Daten arbeiten kann, wenn diese komplett geladen sind.

Wenn größere Datenmengen benötigt werden, kommt deshalb besser der *DataReader* zum Einsatz. Dieser liest die Daten Satz für Satz in einer Vorwärtsbewegung ein.

## Crashkurs Datenzugriff

Erst im Zusammenspiel mit Visual Studio 2005 macht Datenzugriff unbegrenzt Spaß. Je nach verwendeter Version fehlen aber unter Umständen einige der hier beschriebenen Masken und Features. Speziell die Edition Visual Web Developer 2005 weist starke Einschränkungen auf.

## Einfaches datengebundenes Formular

Die einfachste Methode erlaubt es, binnen Sekunden eine tabellarische Darstellung von Daten in einer Webseite zu realisieren. Das setzt voraus, dass im *Datenbank Explorer* (Server Explorer bei Visual Studio 2005) unter *Datenverbindungen* bereits eine Verbindung existiert. Sie können dort natürlich auch eine neue Verbindung zu einer Datenbank anlegen, dann dauert es ein paar Sekunden länger.

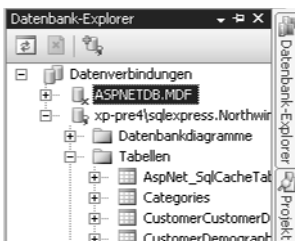


Abbildung 7.1 Datenbank-Explorer von Visual Web Developer Express

Dazu ziehen Sie einfach Ihre gewünschte Tabelle, im folgenden Beispiel Customer, auf das Web-Formular. Wenn Sie nicht alle Felder benötigen, können Sie auch die entsprechenden Felder markieren und auf das Formular ziehen.

Dadurch werden zwei neue Webserver-Steuerelemente erzeugt. Zum Anzeigen der Daten als Tabelle wird das *GridView*-Steuerelement als Nachfolger des *DataGrid*-Steuerelements verwendet. Das zweite Steuerelement ist *SqlDataSource*. Die Gruppe der DataSource-Steuerelemente sind Teil der neuen Datensteuerelemente, die »Code Free Development« ermöglichen. Außerdem sind diese Steuerelemente in der Entwicklungsumgebung zur Entwurfszeit sichtbar und besitzen ein *Kontext-Aufgabenmenü*.

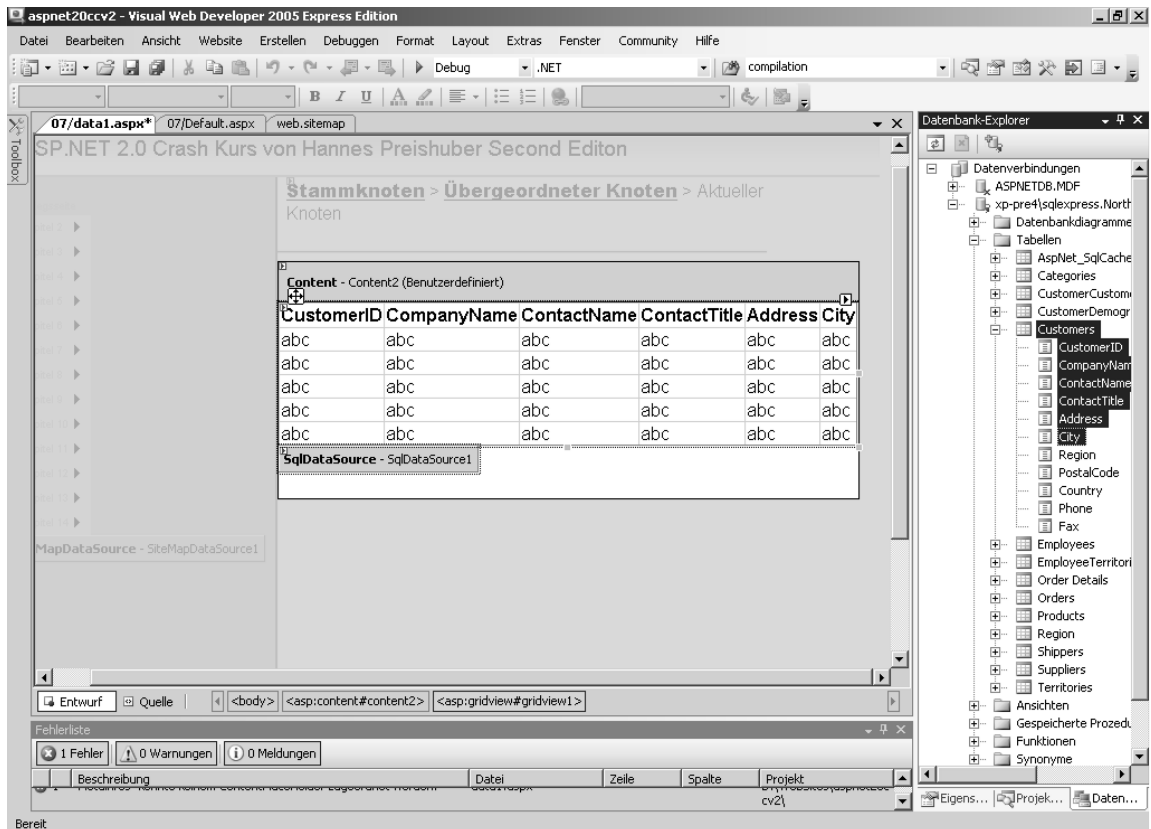


Abbildung 7.2 Per Drag & Drop erzeugte Datensicht

Die so erzeugte ASPX-Seite ist schon voll funktionsfähig und kann im Browser getestet werden.

In der ASPX-Seite sind die beiden Steuerelemente *GridView* und *SqlDataSource* vorhanden. Mit umfangreichen Attributen werden alle Datenzugriffparameter definiert. Es lassen sich die beiden Steuerelemente auch direkt aus der Werkzeugleiste auf die Seite ziehen und manuell entsprechend konfigurieren. Dabei hilft das *Aufgabenmenü*. Das *SqlDataSource*-Objekt kümmert sich zur Laufzeit um die Daten und das *GridView* um die Darstellung derselben.

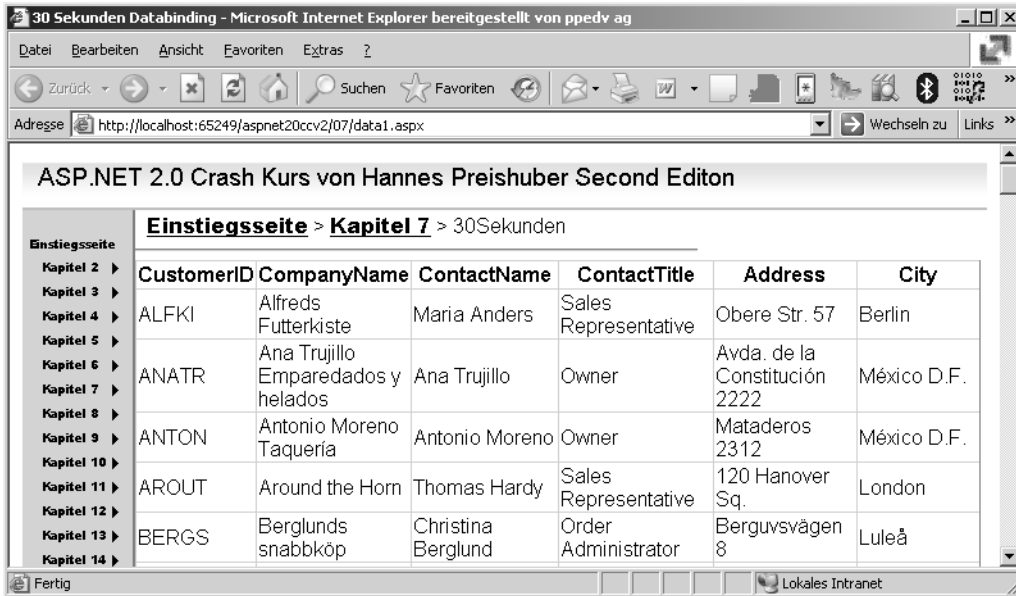


Abbildung 7.3 Datenbindung in 30 Sekunden

Zuallerst muss das `DataSource`-Steuerelement eine Verbindung zur Datenbank aufbauen. Dies wird in der *Verbindungszeichenfolge*, dem `ConnectionString` definiert. Dieser kann direkt in den Quellcode der ASPX-Seite geschrieben oder besser in die `web.config` ausgelagert werden. Damit ist der `ConnectionString` nur einmal pro Anwendung definiert. Die Einbindung in das `SQLDataSource`-Steuerelement funktioniert dank dem neuen ASP.NET Feature *ASP.NET Expressions*, das in Kapitel 12 detailliert beschrieben wird.

Ein weiteres wichtiges Attribut ist das `SelectCommand`, das bestimmt welche Daten geholt werden sollen. Hier gibt es weitere `Command`-Attribute um `Update`-, `Insert`- und `Delete`-Kommandos auszuführen.

Dem `GridView`-Steuerelement wird dann das `DataSource`-Steuerelement mittels des Attributs `DataSourceID` zugewiesen.

```
<asp:GridView ID="GridView1" Runat="server" DataKeyNames="ProductID" DataSourceID="SqlDataSource1">
</asp:GridView>
<asp:SqlDataSource ID="SqlDataSource1" Runat="server" SelectCommand="SELECT [ProductID], [ProductName],
[SupplierID] FROM [Alphabetical list of products]"
ConnectionString="<%= $ ConnectionStrings:AppConnectionString1 %>">
</asp:SqlDataSource>
```

Listing 7.1 Datenbindung per Deklaration in der ASPX-Seite

Eine genauere Beschreibung der visuellen Daten-Steuerelemente folgt in Kapitel 8. Trotzdem kommen hier diese Steuerelemente zur Anwendung, um andere Funktionen von ADO.NET zu beschreiben.

## Einfaches Datenzugriffobjekt

Auch das Erstellen einer Datenzugriffsschicht (Data Access Layer=DAL) kann ähnlich schnell vonstatten gehen. Dabei hilft ein Assistent. An der Vorgehensweise des Assistenten erkennt man sehr gut Details der

hinter ADO.NET liegenden Konzepte. Wenn man den Dialog *Neues Element hinzufügen* aufruft, und den *DataSet-Assistenten* auswählt, könnte man vermuten, dass ein datengebundenes Formular erzeugt wird. Dies ist aber nicht so, es wird eine Art Daten-Zugriffs-Objekt erzeugt. Im nächsten Schritt versucht der Assistent, Sie davon zu überzeugen, Ihre Komponente im *app\_code*-Verzeichnis zu speichern. Das erlaubt den Zugriff von allen Seiten auf die Komponente. Bestätigen Sie also folgenden Dialog mit *Ja*.

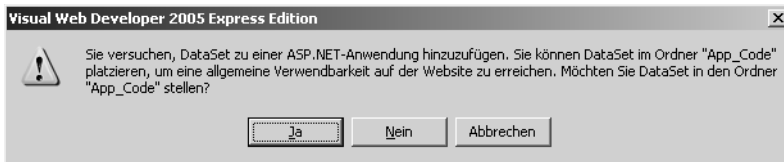


Abbildung 7.4 Ein DataSet wird erzeugt

Wer erwartet, dass nun umfangreicher VB- oder CS-Code erzeugt wird, wird enttäuscht. Es wird einzig eine Datei zur Definition des Daten-Schemas mit der Erweiterung XSD erzeugt. Durch das Speichern im *app\_code*-Verzeichnis wird automatisch eine Klasse von ASP.NET erzeugt.

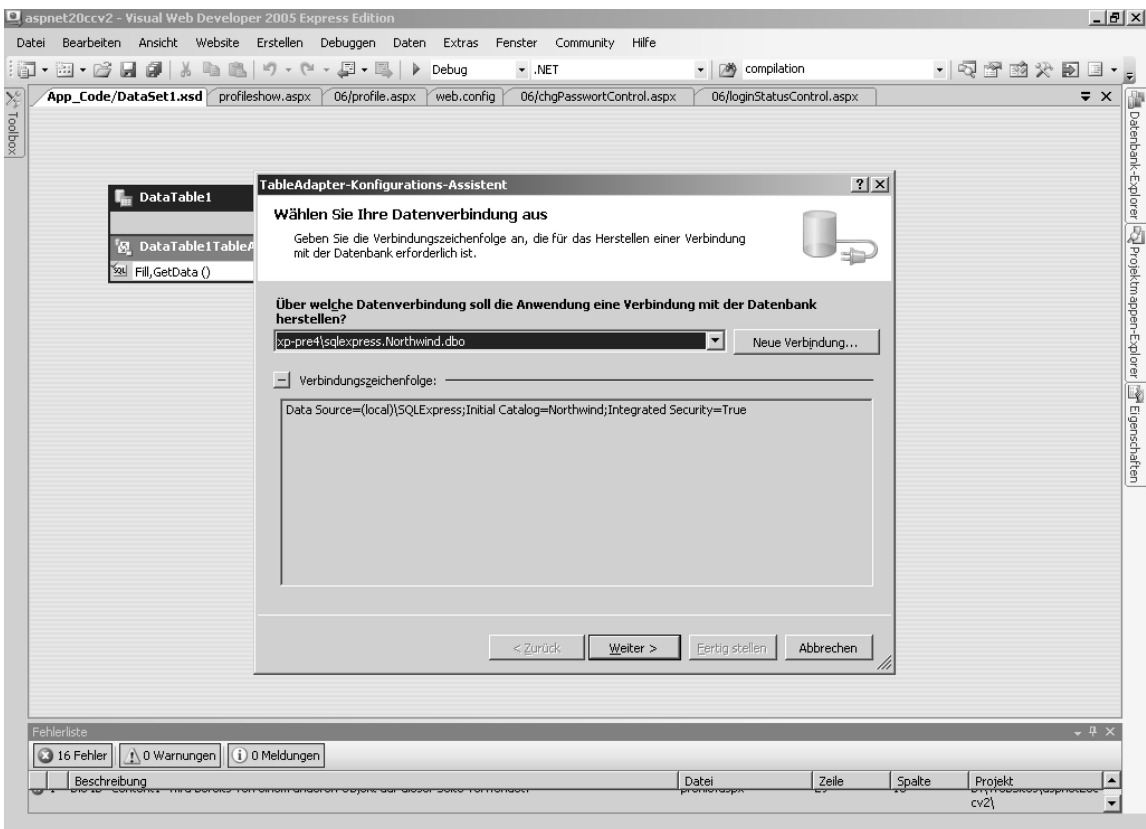


Abbildung 7.5 Table Adapter Configuration Wizard zum Erstellen eines DataSet-Schemas

Als Nächstes wird die Verbindung zur Datenbank ausgewählt oder – je nach Bedarf – eine neue Verbindung erstellt. Dabei unterscheiden sich die Dialoge je nach verwendetem Provider.

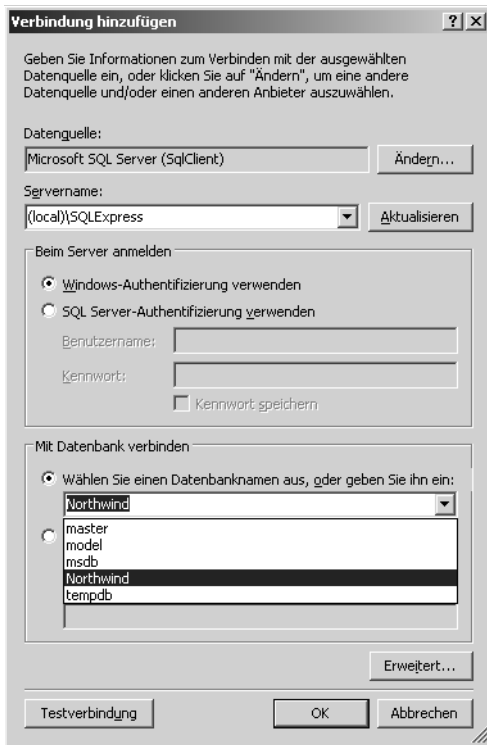


Abbildung 7.6 Erzeugen einer neuen Verbindung

Der nächste Dialog, der hier nicht abgebildet ist, erlaubt es, den Connection String automatisch speichern zu lassen. Dabei kann ausgewählt werden, ob das Passwort mit abgespeichert werden soll. Der Connection String wird in der Datei *web.config* unter ASP.NET 2.0 in dem neuen Bereich *connectionStrings* abgelegt.

```
<connectionStrings>
<add name="Settings.DataComponent.NorthwindConnection"
      connectionString="Server=localhost;Integrated Security=True;Database=Northwind"
      providerName="System.Data.SqlClient" />
</connectionStrings>
```

Listing 7.2 Ausschnitt aus der *web.config*

Im nächsten Schritt wird die Art der verwendeten Befehle ausgewählt. Natürlich kann man SQL-Kommandos verwenden. Dies wird im weiteren Dialog erläutert. Für jeden Befehlstyp wird ein eigenes Kommando benötigt. Das sind die Befehle für *Insert*, *Select*, *Delete* und *Update*.

Ebenso ist es möglich, bestehende gespeicherte Prozeduren (Stored Procedures) zu verwenden.

Besonders empfehlenswert finde ich persönlich aber die »mittlere« Option, die gleich die benötigten gespeicherten Prozeduren erzeugt.

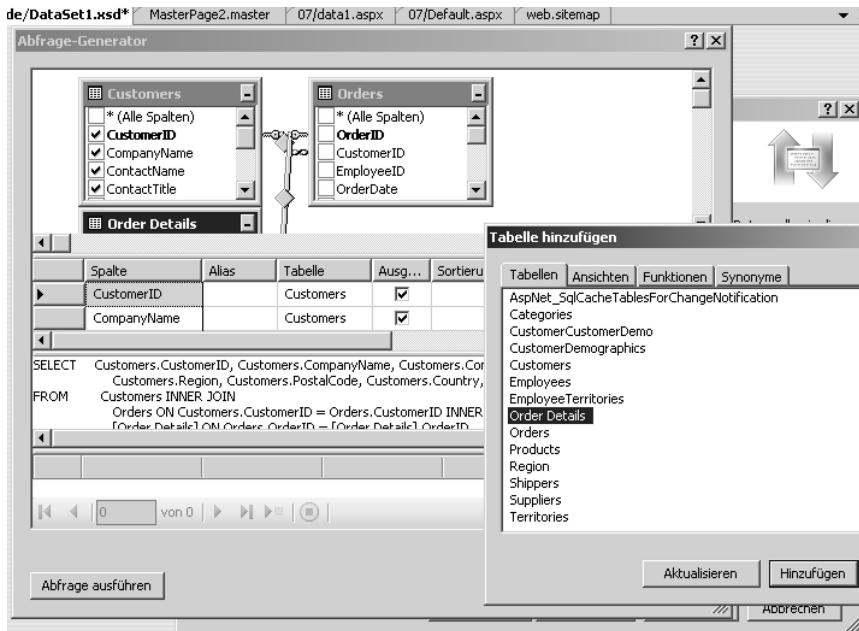


**ACHTUNG** Achten Sie darauf, dass während der Entwicklung mit anderen Rechten an der Datenbank gearbeitet wird, als zur Laufzeit. Dafür verwenden Sie am besten zwei verschiedene Datenbankbenutzer. Speziell zum Erzeugen der gespeicherten Prozeduren sind weit reichende Rechte nötig.



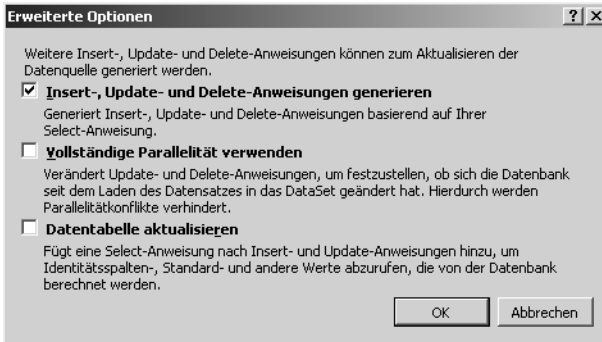
**Abbildung 7.7** Bestimmen des Abfragetyps beim Anlegen eines XSD-Datasets

Für das Erstellen von SQL-Abfragen steht ein umfangreicher Abfrage-Generator (*Querybuilder*) zur Verfügung. Dabei kann sowohl visuell als auch direkt per SQL-Kommandos gearbeitet werden.



**Abbildung 7.8** Der Abfrage-Generator hilft beim Erstellen der SQL-Abfragen

Anhand des *Select*-Kommandos werden auch die passenden SQL-Befehle für *Update*, *Delete* und *Insert* erzeugt. Das funktioniert natürlich nur, wenn eine Spalte eingebunden ist, die der Primärschlüssel ist. Wenn Sie das nicht benötigen, können Sie in dem Dialog unter *Erweiterte Optionen* die entsprechende Option(en) deaktivieren. Dort befinden sich die drei Optionen für *Vollständige Parallelität verwenden* (*optimistic concurrency*), *Datentabelle aktualisieren* und *Insert-, Update- und Delete-Anweisungen generieren*.



**Abbildung 7.9** Erweiterte Optionen beim Anlegen des DataSet XSD

Je nach gewählter Option werden dann die SQL-Kommandos erzeugt bzw. können direkt eingetragen werden.



**Abbildung 7.10** Das generierte SQL-Statement

Als Nächstes wird festgelegt, ob die erzeugte Klasse eine Methode für Lesen und Schreiben der Daten zur Verfügung stellen soll. Die Option *DataTable füllen* wird verwendet, um ein *DataSet*- oder *DataTable*-Objekt mit Daten aus der Datenbank zu füllen. Im Normalfall belässt man es beim vorgeschlagenen Namen.

Die Option *DataTable zurückgeben* definiert den Namen der Funktion, die aus dem Datenobjekt ein *DataTable*-Objekt zurückgibt.

Die dritte Option *Methoden erstellen um Updates direkt an die Datenbank zu senden* ist ausgegraut, wenn der Assistent die entsprechenden SQL-Kommandos aufgrund der Komplexität nicht erzeugen kann oder der Primärschlüssel fehlt.



Abbildung 7.11 Methoden werden erzeugt

Der letzte Dialog dient nur zur Kontrolle der durchgeführten Aktionen.

Doch was ist nun mit der XSD-Datei, dem Resultat des Assistenten, zu tun? Eine berechtigte Frage, wenn man bedenkt, dass offensichtlich keine neue Klasse, die man verwenden könnte, aus der Aktion hervorgegangen ist. Ein Blick in den Objektbrowser (`[Strg] + [Alt] + [J]`) offenbart aber, dass es jetzt mehr als ein Objekt gibt, das aus *myDataComponent* entstanden ist.

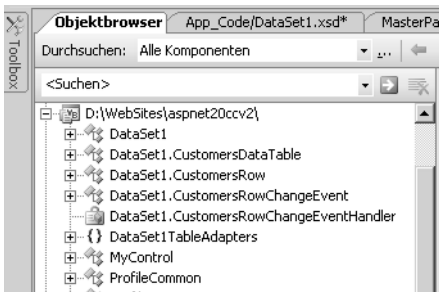


Abbildung 7.12 Der Objektbrowser enttarnt, dass mehrere Objekte aus *DataSet1* entstanden sind

Die Funktionen *Fill* und *GetData* finden sich im *TableAdapter*. Dieser wiederum befindet sich in der gleichnamigen *TableAdapters*-Auflistung (*CustomersTableAdapter*). Mit einer Codezeile kann dann eine Instanz davon erzeugt werden, mit der Daten per *GetData* in ein *Table*-Objekt gelesen und in diesem Fall per *GridView* dargestellt werden.

```
Dim myDA As New DataSet1TableAdapters.CustomersTableAdapter()
GridView1.DataSource = myDA.GetData
GridView1.DataBind()
```

**Listing 7.3** Das DataObjekt wird verwendet

Das Ganze ähnelt dem Vorgehen bei der Verwendung eines klassischen Datenzugriffs wie bei ADO.NET per DataSet. Die Datendefinitionen befinden sich aber nun in einer XSD-Datei.

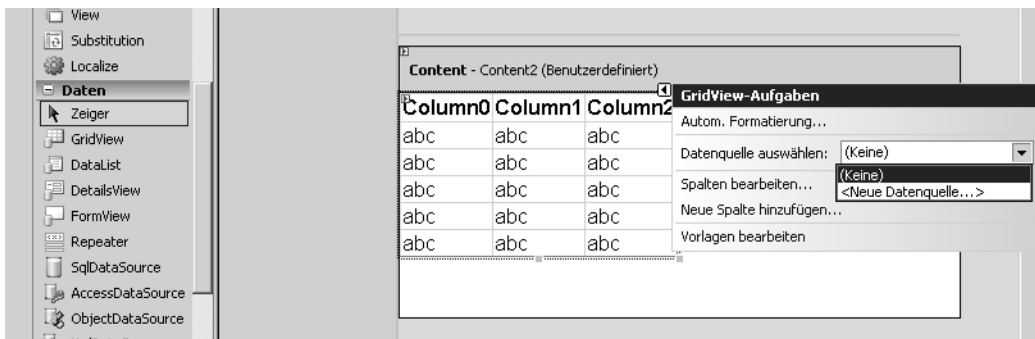
## Datenbindung ohne Code

Die DataSource-Steuerelemente erlauben es, ohne Code Daten zu visualisieren. Zur visuellen Darstellung zur Laufzeit kommen alle Steuerelemente in Frage, an die sich Daten binden lassen. In den nächsten Beispielen wird hauptsächlich das GridView-Steuerelement eingesetzt.

**ACHTUNG** Bei ASP.NET 1.x war es notwendig, die DataSource-Eigenschaft per Code zuzuweisen und dann mit dem *Bind*-Befehl zu aktivieren. Das ist jetzt nicht mehr nötig. Allerdings kann auch wie bisher ein *DataSet*-Objekt erzeugt werden. Das Vorgehen ist dann dasselbe wie in ASP.NET 1.x.

## Daten anzeigen

Um Daten schnell und ohne Einsatz von Programmcode anzuzeigen, muss das Steuerelement *GridView* per Drag & Drop aus der Werkzeugleiste auf das Webformular gezogen werden. Dann wird per *Aufgabenmenü* eine *neue Datenquelle* hinzugefügt und konfiguriert.



**Abbildung 7.13** In der Entwicklungsumgebung wird ein GridView-Steuerelement an eine Datenquelle gebunden

Aus den verschiedenen Datenquellentypen wird der Punkt *Datenbank* ausgewählt. Damit wird später ein *SqlDataSource*-Steuerelement erzeugt. Da in den Beispielen auf einen SQL Server zugegriffen wird, müssen im folgenden Dialog Daten wie Servername, Benutzername, Passwort und benötigte Datenbank im *ConnectionString* angegeben bzw. ein vorhandener *ConnectionString* ausgewählt werden.

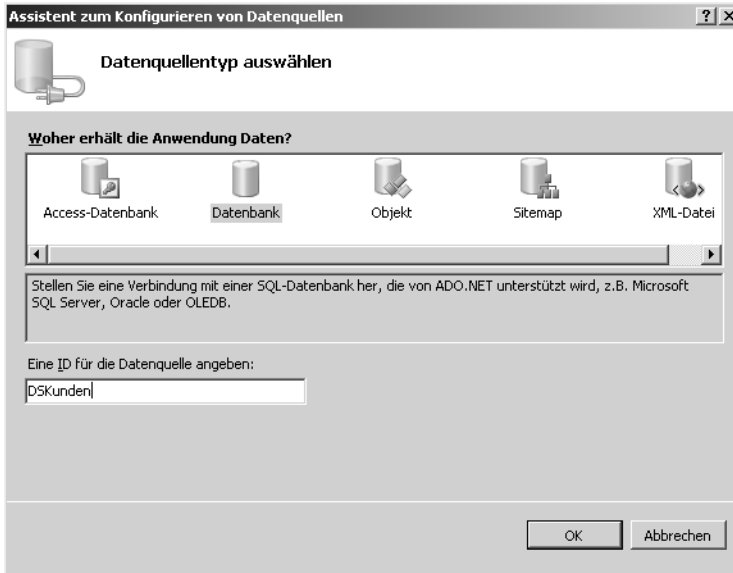


Abbildung 7.14 Datenbank wird SQLData-Source

Im nächsten Dialog *Datenquelle konfigurieren* werden die Tabelle und die anzuzeigenden Spalten ausgewählt.

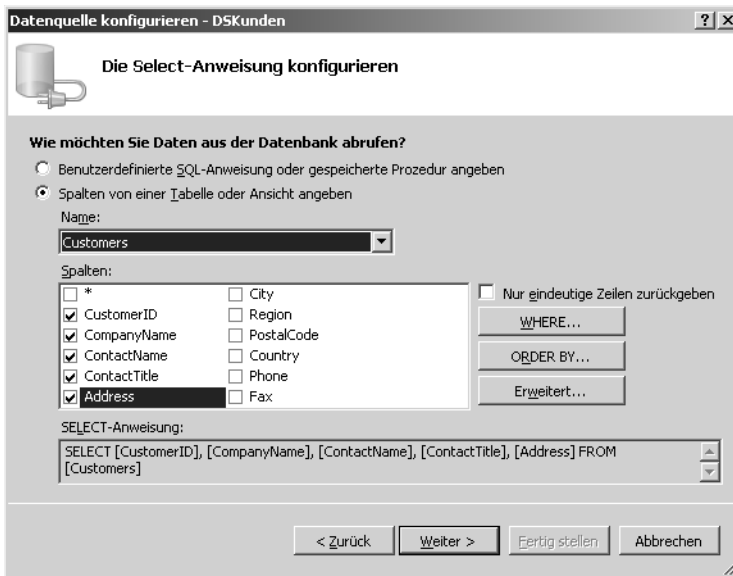


Abbildung 7.15 Tabelle auswählen

Beenden Sie den Assistenten mit der *Weiter*-Schaltfläche.

In der ASPX-Seite wird per Deklaration das SQL-Kommando hinterlegt, mit dem die Daten abgefragt werden. Die benötigten Spalten werden im *GridView* als *gebundene Spalte* (Bound Field) automatisch erzeugt. Dabei wird sogar der jeweilige Datentyp berücksichtigt. So wird ein boolescher Wert beispielsweise als *CheckBox* dargestellt. Achten Sie darauf, dass das Attribut *AutoGenerateColumns* auf *false* gesetzt ist oder die datengebundenen Felder entfernt werden, da ansonsten die Spalten doppelt dargestellt werden.

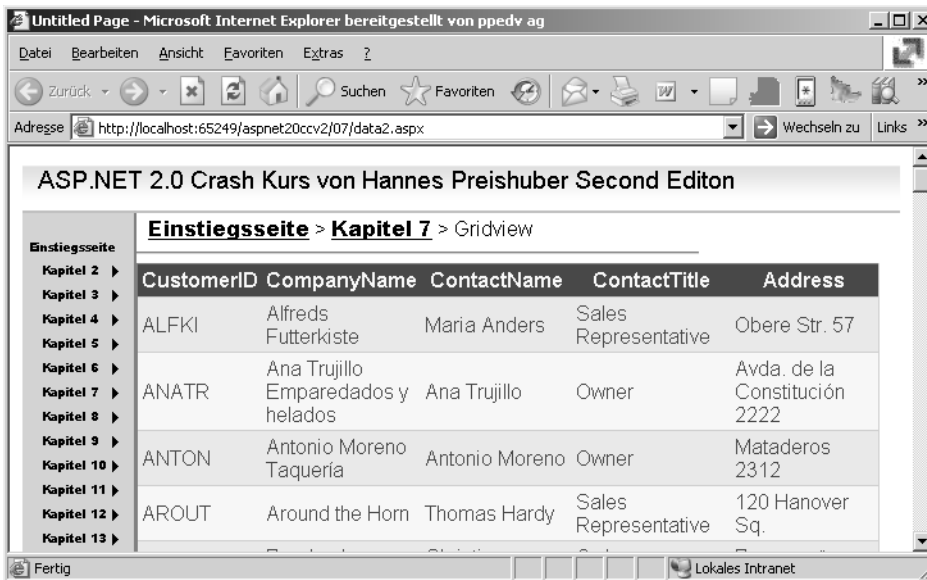
```

<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataKeyNames="CustomerID"
    DataSourceID="DSKunden">
    <Columns>
        <asp:BoundField DataField="CustomerID" HeaderText="CustomerID" ReadOnly="True"
            SortExpression="CustomerID" />
        <asp:BoundField DataField="CompanyName" HeaderText="CompanyName" SortExpression="CompanyName" />
        <asp:BoundField DataField="ContactName" HeaderText="ContactName" SortExpression="ContactName" />
        <asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle" SortExpression="ContactTitle" />
        <asp:BoundField DataField="Address" HeaderText="Address" SortExpression="Address" />
    </Columns>
</asp:GridView>
<asp:SqlDataSource ID="DSKunden" runat="server"
    ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
    SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName], [ContactTitle], [Address] FROM
    [Customers]">
</asp:SqlDataSource>

```

**Listing 7.4** SQL-Daten werden als Liste angezeigt

Mit dem Menüpunkt *Autoformat* aus dem *Aufgabenmenü* des *GridView*-Steuerelements kann der Darstellung schließlich noch ein ansprechendes Design zugeordnet werden.



**Abbildung 7.16** Formatierte tabellarische Darstellung mit GridView

## Sortieren der Ansicht

Wer schon einmal versucht hat, in einer Browser-Darstellung Daten zu sortieren, wird wissen, dass dies erheblichen Aufwand bedeutet. Selbst in ASP.NET 1.x wird dazu noch eine Menge Code benötigt. Auch hier hilft das *GridView*-Steuerelement, sodass am Schluss Sortierfunktionalität ohne Programmieraufwand möglich wird.

Die Aufgabe wird durch Setzen der Checkbox *Sorting aktivieren* im *Aufgabenmenü* des *GridView*-Steuerelements erledigt. Puristen können das auch mit dem Attribut *AllowSorting* in der ASPX-Seite beeinflussen. Das Ergebnis ist dasselbe.

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1" AllowSorting="True">
```

Der Benutzer kann zur Laufzeit dann durch Klick auf die Spaltenüberschrift nach der entsprechenden Spalte sortieren. Beim ersten Klick wird der »kleinste« Eintrag zuerst angezeigt. Beim zweiten Klick wird die Sortierreihenfolge geändert. Bei jeder Änderung der Sortierung wird ein Postback zum Server ausgeführt.

**ACHTUNG** Im Gegensatz zu ASP.NET 1.x funktioniert die Sortierung auch bei ausgeschaltetem Viewstate. Da der Viewstate große Datenmengen zwischen Server und Browser hin und her sendet, ist es empfehlenswert, ihn per *EnableViewState*-Attribut zu deaktivieren.

## Blättern

Ganz ähnlich wie die Sortierung funktioniert das Blättern in Datensätzen. Sie müssen nur das Attribut *AllowPaging* auf *true* setzen. Das können Sie per *Aufgabenmenü* oder direkt in der ASPX-Seite machen.

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
EnableViewState="False" AllowPaging="True" PageSize="5">
```

Das Erscheinungsbild und die Funktionsweise der *Paging*-Funktion lassen sich über viele Attribute beeinflussen. Dies ist in Kapitel 8 näher beschrieben.

**ACHTUNG** Bei jedem *Paging*-Vorgang werden die kompletten Daten aus der Datenbank gelesen. Wenn Sie dies umgehen wollen, müssen Sie einen eigenen *Paging*-Mechanismus implementieren oder Data Caching verwenden.

## Data Caching

Die Details von Caching wurden in Kapitel 3 beschrieben. Auch das *DataSource*-Steuerelement kann davon profitieren. Es ist ausreichend, per *EnableCaching*-Attribut das Caching zu aktivieren und die Dauer in Sekunden per *CacheDuration* anzugeben.

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
SelectCommand="SELECT [ProductName], [UnitPrice], [Discontinued] FROM [Products]"
ConnectionString="Server=localhost;User ID=sa; password=;Database=Northwind"
ProviderName="System.Data.SqlClient"
CacheDuration="60"
EnableCaching="true" >
```

**Listing 7.5** Caching im *SqlDataSource*-Steuerelement

Gerade beim *Paging* von größeren Datenmengen reduziert dies die Datenbankzugriffe erheblich.

## ControlParameter

Datentabellen mit vollständigem Inhalt darzustellen kann oft unerwünscht sein. Eine Selektion der Daten, z.B. durch Begrenzung auf einen Postleitzahlenbereich, ist eine oft formulierte Anforderung. Die DataSource-Steuerelemente haben deshalb die Möglichkeit zusätzliche Parameter für Select, Insert, Delete und Update zu verwenden. Die so definierten Parameter können Ihre Daten aus unterschiedlichen Quellen beziehen (siehe Tabelle 7.3).

Bei einem SQL-Select-Kommando wird das Element *SelectParameters* des DataSource-Steuerelements verwendet. Man kann dies direkt im Code der ASPX-Seite eintragen oder über das *Aufgabenmenü* des SQL-DataSource-Steuerelements den Menüpunkt *Datenquelle konfigurieren* aufrufen. Im übernächsten Dialog *Select Anweisung konfigurieren* kann über den Button *WHERE* das SQL-Statement mit einer Where-Bedingung ergänzt werden. Genauer gesagt wird eine Zuordnung eines Datenbankfeldes und eines Eingabe-Steuerelements vorgenommen. Hier können also eine oder auch mehrere Abhängigkeiten vom Benutzer-Steuerelement definiert werden. Mit *SelectParameters* wird also nur eine beschränkte Datenmenge aus der Datenbank übertragen.

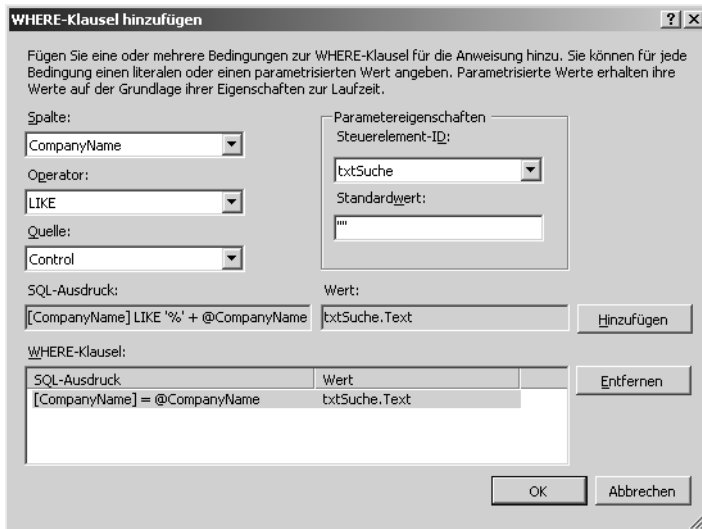


Abbildung 7.17 Where-Bedingung im SQLDataSource-Steuerelement

Dem *SQLDataSource*-Steuerelement wird so im Attribut *SelectCommand* eine *Where*-Klausel zugeordnet, die auf einen SQL-Parameter verweist. Im Unterelement *SelectParameters* wird dann ein externes Steuerelement per *ControlParameter* als Quelle definiert.

Indem hier ein Leerzeichen als Standardwert (*DefaultValue*) vorgegeben wird, ist die Liste im Browser beim ersten Aufruf ohne Eingabewert leer. Das sollte speziell bei großen Datenmengen berücksichtigt werden, da ansonsten als Erstes die ganze Tabelle zeitaufwändig geladen wird. Der Wert für die *WHERE*-Bedingung wird einem anderen Steuerelement entnommen – einer *TextBox*. Die Attribute *ControlID* und *PropertyName* stellen hier die Verbindung her.

Der *ConnectionString* ist nun in der *web.config* ausgelagert und wird per ASP.NET Expression geladen.



```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    ConnectionString="<%= $ ConnectionStrings:AppConnectionString1 %>"
    SelectCommand="SELECT [ProductName], [UnitPrice], [Discontinued] FROM [Products] WHERE
        ([ProductName] LIKE '%' + @ProductName2 + '%')">
<SelectParameters>
    <asp:ControlParameter Name="ProductName2" DefaultValue=" " Type="String"
        ControlID="TextBox1" PropertyName="Text"></asp:ControlParameter>
</SelectParameters>
</asp:SqlDataSource>
```

**Listing 7.6** Code der ASPX-Seite mit Steuerelement-Parameter

Natürlich lassen sich auch beliebige andere Webserver-Steuerelemente zur Kontrolle der Datenbank-Abfrage verwenden. Auch der eigentliche Wert muss nicht zwangsläufig an die *Text*-Eigenschaft gebunden werden. Mit dem Attribut *PropertyName* lässt sich jedes Attribut mit jedem möglichen Datentyp verwenden. Später wird diese Vorgehensweise als Grundlage für eine Master-Detail-Ansicht verwendet.

---

**HINWEIS** Je nach Datenbank werden die SQL-Parameter unterschiedlich gehandhabt. SQL Server verwendet das @ gefolgt vom Namen des Parameters. Access verwendet ein ? ohne Namenszusatz. Bei Access müssen deshalb die Parameter in der Reihenfolge stimmen.

---

## Daten ändern

Auch das Editieren und Löschen von Daten ist grundsätzlich ohne Code mit *GridView* und *DataSource* möglich. Allerdings müssen gegenüber der bisherigen Lösung kleine Änderungen vorgenommen werden. Es wird das SQL-Kommando für das Update benötigt.

Dafür muss zunächst die Datenquelle im *SQLDataSource*-Steuerelement definiert werden. Wählen Sie bei Ihrer Datenquelle aus dem *Aufgabenmenü* den Menüpunkt *Datenquelle konfigurieren* aus. Im Dialogschritt *Select Anweisung konfigurieren* wählen Sie den Button *erweitert* aus. Nur wenn die Daten aus einer Tabelle oder Sicht (View) kommen, ist diese Option verfügbar, andernfalls ist diese deaktiviert.

Im folgenden Dialog *erweiterte SQL Generierungs Optionen* sind die Optionen *Insert*, *Update* und *Delete Anweisung generieren* und *vollständige Parallelität verwenden (optimistic concurrency)* deaktiviert, wenn das Select Statement nicht eine Spalte mit der Eigenschaft *Primärschlüssel (Primary Key)* umfasst.

Mit der ersten Option werden damit parametrisierte SQL-Kommandos erzeugt. Die Parameter für die SQL-Kommandos werden über die Elemente *DeleteParameter*, *InsertParameter* und *UpdateParameter* definiert. Beenden Sie den Assistenten unter Auswahl der ersten Option

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    ProviderName="<%= $ ConnectionStrings:AppConnectionString1.ProviderName %>"
    UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName, [ContactName] = @ContactName,
        [ContactTitle] = @ContactTitle, [Address] = @Address, [City] = @City, [Region] = @Region,
        [PostalCode] = @PostalCode, [Country] = @Country, [Phone] = @Phone, [Fax] = @Fax
        WHERE [CustomerID] = @CustomerID"
    ...
    DeleteCommand="DELETE FROM [Customers] WHERE [CustomerID] = @CustomerID"
    SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName], [ContactTitle], [Address],
        [City], [Region], [PostalCode], [Country], [Phone], [Fax] FROM [Customers] "
    ConnectionString="<%= $ ConnectionStrings:AppConnectionString1 %>">
```

**Listing 7.7** SQLDataSource mit zusätzlichem *Update*- und *Delete*-Kommando

```

<DeleteParameters>
  <asp:Parameter Type="String" Name="CustomerID"></asp:Parameter>
</DeleteParameters>
<UpdateParameters>
  <asp:Parameter Type="String" Name="CompanyName"></asp:Parameter>
  <asp:Parameter Type="String" Name="ContactName"></asp:Parameter>
...
</UpdateParameters>
</asp:SqlDataSource>

```

**Listing 7.7** SQLDataSource mit zusätzlichem *Update*- und *Delete*-Kommando (Fortsetzung)

Natürlich kann sowohl per Assistent als auch in der ASPX-Seite jedes SQL-Kommando geändert werden. Eventuell muss dann das Schema des *GridView*-Steuerelements aktualisiert werden, um auch in der Entwicklungsumgebung die aktuellen Felder zu sehen.

Als Nächstes müssen im *GridView*-Steuerelement die Buttons für die *Edit*- und *Delete*-Aktionen erzeugt werden. Auch das lässt sich per *Aufgabenmenü* des *GridView*-Steuerelements erledigen. Wählen Sie dort die Optionsbox *bearbeiten aktivieren*. Dadurch wird eine neue Spalte im *Columns*-Element erzeugt, das *CommandField*.

```
<asp:CommandField ShowEditButton="True" />
```

Alternativ kann eine solche Spalte auch anders erzeugt werden. Wählen Sie im Aufgabenmenü des *GridView*-Steuerelements dazu den Menüpunkt *Spalten bearbeiten* aus. Im Dialogfenster *Felder* können dann die Felder definiert werden. Kapitel 8 beschreibt die möglichen Einstellungen des *GridView*-Steuerelements im Detail.

In der ASPX-Seite ist im *GridView*-Steuerelement anschließend ein *Columns*-Bereich vorhanden. Je Spalte ist ein ASP.NET-Element vorhanden.

```

<Columns>
  <asp:BoundField HeaderText="ProductName" DataField="ProductName"
    SortExpression="ProductName"></asp:BoundField>
  <asp:BoundField HeaderText="UnitPrice" DataField="UnitPrice"
    SortExpression="UnitPrice"></asp:BoundField>
  <asp:CheckBoxField HeaderText="Discontinued" SortExpression="Discontinued"
    DataField="Discontinued"></asp:CheckBoxField>
  <asp:CommandField ShowEditButton="True"></asp:CommandField>
  <asp:CommandField ShowDeleteButton="True"></asp:CommandField>
</Columns>

```

**Listing 7.8** Die *Columns*-Auflistung des *GridView*-Steuerelements

Bei der Standardeinstellung werden die Aktionen in der Webseite über Hyperlinks durchgeführt. Es lassen sich aber natürlich per Attribut auch Buttons daraus machen. Auch hier funktioniert das Beispiel im Browser sofort. Wenn der Benutzer zur Laufzeit auf *bearbeiten* klickt, wird der Datensatz mit Textboxen bzw. aktivierten Checkboxen dargestellt. Statt des *bearbeiten*-Links erscheinen zwei Links mit Beschriftung und den Funktionen *aktualisieren* und *abbrechen*.

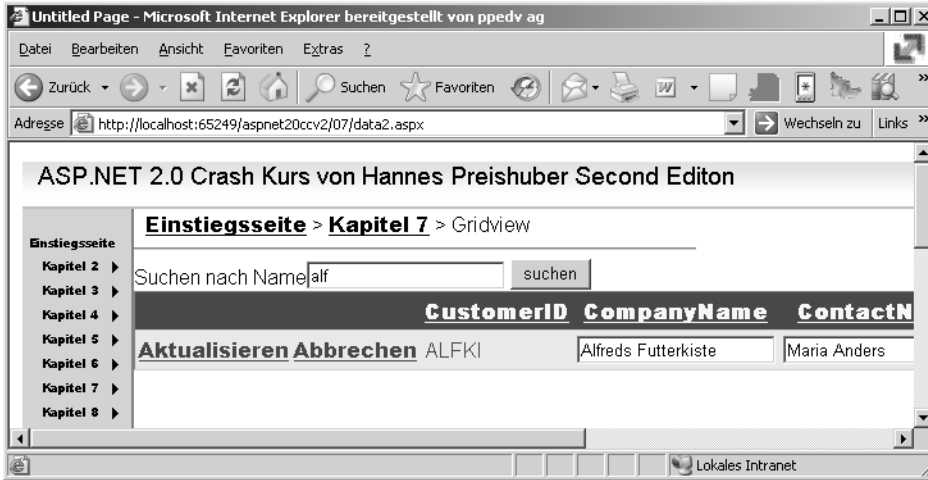


Abbildung 7.18 Ein Datensatz wird im GridView editiert

Um einen Datensatz zu löschen, müssen Sie sicherstellen, dass das *Delete*-Kommando im *SQLDataSource*-Steuerelement definiert und die Spalte im *GridView*-Steuerelement vorhanden ist. Der Datensatz wird dann ohne Rückfrage gelöscht.

### Null-Werte

Ein weiteres Problem ergibt sich, wenn die Tabellen-Felder Nullwerte enthalten. Eine Textbox enthält, wenn diese leer ist, ein Leerzeichen. Ein Update gegen ein Feld vom Typ Datum muss dann fehlschlagen.

Die Lösung dazu ist das Attribut im ControlParameter *ConvertEmptyStringToNull*, mit dem eben ein Leerzeichen dann in ein *DBNull* übersetzt wird.

**ACHTUNG** Das *GridView*-Steuerelement kann keine neuen Datensätze einfügen. Dazu dient das *DetailsView*-Steuerelement, das in Kapitel 8 behandelt wird.

## Datenquellen-Steuerelemente

Um den Entwicklern das Schreiben von Codezeilen zu ersparen, hat das ASP.NET-Team die *DataSource*-Steuerelemente entwickelt. Bisher wurde aus dieser Gruppe hier das *SqlDataSource*-Steuerelement kurz beschrieben. Dies sind visuelle Steuerelemente, die per Deklaration gesteuert werden. Für jede Art von Datenquelle wird ein eigenes *DataSource*-Steuerelement benötigt. Nach außen hin sehen alle gleich aus. Genauer gesagt, ist die Schnittstelle durch das Interface *IDataSource* identisch.

Die Komplexität und Zugriffsdetails sind sozusagen im Steuerelement versteckt. Der Entwickler muss nur die Kommandos oder Funktionsnamen für *SELECT*, *INSERT*, *DELETE* und *UPDATE* per Attribut deklarieren.

Nicht ganz zu unterschätzen ist der visuelle Teil in der Entwicklungsumgebung. Mit dem *Aufgabenmenü* können verschiedene Assistenten gestartet werden, die auch komplexere Tätigkeiten übernehmen.

Die Webserver-Steuerelemente, die Daten darstellen, werden durch das Attribut *DataSourceID* mit dem DataSource-Steuerelement verbunden. Steuerelemente, die dafür in Fragen kommen, befinden sich in den Namensräumen *System.Web.UI.WebControls* und *System.Web.UI.HtmlControls*

Es gibt zwei Grundtypen von DataSource-Steuerelementen, die normalerweise an unterschiedliche Darstellungs-Steuerelemente gebunden werden. Die Basisklasse *DataSourceControl* wird für alle relationalen Datenquellen verwendet; die zweite Basisklasse *HierarchicalDataSourceControl* für alle hierarchischen Datenquellen wie XML.

In Visual Studio 2005 sind folgende Steuerelemente vorhanden:

DataSource Steuerelement	Verwendung
<i>SQLDataSource</i>	Wird für Verbindung zu SQL-basierten Server Systemen verwendet.
<i>AccessDataSource</i>	Wird für Verbindung zu Access-Datenquellen verwendet. Access-Dateien haben die Endung MDB. Sie sollten im Verzeichnis <i>app_data</i> abgelegt werden.
<i>XmlDataSource</i>	Wird für die Verbindung zu XML-Daten verwendet. Die Daten können aus Dateien oder auch z.B. von einem URL stammen.
<i>SiteMapDataSource</i>	Wird für die Verbindung zu einer SiteMap-Datei verwendet.
<i>ObjectDataSource</i>	Wird für die Verbindung zu einem Geschäftsobjekt verwendet.

**Tabelle 7.1** DataSource-Steuerelemente

Nach Aussagen von Microsoft sind noch weitere DataSource-Steuerelemente zu erwarten. So ist ein Steuerelement für den Excel- oder Index-Server-Zugriff denkbar. Es ist auch wahrscheinlich, dass mindestens ein DataSource-Steuerelement samt Quellcode ausgeliefert wird, sodass eigene Anpassungen einfach durchzuführen sind.

## SQLDataSource

In diesem Kapitel ist das *SQLDataSource*-Steuerelement bereits mehrfach eingesetzt worden. Obwohl der Name es vermuten lässt, ist dieses Steuerelement nicht nur für den SQL Server oder MSDE verwendbar. Durch das zusätzliche Attribut *ProviderName* eignet sich das Steuerelement auch für andere Datenbanktypen die mit SQL-Kommandos gesteuert werden.

Im einfachsten Fall wird auf diese Weise durch das Ziehen einer Tabelle (per Drag & Drop) aus dem Server Explorer eine *SQLDataSource* aus einer bestehenden Datenquelle erzeugt.

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server" SelectCommand="SELECT [ProductID],
    [ProductName], [SupplierID] FROM [Alphabetical list of products]"
    ConnectionString="<%= $ ConnectionStrings:AppConnectionString1 %>">
</asp:SqlDataSource>
```

**Listing 7.9** Deklarative Datenbindung mit SQLDataSource

Obwohl ohne Code erzeugt, kann das *SQLDataSource*-Steuerelement auch per Code gesteuert werden. So lässt sich z.B. ein neuer Datensatz erzeugen, indem die nötigen Parameter belegt werden und das Insert Kommando ausgeführt wird.

```
Protected Sub LinkButton1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    DSKunden.InsertParameters("customerid").DefaultValue = "NEUER"
    DSKunden.Insert()
End Sub
```

**Listing 7.10** Neuer Datensatz mit `SQLDataSource`-Steuerelement

## Verfügbare Provider

Die für das `SqlDataSource`-Steuerelement verfügbaren Provider sind in der `machine.config` deklariert. Im Folgenden finden Sie einen Auszug daraus, der das Element `DbProviderFactories` wiedergibt.

```
<system.data>
  <DbProviderFactories>
    <add name="Odbc Data Provider" invariant="System.Data.Odbc" description=".Net Framework Data
    Provider for Odbc" type="System.Data.Odbc.OdbcFactory, System.Data, Version=2.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
    <add name="OleDb Data Provider" invariant="System.Data.OleDb" description=".Net Framework Data
    Provider for OleDb" type="System.Data.OleDb.OleDbFactory, System.Data, Version=2.0.0.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <add name="OracleClient Data Provider" invariant="System.Data.OracleClient" description=".Net
    Framework Data Provider for Oracle" type="System.Data.OracleClient.OracleClientFactory,
    System.Data.OracleClient, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <add name="SqlClient Data Provider" invariant="System.Data.SqlClient" description=".Net Framework
    Data Provider for SqlServer" type="System.Data.SqlClient.SqlClientFactory, System.Data,
    Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </DbProviderFactories>
</system.data>
```

**Listing 7.11** Provider-Deklaration in `machine.config`

## SQLDataSource-Eigenschaften

Obwohl sehr umfangreich, werden hier die Eigenschaften und Methoden der `SQLDataSource` genauer beschrieben. Es gibt so viele neue Möglichkeiten, die helfen, schneller zum Ziel zu kommen, dass es sich lohnt, sich mit den Details auseinander zu setzen. Die üblichen und bisher bereits verwendeten und erklärten Steuerelement-Attribute fehlen in der folgenden Auflistung.

Eigenschaft	Verwendung
<code>CacheDuration</code>	Setzt oder liest die Dauer in Sekunden für das Daten-Caching des Steuerelements. <code>-1</code> bedeutet, dass unendlich lange gewartet wird.
<code>CacheExpirationPolicy</code>	Setzt oder liest die Einstellung, die bestimmt, wie die Behandlung der in <code>CacheDuration</code> gesetzten Zeit vorgenommen wird. Mögliche Parameter sind <i>Absolute</i> oder <i>Sliding</i> . Wenn <i>Sliding</i> gesetzt ist, wird die Ablaufdauer des Caches ab letzter Verwendung berechnet. Bei einer dauerhaft benutzten Seite müssen dadurch Daten nie erneuert werden.
<code>CacheKeyDependency</code>	Setzt oder liest die Einstellung für die Abhängigkeit des Caches.
<code>CancelSelectOnNullParameter</code>	Setzt oder liest die möglichen Werte <i>true</i> oder <i>false</i> . Ist diese Eigenschaft <i>true</i> , wird geprüft, ob einer der Parameter des SQL-Kommandos <i>Nothing</i> (Null C#) enthält, was bewirkt, dass die durchzuführende Abfrage gegebenenfalls abgebrochen wird.

**Tabelle 7.2** Auszug aus den Eigenschaften des `SQLDataSource`-Steuerelements

Eigenschaft	Verwendung
<i>ConflictDetection</i>	Bestimmt oder ermittelt die möglichen Werte <i>CompareAllValues</i> oder <i>OverwriteChanges</i> . Damit lässt sich das Verhalten bei Update- oder Delete-Kommandos steuern, wenn sich die Originaldaten verändert haben.
<i>ConnectionString</i>	Setzt oder liest die Verbindungszeichenfolge. Die Verbindungszeichenfolge muss grundsätzlich angegeben werden, damit eine Verbindung zur Datenbank überhaupt aufgebaut werden kann.
<i>DataSourceMode</i>	Mit dieser Eigenschaft steuern Sie, welche Klassen zum Zugriff auf Daten verwendet werden ( <i>DataReader</i> bzw. <i>DataSet</i> ). Einige Operationen, wie beispielsweise das Caching, erfordern hier vom Standard ( <i>DataSet</i> ) abweichende Einstellungen.
<i>DeleteCommand</i>	Bestimmt oder ermittelt das SQL-Kommando, das beim Aufruf der <i>Delete</i> -Funktion ausgeführt wird. Achten Sie darauf, dass die SQL-Kommandos passend zum Datenprovider formuliert sein müssen – also z.B. »@« als Präfix für SQL-Parameter und »?« als Präfix für ODBC-Parameter.
<i>DeleteCommandType</i>	Damit wird definiert, ob das Delete Kommando vom Typ <i>SQL(Text)</i> oder gespeicherte Prozedur ( <i>StoredProcedure</i> ) ist. Das gleiche gilt für <i>SelectCommandType</i> , <i>InsertCommandType</i> und <i>UpdateCommandType</i> .
<i>DeleteParameters</i>	Setzt oder liest die <i>Parameter</i> -Auflistung zum <i>Delete</i> -Kommando. Diese kann auch aus anderen Steuerelementen stammen.
<i>EnableCaching</i>	Setzt oder liest den Wert, der bestimmt, ob Data Caching eingeschaltet ( <i>true</i> ) oder ausgeschaltet ( <i>false</i> ) sein soll.
<i>FilterExpression</i>	Setzt oder liest einen Ausdruck zum Filtern der Daten. Dieser bezieht sich auf die <i>Parameter</i> -Auflistung.
<i>FilterParameters</i>	Setzt oder liest eine <i>Parameter</i> -Auflistung. Diese bezieht sich z.B. auf den Inhalt von externen Steuerelementen.
<i>InsertCommand</i>	Setzt oder liest das SQL-Kommando, das beim Aufruf der <i>Insert</i> -Funktion ausgeführt wird.
<i>InsertParameters</i>	Bestimmt oder ermittelt die <i>Parameter</i> -Collection zum <i>Insert</i> -Kommando. Diese können auch aus anderen Steuerelementen stammen
<i>OldValuesParameterFormatString</i>	Setzt oder liest den Format-String, der für die ursprünglichen Werte verwendet wird. Das hat nur Auswirkungen, wenn der <i>Delete</i> - bzw. <i>Update</i> -Modus auf <i>Pessimistic</i> gesetzt ist. Der Default-Wert ist »{0}«.
<i>ProviderName</i>	Setzt oder liest den Namensraum des verwendeten Datenproviders. Fehlt diese Angabe, wird mit <i>System.Data.SqlClient</i> gearbeitet.
<i>SelectCommand</i>	Setzt oder liest das SQL-Kommando, das beim Aufruf der <i>Select</i> -Funktion ausgeführt wird.
<i>SelectParameters</i>	Setzt oder liest die <i>Parameter</i> -Auflistung zum <i>Select</i> -Kommando. Diese kann auch aus anderen Steuerelementen stammen.
<i>SortParameterName</i>	Setzt oder liest den Parameter, der für die Sortierung verwendet wird. Dieser wird nur ausgewertet, wenn der <i>Select</i> -Befehl eine <i>Stored Procedure</i> (gespeicherte Prozedur) aufruft.
<i>SqlCacheDependency</i>	Setzt oder liest die Einstellung für die SQL-Cache-Abhängigkeit. Dies wird nur von wenigen Datenbanken unterstützt. Die Zeichenkette wird in der Form »Connection:Tabelle« angegeben. Außerdem muss in der <i>web.config</i> ein entsprechender Eintrag vorhanden sein.
<i>UpdateCommand</i>	Setzt oder liest das SQL-Kommando, das beim Aufruf der <i>Delete</i> -Funktion ausgeführt wird.
<i>UpdateParameters</i>	Setzt oder liest die <i>Parameter</i> -Auflistung zum <i>Update</i> -Kommando. Diese kann auch aus anderen Steuerelementen stammen

**Tabelle 7.2** Auszug aus den Eigenschaften des *SQLDataSource*-Steuerelements (Fortsetzung)

## SelectParameter

In einem SQL-Kommando, z.B. dem *Select*-Command, können ein oder mehrere SQL-Parameter angegeben werden, um die Abfrage mit einer *Where*-Bedingung einzuschränken. Sämtliche Parameter finden sich in der *Parameter*-Auflistung. Im *DataSource*-Steuerelement gibt es dafür das Unterelement *SelectParameters*, innerhalb dessen alle Parameter definiert werden. Der ASP.NET-*ControlParameter* erhält seinen Wert von einem ASP.NET Webserver-Steuerelement, das mit dem Attribut *ControlID* das Steuerelement benennt, aus dem der eigentliche Parameterwert stammen soll. Dazu wird auch das Attribut *PropertyName* benötigt, das auf die Eigenschaft des Steuerelements verweist, die den Parameterwert liefert. Der Name des Parameters wird im SQL-*Select*-Kommando verwendet. Je nach Datentyp ist es erforderlich, auch den Typ – hier *String* – anzugeben.

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    ConnectionString="<%= $ ConnectionStrings:AppConnectionString1 %>"
    SelectCommand="SELECT [ProductName], [UnitPrice], [Discontinued] FROM [Products]
        WHERE ([ProductName] LIKE '%' + @ProductName2 + '%')">
    <SelectParameters>
        <asp:ControlParameter Name="ProductName2" DefaultValue=" " Type="String"
            ControlID="TextBox1" PropertyName="Text"></asp:ControlParameter>
    </SelectParameters>
</asp:SqlDataSource>
```

**Listing 7.12** ControlParameter im SqlDataSource-Steuerelement

Es lassen sich mehrere Parameter kombinieren. Das SQL-Kommando muss dann entsprechend angepasst werden. Nicht nur von Steuerelementen können die Parameter kommen, sondern auch noch aus anderen Quellen.

ParameterTyp	Verwendung
<i>ControlParameter</i>	Der Wert stammt aus einem Web Server- oder HTML Server Steuerelement, dessen Name über das Attribut <i>Name</i> definiert wird. Als zusätzliches Attribut muss <i>PropertyName</i> angegeben werden.
<i>CookieParameter</i>	Der Wert stammt aus einem Cookie. Der Name des Cookies wird über <i>CookieName</i> definiert.
<i>FormParameter</i>	Der Wert stammt aus einem Formularfeld. Dieses wird per POST oder GET übergeben. Der Name des Attributs zum Zuweisen des Feldes lautet <i>FormField</i> .
<i>Parameter</i>	Der Wert stammt vermutlich aus dem Attribut <i>DefaultValue</i> . Die finale Online-Hilfe der Beta 2 wird darüber hoffentlich mehr Auskunft geben können.
<i>ProfileParameter</i>	Der Wert stammt aus dem ASP.NET-Personalisierungssystem. Die Zuweisung erfolgt per <i>PropertyName</i> -Attribut.
<i>QueryStringParameter</i>	Der Wert stammt aus einem QueryString. Der Name der <i>QueryString</i> -Variablen wird in <i>QueryStringField</i> eingegeben.
<i>SessionParameter</i>	Der Wert stammt aus einer Session-Variablen, deren Name in das Attribut <i>SessionField</i> eingetragen wird.

**Tabelle 7.3** Die möglichen Parametertypen eines DataSource-Steuerelements

Möglicherweise ist Ihnen schon das Attribut *Direction* aufgefallen. Damit ist natürlich die Übergaberichtung des Parameters gemeint. Gültige Werte sind *Input*, *Output*, *InputOutput* und *ReturnValue*. Speziell im Zusammenhang mit Stored Procedures spielt die Verwendung eine Rolle.

## Dynamische DataSource

Sowohl *SQLDataSource* als auch zugehörige Parameter lassen sich dynamisch per Code generieren. In diesem Beispiel wird eine *SQLDataSource* erzeugt und der Steuerelement-Auflistung der Seite hinzugefügt. Der Parameter wird vorher noch per *Add* dem *SQLDataSource*-Steuerelement zugewiesen. Den *ConnectionString* (die Verbindungszeichenfolge) kann man mit dem *ConfigurationManager*-Objekt aus der *web.config* auslesen.

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim sqlSource As SqlDataSource
    sqlSource = New
    SqlDataSource(ConfigurationManager.ConnectionStrings("AppConnectionString1").ConnectionString,
        "SELECT FirstName, LastName FROM Employees WHERE LastName LIKE @FName+'%';")
    Dim famName As New Parameter()
    famName.Name = "FName"
    famName.Type = TypeCode.String
    famName.DefaultValue = "D"
    sqlSource.SelectParameters.Add(famName)
    Page.Controls.Add(sqlSource)
    GridView1.DataSource = sqlSource
    GridView1.DataBind()
End Sub
```

**Listing 7.13** Dynamisch zugewiesener Parameter

## Cachen von Daten

In Kapitel 3 wurden schon die Grundlagen des Caching erläutert. Auch das *SQLDataSource*-Steuerelement unterstützt Caching. Gecacht wird dabei ein *DataSet*. Die Dauer wird in Sekunden über das Attribut *CacheDuration* festgelegt. Das alleine ist aber noch nicht ausreichend. Das Caching muss grundsätzlich noch mit dem Attribut *enablecaching* aktiviert werden.

```
<asp:SqlDataSource ID="SqlDataSource1"
    CacheDuration="60"
    enablecaching="true"
    Runat="server" ConnectionString="<%= $ ConnectionStrings:AppConnectionString1 %>"
    SelectCommand="SELECT [ProductID], [ProductName], [SupplierID] FROM [Alphabetical list of
products]">
</asp:SqlDataSource>
```

**Listing 7.14** Der Inhalt des *SQLDataSource*-Steuerelements wird gecacht

Wenn nun ein zweiter Benutzer innerhalb von Sekunden die Seite abrufen, wird diese aus dem Cache geladen. Der Datenbankserver wird dann nicht mit einer Abfrage belastet.

---

**ACHTUNG** Soll Performance eine wichtige Rolle spielen, sollten Sie immer das Tool SQL-Profilier zur Leistungsanalyse verwenden. Damit kann man sehr gut erkennen, welche Abfragen wie oft an die Datenbank gesendet werden. Ein falsch konfigurierter Cache verursacht meist keine Laufzeitfehler, sodass z.B. häufige und unnötige Anfragen an den Datenbank-Server unbemerkt bleiben und die Leistungsfähigkeit darunter leidet.

---

## Filtern von Daten

Im Gegensatz zur Verwendung von *Select*-Parametern wird mit den Filterparametern eine Offline-Filterung durchgeführt. Das bedeutet, dass die Daten komplett aus der Datenbank geladen werden. Erst ASP.NET selekt-



tiert dann die anzuzeigenden Datensätze. Ein mögliches Einsatzszenario wären mehrere Steuerelemente in der Seite, deren Werte abhängig von dem Inhalt eines DataSource-Steuerelements visualisiert werden sollen.

Richtig elegant wird es im Zusammenspiel mit Caching. Daten, die sich nicht allzu oft ändern und durchsucht werden sollen, lassen sich so optimal handhaben. Ein Artikelkatalog eines Online-Shops ist dafür ein hervorragendes Beispiel. Die Grunddaten der Artikel ändern sich eher selten. Die Besucher suchen aber häufig in diesen Daten. Um hier maximalen Durchsatz zu erzielen, ist die Kombination von Daten-Caching und -Filterung die optimale Lösung.

Ganz ähnlich wie bei den *Select*-Parametern werden ein oder mehrere Parameter in der *FilterParameter*-Auflistung des *SqlDataSource*-Steuerelements definiert. Die Parameter wiederum sind in der ASPX-Seite selbst Unterelemente. Es gibt, genau wie bei *Select*-Parametern, *ControlParameter*, *CookieParameter*, *FormParameter*, *ProfileParameter*, *QueryStringParameter* und *SessionParameter*.

Ein wichtiger Unterschied liegt in der Verwendung der Parameter. Diese werden durchnummeriert und in geschweifte Klammern »{}« gesetzt. Ein weiterer Unterschied ist, dass kein Assistent hilft und alles direkt in der ASPX-Seite in der *Quell*-Ansicht bearbeitet werden muss.

Mit dem Attribut *FilterExpression* können »SQL-WHERE-ähnliche« Bedingungen definiert werden. Mehrere Bedingungen können mit AND, OR usw. kombiniert werden:

```
<asp:SqlDataSource ID="SqlDataSource1"
  CacheDuration="60"
  enablecaching="true"
  FilterExpression="ProductName LIKE '{0}%"
  Runat="server" ConnectionString="<%= $ ConnectionStrings:AppConnectionString %>"
  SelectCommand="SELECT [ProductID], [ProductName], [SupplierID] FROM [Alphabetical list of_
  products]">
  <FilterParameters>
    <asp:ControlParameter Name="PName" Type="String" ControlID="TextBox1"
      PropertyName="Text"></asp:ControlParameter>
  </FilterParameters>
</asp:SqlDataSource>
```

Listing 7.15 Daten werden gecacht und gefiltert

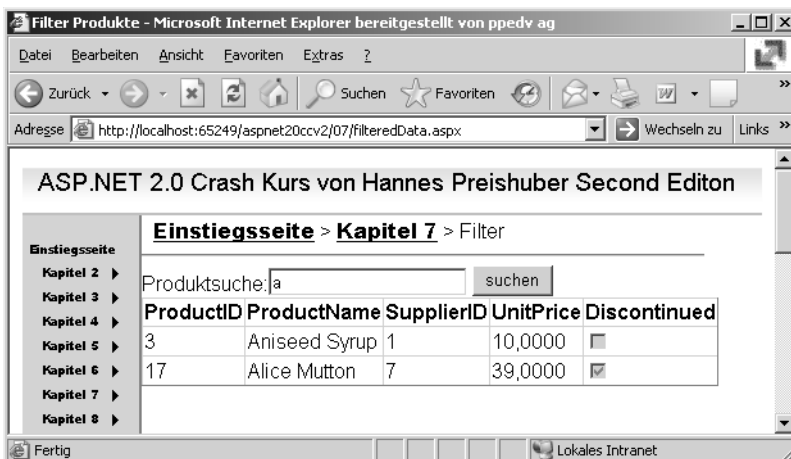


Abbildung 7.19 Filter von Daten im Cache

Man merkt beim Aufruf im Browser den enormen Performance-Gewinn schon ohne Messung. Zugriffe auf den Speicher sind nun einmal einfach mehr als ca. 100-mal schneller als Zugriffe auf Datenbanken.

## Ereignisse

Auch wenn auf den ersten Blick das `SQLDataSource`-Steuerelement keine Wünsche mehr offen lässt, stößt man in der Praxis doch auf Grenzen. Natürlich lässt sich auch ein eigenes `DataSource`-Steuerelement erstellen, indem von der entsprechenden Basisklasse `DataSourceControl` geerbt wird. Doch es muss nicht gleich so viel Aufwand getrieben werden. Als erste Alternative bietet sich an, die Ereignismethoden des Steuerelements zu nutzen.

Bislang schon bekannte Steuerelement-Ereignisse sind in der folgenden Tabelle nicht erwähnt:

Ereignis von <code>SQLDataSource</code>	Beschreibung
<code>DataBinding</code>	Wird ausgelöst, wenn Daten gebunden werden.
<code>Deleted</code>	Wird ausgelöst, wenn Daten gelöscht worden sind.
<code>Deleting</code>	Wird ausgelöst, wenn die Daten gelöscht werden sollen. Der Löschvorgang kann mit der Eigenschaft <code>Cancel</code> der Event-Argumente abgebrochen werden.
<code>Filtering</code>	Der Event wird beim Filtervorgang ausgeführt. Dort kann z.B. mit <code>e.ParameterValues.Item(0).ToString</code> auf die Filterwerte zugegriffen werden.
<code>Inserted</code>	Wird ausgelöst, wenn Daten eingefügt worden sind. Mit der Eigenschaft <code>AffectedRows</code> der Event-Argumente kann die Anzahl der neuen Datensätze ermittelt werden. Als weitere Eigenschaft steht <code>Exception</code> zur Verfügung, mit der eine etwaige Fehlermeldung ausgelesen werden kann.
<code>Inserting</code>	Wird ausgelöst, wenn Daten eingefügt werden sollen. Der Vorgang kann mit der Eigenschaft <code>Cancel</code> der Event-Argumente abgebrochen werden.
<code>Selected</code>	Wird ausgelöst, wenn Daten selektiert worden sind.
<code>Selecting</code>	Wird ausgelöst, wenn Daten per <code>Select</code> -Kommando ausgewählt werden sollen. Der Vorgang kann mit der Eigenschaft <code>Cancel</code> der Event-Argumente abgebrochen werden.
<code>Updated</code>	Wird ausgelöst, wenn Daten gespeichert worden sind.
<code>Updating</code>	Wird ausgelöst, wenn die Daten per <code>Update</code> -Kommando gespeichert werden sollen. Der Vorgang kann mit der Eigenschaft <code>Cancel</code> der Event-Argumente abgebrochen werden.

**Tabelle 7.4** Die wichtigsten Ereignisse des `SQLDataSource`-Steuerelements

## AccessDataSource

Das `AccessDataSource`-Steuerelement ist für den Zugriff auf Access-Datenbanken gedacht. Auch wenn eine Access-Datei mit der Endung `MDB` eigentlich heute immer seltener zum Einsatz kommt, ist das Steuerelement dennoch hier und da notwendig. Das einzige wichtige Attribut ist `DataFile`.

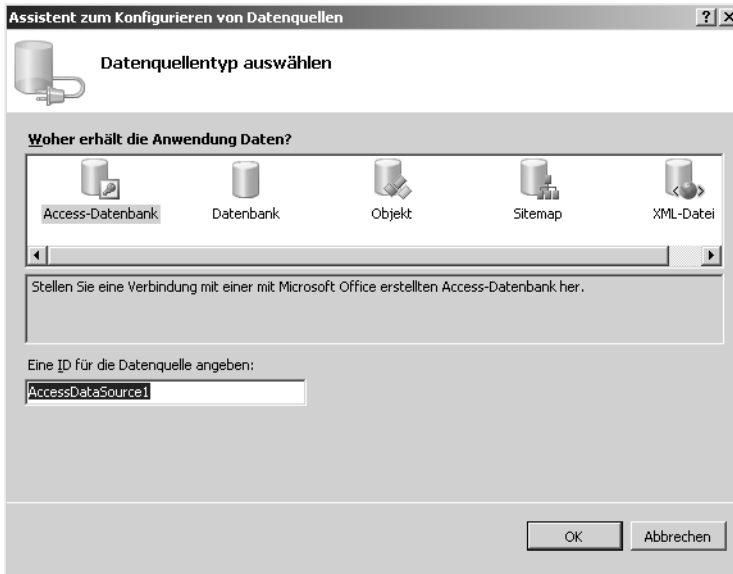
```
<asp:AccessDataSource DataFile="~/App_Data/AspNetDB.mdb" ID="AccessDataSource1" runat="server"
    SelectCommand="SELECT DISTINCT * FROM [vw_aspnet_Users]">
</asp:AccessDataSource>
```

**Listing 7.16** `AccessDataSource` in Minimalkonfiguration

**HINWEIS**

Der häufigste Fehler, der im Zusammenhang mit MDB-Dateien auftritt, beruht auf fehlenden Rechten. Im Verzeichnis, in dem sich die MDB-Datei befindet, benötigt der Web-Benutzer Lese- und Schreibrechte. Da ASPX-Seiten meist mit dem ASP.NET-Account laufen und nicht unter dem Benutzerkonto des jeweiligen Benutzers, fehlen diese Rechte oftmals. Am einfachsten legen Sie die MDB-Dateien immer in das *app\_data*-Verzeichnis. Damit ist auch gleichzeitig der Schutz gegen einen unbefugten Download dieser Datei gewährleistet.

Die Vorgehensweise ist analog zur Verwendung einer *SQLDataSource*, nur wird nun *Access-Datenbank* ausgewählt.



**Abbildung 7.20** Access-Datenbank einbinden

Damit werden auch die SQL-Kommandos für *Update*, *Insert* und *Delete* sauber erzeugt.

Um dem Benutzer die Möglichkeit einzuräumen, die Daten zu verändern, muss beim erzeugten *GridView*-Steuerelement noch per *Aufgabenmenü* die Checkbox *bearbeiten aktivieren* aktiviert werden.

Achten Sie darauf, das Attribut *DataKeyNames* zu setzen. Die SQL-Kommandos setzen darauf auf. Auf diese Weise werden beispielsweise bei einem *Update*-Vorgang geänderte Datenzeilen erkannt und nur diese in die Datenbank zurückgeschrieben.

```
<asp:GridView AutoGenerateColumns="False" DataKeyNames="UID" DataSourceID="AccessDataSource1"
  EmptyDataText="There are no data records to display." ID="GridView1" runat="server">
  <Columns>
    <asp:CommandField ShowEditButton="True">
  </asp:CommandField>
    <asp:BoundField DataField="UID" HeaderText="UID" ReadOnly="True" SortExpression="UID">
  </asp:BoundField>
    <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name">
  </asp:BoundField>
    <asp:BoundField DataField="Email" HeaderText="Email" SortExpression="Email">
  </asp:BoundField>
  </Columns>
</asp:GridView>
```

**Listing 7.17** Lesen und Schreiben von Access-Daten

```

    </Columns>
  </asp:GridView>
  <asp:AccessDataSource ID="AccessDataSource1" runat="server" DataFile="~/App_Data/devtrain.mdb"
    DeleteCommand="DELETE FROM [Benutzer] WHERE [UID] = ?" InsertCommand="INSERT INTO [Benutzer]
    ([UID], [Name], [Email]) VALUES (?, ?, ?)"
    SelectCommand="SELECT * FROM [Benutzer]" UpdateCommand="UPDATE [Benutzer] SET [Name] = ?, [Email]
    = ? WHERE [UID] = ?">
    <DeleteParameters>
      <asp:Parameter Name="UID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
      <asp:Parameter Name="Name" Type="String" />
      <asp:Parameter Name="Email" Type="String" />
      <asp:Parameter Name="UID" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
      <asp:Parameter Name="UID" Type="Int32" />
      <asp:Parameter Name="Name" Type="String" />
      <asp:Parameter Name="Email" Type="String" />
    </InsertParameters>
  </asp:AccessDataSource>

```

**Listing 7.17** Lesen und Schreiben von Access-Daten (Fortsetzung)

Da bei Access die Parameter nicht benannt sind, muss im Bereich *UpdateParameters* die Reihenfolge der Parameter identisch zum Update-Kommando sein. Der letzte Parameter wird dann in der *Where*-Bedingung verwendet.

## ObjectDataSource

Das *ObjectDataSource*-Steuerelement dient dazu, Geschäftsobjekte an die visuellen Daten-Steuerelemente zu binden. Oft werden eigene Daten-Schichten entwickelt und genau die können dann auch wieder »codeless« an die Webserver-Steuerelemente gebunden werden. Als Data Layer kann ein streng typisiertes DataSet als XSD-Datei im *app\_code*-Verzeichnis verwendet werden.

```

Imports Microsoft.VisualBasic
Imports System.Data
Imports System.Data.SqlClient
Imports System.Configuration
Public Class myDAL
  Public Function getData() As sqldatareader
    Dim con As New _
    SqlConnection(ConfigurationManager.ConnectionStrings("AppConnectionString1").ConnectionString)
    Dim cmd As New SqlCommand("SELECT * FROM Products", con)
    con.Open()
    Dim dtr As SqlDataReader = _
    cmd.ExecuteReader(CommandBehavior.CloseConnection)
    Return dtr
  End Function
End Class

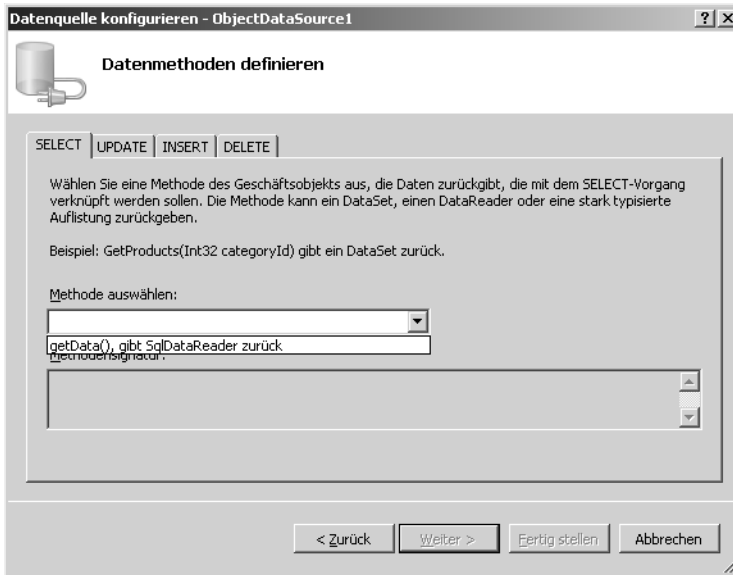
```

**Listing 7.18** Das Geschäftsobjekt

Es bestehen mehrere Daten-Format-Möglichkeiten, um zwischen Ihrem Daten-Objekt und dem *ObjectDataSource*-Steuerelement die Daten auszutauschen. Das wird häufig über eigen erstellte Datenobjekte (als

Klassen oder Struct) oder aber auch per *DataSet* geschehen. Beim Lesen ist auch ein *DataReader* einsetzbar. Ein solches minimalistisches Geschäftsobjekt wird im Folgenden als Beispiel verwendet.

Diesen Code legt man in einer VB-Datei im *app\_code*-Verzeichnis ab. Als Nächstes wird ein *ObjectDataSource*-Steuerelement in die ASPX-Seite gezogen. Im Kontextmenü *ObjectDataSource-Aufgaben* wird die Datenquelle konfiguriert. Dabei verbindet man in der DropDown-Liste diese mit der Klasse *myDAL*. Achten Sie darauf, im Dialog *ein Geschäftsobjekt auswählen* die Option *nur Datenkomponenten anzeigen* zu deaktivieren. Im nächsten Dialog werden die Funktionen aus dem Daten-Objekt *myDAL* den Methoden zugewiesen.



**Abbildung 7.21** Assistent zum Konfigurieren des *ObjectDataSource*-Steuerelements

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod="getData"
    TypeName="myDAL">
</asp:ObjectDataSource>
```

**Listing 7.19** *ObjectDataSource*-Steuerelement

Der Rest ist wie gehabt: Fügen Sie ein *GridView*-Steuerelement ein, verbinden Sie es mit der *SQLDataSource*, und Sie sind fertig.

Die Übergabe von Parametern setzt voraus, dass die Funktionen in der Datenzugriffskomponente entsprechende Parameter anbieten. In der Praxis wird man meist mit überladenen Versionen der Funktion arbeiten, mit und ohne Parameter.

Etwas aufwändiger wird es, wenn Paging unterstützt werden soll. Dabei ist gerade das Blättern in großen Datenmengen ein hervorragendes Szenario für den Einsatz eines Datenzugriffs-Objektes. Die Daten werden dann innerhalb der Datenschicht in den Cache geschrieben bzw. wieder von dort gelesen und nur die letztendlich anzuzeigenden Daten an das *ObjectDataSource*-Steuerelement übertragen.

Im folgenden Beispiel wird ein solcher Zugriff auf das Dateisystem mit eingebauter Volltextsuche realisiert, um so die komplette Beispieldatenbank zu diesem Buch im Webbrowser durchsuchen zu können. Im ersten Schritt wird eine Klasse erstellt, die als Objektcontainer für eine Datei dienen soll. Da diese Anwendung im Browser läuft, werden die Eigenschaften *URL* und *Dateiname* benötigt.

```

Imports Microsoft.VisualBasic
Public Class DateiObjekt
    Private _Datei As String
    Public Property Datei() As String
        Get
            Return _Datei
        End Get
        Set(ByVal Value As String)
            _Datei = Value
        End Set
    End Property
    Private _URL As String
    Public Property URL() As String
        Get
            Return _URL
        End Get
        Set(ByVal Value As String)
            URL = Value
        End Set
    End Property
    Public Sub New()
    End Sub
    Public Sub New(ByVal Datei As String, ByVal URL As String)
        Me.Datei = Datei
        Me.URL = URL
    End Sub
End Class

```

Listing 7.20 Das Datei-Objekt

In der ASPX-Seite werden eine *ObjectDataSource* und ein verbundenes *GridView*-Steuerelement platziert. Dazu wird das Paging im *Aufgabenmenü* aktiviert.

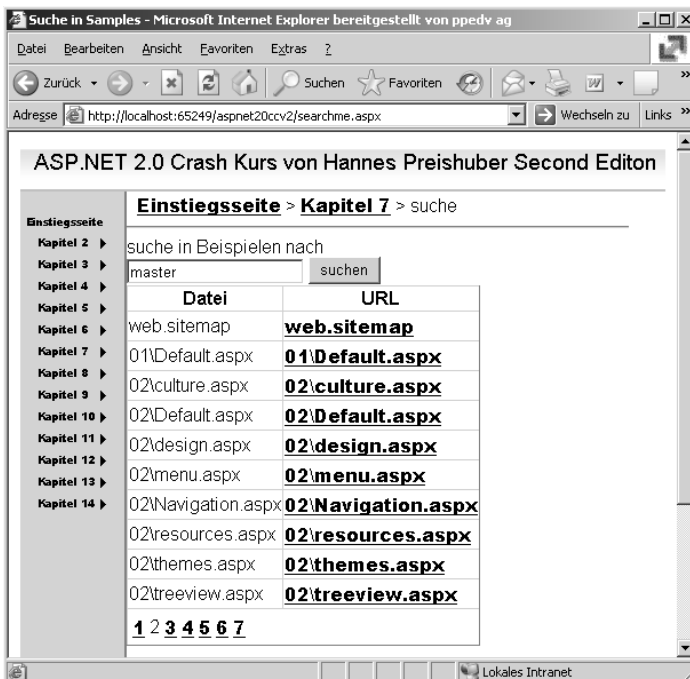


Abbildung 7.22 Suchen mit einem Datenzugriffs-Objekt

Einige zusätzliche Attribute sind nötig, um die Funktionen bzw. Parameter im Datenzugriffsobjekt zuzuordnen. Mit dem Attribut *SelectCountMethod* wird der Name der Funktion angegeben, die die Anzahl der Objekte zurückliefert. Für das Paging wird dann noch der *StartRowIndexParameterName* und *MaximumRowsParameterName* benötigt.

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    SelectMethod="GetResults" TypeName="FileDAL"
    StartRowIndexParameterName="startIndex" MaximumRowsParameterName="maxRows"
    SelectCountMethod="GetNumberOfFiles" >
  <SelectParameters>
    <asp:ControlParameter ControlID="txtSuchen" Name="SuchBegriff" PropertyName="Text"
      Type="String" ConvertEmptyStringToNull=true/>
  </SelectParameters>
</asp:ObjectDataSource>
```

**Listing 7.21** Das ObjectDataSource-Steuerelement wird an das Datenobjekt gebunden

Das folgende etwas umfangreichere Code-Schnipsel beinhaltet die komplette Funktionalität der Datenzugriffsklasse. Die Ergebnismenge pro Suchbegriff wird in den Cache geschrieben, um hauptsächlich das Paging zu beschleunigen. Das Beispiel beinhaltet einige der neuen VB 2005 Funktionen wie *Using*, den *My*-Namensraum oder *Generics*.

Auf eine ausführliche Beschreibung wird hier verzichtet.

```
Imports System.IO
Imports System.Configuration
Imports System.Collections.Generic
Public Class FileDAL
  Public Sub New()
  End Sub
  Private _itemcount As Integer = 0
  Public Function GetResults(ByVal SuchBegriff As String) As List(Of DateiObjekt)
    Dim offset As String = HttpContext.Current.Server.MapPath(".") + "\"
    Dim FO As List(Of DateiObjekt) = New List(Of DateiObjekt)
    If Not IsNothing(SuchBegriff) Then
      If IsNothing(HttpContext.Current.Cache(SuchBegriff)) Then
        Dim datei As String
        For Each datei In My.Computer.FileSystem.GetFiles(HttpContext.Current.Server.MapPath("."), _
          FileIO.SearchOption.SearchAllSubDirectories)
          Using sr As New StreamReader(datei)
            Dim line As String
            While (sr.Peek > -1)
              line = sr.ReadLine()
              If line.IndexOf(SuchBegriff, StringComparison.OrdinalIgnoreCase) > 0 Then
                FO.Add(New DateiObjekt(datei.Replace(offset, ""), datei.Replace(offset, "")))
                _itemcount += 1
              Exit While
            End If
          End While
        End Using
      Next
      HttpContext.Current.Cache.Insert(SuchBegriff, FO, Nothing, _
        System.Web.Caching.Cache.NoAbsoluteExpiration, _
        New TimeSpan(0, 30, 0))
    Else
```

**Listing 7.22** 50 Zeilen Code - das komplette Datenzugriffsobjekt

```

        FO = CType(HttpContext.Current.Cache(SuchBegriff), List(Of DateiObjekt))
        _itemcount = FO.Count
    End If
End If
Return FO
End Function

Public Function GetResults(ByVal maxRows As Integer, ByVal startIndex As Integer, _
    ByVal Suchbegriff As String) As IList(Of DateiObjekt)
    Dim pagedFO As IList(Of DateiObjekt) = New List(Of DateiObjekt)
    If IsNothing(Suchbegriff) = False Then
        Dim FO As IList(Of DateiObjekt) = Me.GetResults(Suchbegriff)
        Dim i As Integer
        For i = startIndex To IIf((startIndex + maxRows < _itemcount), (startIndex + maxRows), _
            _itemcount) Step i + 1
            pagedFO.Add(FO(i))
        Next
    End If
    Return pagedFO
End Function

Public Function GetNumberOfFiles(ByVal SuchBegriff As String) As Integer
    Return _itemcount
End Function

End Class

```

**Listing 7.22** 50 Zeilen Code - das komplette Datenzugriffsobjekt (Fortsetzung)

## XMLDataSource

Mit dem *XmlDataSource*-Steuerelement können XML-basierte und damit hierarchische Daten angesprochen werden. Drei Attribute sind besonders wichtig und können auch per *Aufgabenmenü* eingestellt werden:

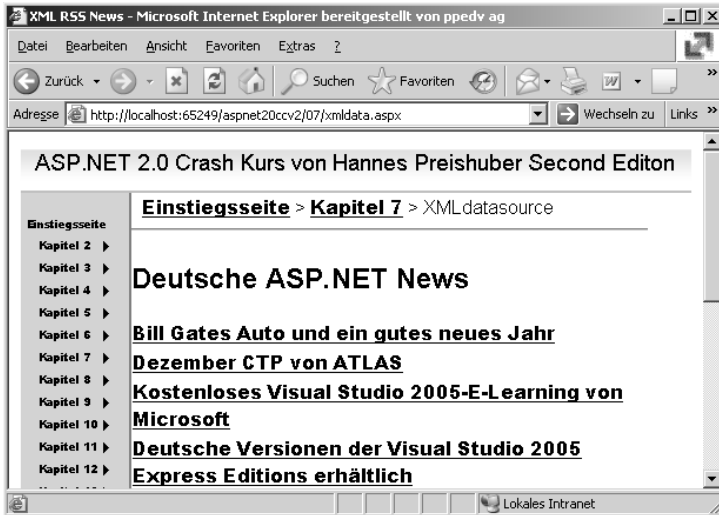
Mit *Datendatei* wird die XML-Datei angegeben, die die Datenquelle darstellen soll. Mit *Transformationsdatei* kann eine XSL-Datei geladen werden, die die Daten transformiert. Mit *XPath-Ausdruck* kann eine Art Abfrage über die Daten stattfinden. Die Attribute können natürlich auch zur Laufzeit gesetzt werden.

Normalerweise dient dieses Steuerelement als Datenquelle für ein TreeView oder ein Menü-Steuerelement. Aber es kann auch für übliche DataControls verwendet werden.

Im folgenden Beispiel wird der RSS-Feed des Magazins ASP.NET Professional ausgelesen und in die eigene Webseite eingebaut. Damit sind die ASP.NET-News auch dort zu lesen. Zur Anzeige wird ein *DataList*-Steuerelement verwendet, das im nächsten Kapitel beschrieben wird.

Das *XmlDataSource*-Steuerelement erhält mit dem Attribut *Datendatei* eine Referenz auf den URL. Die News-Einträge befinden sich im *Item*-Element des RSS-Feeds. Mit dem *XPath-Ausdruck* wird eine entsprechende Adressierung der Knoten vorgenommen. Der besondere Clou ist, dass nicht bei jedem Seitenabruf der RSS-Feed erneuert wird. Dazu wird *EnableCaching* auf *True* gesetzt und die Lebensdauer mit *Cacheduration* auf 600 Sekunden eingestellt.





**Abbildung 7.23** RSS-News per *DataSource*-Steuerelement einbinden

Im *DataList*-Steuerelement wird per *XPath-Datenbindung* auf die Unterelemente des *Item*-Elements zugegriffen. Die Bindungssyntax lautet dazu `<%#XPath("feld")%>`.

```
<asp:XmlDataSource DataFile="http://www.aspnet-professional.de/newsrss.aspx" ID="XmlDataSource1"
    EnableCaching="true"
    runat="server" XPath="rss/channel/item" CacheDuration="600">
</asp:XmlDataSource>
<asp:datalist id="DataList1" runat="server" datasourceid="XmlDataSource1">
  <itemtemplate>
    <a href="<%= XPath("link") %>" target="_blank"><%=XPath("title") %></a>
    <br />
  </itemtemplate>
</asp:datalist>
```

Mehr ist wirklich nicht zu tun, und Sie können jedes Blog oder jeden RSS-Feed in Websites einbinden. Den XML-Quelltext des RSS-Feeds finden Sie unter <http://www.aspnet-professional.de/newsrss.aspx>.

## SiteMapDataSource

Das *SiteMapDataSource*-Steuerelement dient zum Zugriff auf SiteMap-Datenquellen. Das ist grundsätzlich bereits in Kapitel 2 beschrieben worden. Damit werden Seitennavigations-Informationen verwaltet.

Wenn Sie keinen weiteren Parameter angeben, wird auf die Datei *web.sitemap* verwiesen.

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

## Neues in ADO.NET 2.0

Trotz der schönen neuen »codefreien« Welt in ASP.NET 2.0 braucht man auch direkten Zugriff auf Daten, ohne die *DataControls* verwenden zu müssen. Das folgende Beispiel zeigt, wie man Daten aus einem SQL Server liest und darstellt.

Dazu benötigt man Funktionen aus den beiden Namensräumen *System.Data* und *System.Data.SqlClient*, die entsprechend eingebunden werden müssen. Als Container wird anschließend ein *DataSet* instanziiert. Als Nächstes wird eine Verbindung zur Datenbank aufgebaut. Die nächste Zeile erzeugt ein *DataAdapter*-Objekt, das definiert, was woher gelesen werden soll. Per *Fill*-Methode fließen die Daten aus der Datenbank ins *DataSet*. Da die Verbindung danach nicht mehr benötigt wird, muss sie per *Close* geschlossen werden.

Dem *GridView*-Steuerelement wird dann dieses *DataSet* als Datenquelle zugewiesen. Erst mit der Methode *DataBind* wird die Bindung aktiviert.

Zu ASP.NET 1.x gibt es zwei nennenswerte Unterschiede. Bisher wurde ein *DataGrid* verwendet, die Verbindungszeichenfolge wird nunmehr aus dem neuen Bereich *ConnectionStrings* aus der *web.config* eingelesen. Dabei hilft das *ConfigurationManager*-Objekt.

```
<%@ Page Language="VB" MasterPageFile="~/all.master" Title="Untitled Page" %>
<%@ Import Namespace="system.data"%>
<%@ Import Namespace="system.data.sqlclient"%>
<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim DS As New DataSet
        Dim conn As New _

SqlConnection(ConfigurationManager.ConnectionStrings("AppConnectionString1").ConnectionString)
        Dim DA As New SqlDataAdapter("Select * from products", conn)
        DA.Fill(DS)
conn.Close()
        GridView1.DataSource = DS
        GridView1.DataBind()
    End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <asp:GridView ID="GridView1" Runat="server">
    </asp:GridView>
</asp:Content>
```

**Listing 7.23** Daten mit ADO.NET als Tabelle darstellen

## Datenbindung

Von Datenbindung spricht man, wenn Steuerelemente im Formular direkt an Tabellenfelder gebunden werden. Ganz so direkt funktioniert das mit dem unter ADO.NET üblichen, verbindungslosen Verfahren natürlich nicht – aber sinngemäß ist es dennoch richtig formuliert. Dabei ist »Datenbindung« nur mit den Listen-Steuerelementen möglich, und auch wiederum nur mit denen, die über Templates verfügen. In einem Template werden dann Webserver-Steuerelemente wie eine *TextBox* oder ein *Label* eingebettet und an Daten gebunden.

Die Syntax lautet für alle ASP.NET-Versionen `<%# ...%>` – aber auch hier hat ASP.NET 2.0 einige Vereinfachungen und neue Funktionen:

### Eval

Zum Ausgeben eines einzelnen Feldes gibt es die Hilfsfunktion *Eval*.

```
<%#Eval("FieldName")%>
```

Mit *Eval* greifen Sie auf den aktuellen Datensatz des Datenelements zu und formatieren gleichzeitig die Ausgabe. Wenn spezielle Formatausgaben wie z.B. ein kurz formatiertes, deutsches Datum nötig sind, gibt es eine Überladung mit einem zweiten Parameter, der die Formatierung steuert.

```
<%# Eval("Datum", "{0:d}") %>
```

## Multiple Active Result Sets (MARS)

In den früheren .NET Versionen konnte pro aktiver SQL Server Verbindung nur ein DataReader geöffnet werden. Mit dem DataReader ist es möglich Datensätze einzeln und nur vorwärts zu lesen. Damit ist gerade für die Verarbeitung von großen Datenmengen der DataReader dem DataSet vorzuziehen, da sofort mit den eintreffenden Daten gearbeitet werden kann. Wenn nun z.B. über Relationen verbundene Daten gleichzeitig gelesen werden sollen, muss dafür pro DataReader eine eigene Verbindung (Connection) geöffnet werden und Connections sind sehr rare Ressourcen. Deshalb gibt es seit SQL Server 2005 die Möglichkeit eine Connection mehrfach zu nutzen. Dazu muss nur das Attribut *MultipleActiveResultSets=true* in den Connection String eingefügt werden. Im folgenden Beispiel wird das einfach per Zeichenkettenzusammensetzung aus dem Connection String aus der web.config erledigt.

Das folgende Beispiel soll eine HTML-Liste mit *OptionGroup*-Elementen gruppieren und so das Bestelldatum und die dazugehörigen Stückzahlen und Preise anzeigen. Das ist mit keinem ASP.NET-Steuerelement möglich. Insofern wird hier einfach HTML-Code zusammengesetzt und letztendlich an die Stelle eines Literal-Steuerelements gesetzt.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim html As New StringBuilder
    html.Append("<select size=20>")
    Dim conn As New
SqlConnection(ConfigurationManager.ConnectionStrings("northwindconnectionstring").ConnectionString _
+ ";MultipleActiveResultSets=True")
    Dim cmd As SqlCommand = New SqlCommand("SELECT * FROM orders", conn)
    conn.Open()
    Dim rdOrders As SqlDataReader = cmd.ExecuteReader()
    While rdOrders.Read
        html.Append("<optgroup label=")
        html.Append(rdOrders("OrderDate"))
        html.Append(">")
        Dim cmdDet As SqlCommand = New SqlCommand(
            "SELECT * FROM [order details] where orderid=@orderid", conn)
        Dim par As New SqlParameter("@orderid", SqlDbType.Int)
        par.Value = rdOrders("orderid")
        cmdDet.Parameters.Add(par)
        Dim rdDetails As SqlDataReader = cmdDet.ExecuteReader()
        While rdDetails.Read
            html.Append("<option>")
            html.Append(rdDetails("quantity").ToString + ":" + rdDetails("unitprice").ToString)
            html.Append("</option>")
        End While
        html.Append("<optgroup>")
    End While
    html.Append("</select>")
    rdOrders.Close()
    conn.Close()
    Literal1.Text = html.ToString
End Sub
```

**Listing 7.24** MARS-Beispiel

Aus Performancegründen sollte für mehrfache Zeichenkettenoperationen immer ein *StringBuilder*-Objekt verwendet werden. Weiterhin sollen Connections möglichst kurz offen bleiben. Durch Einsatz von SQL-Parametern wird die Anwendung gegen SQL-Injection gesichert.

Zur Laufzeit sieht dann der Benutzer folgendes Ergebnis:

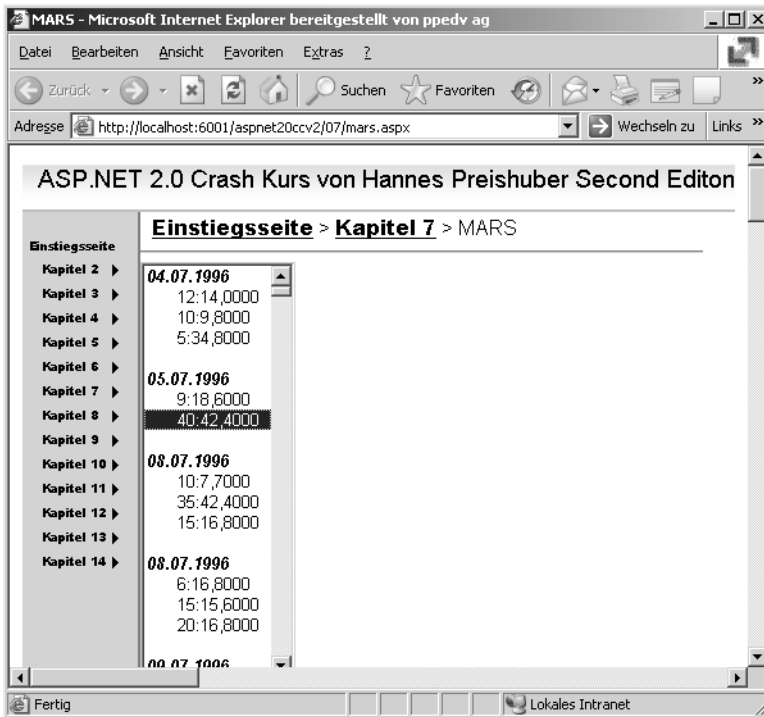


Abbildung 7.24 SELECT-Listenelement mit OPTGROUP

## Bind

Mit *Bind* wird die Eval-Funktion zur Zweigege-Datenbindung aufgewertet. Diese Funktion kommt z.B. in den Templates des *GridView*- und *FormView*-Steuerelements zum Einsatz:

```
<asp:TextBox Text='<%=# Bind("Name") %>' Runat="server" ID="TextBox1"></asp:TextBox>
```

## XPath

Wenn Sie das *XmlDataSource*-Steuerelement verwenden, wird zur Datenbindung auf das einzelne Feld ein Abfrageausdruck nach XPath-Syntax verwendet:

```
<%=#XPath("title") %>
```

## Kapitel 8

# DataView-Steuererelemente

### **In diesem Kapitel:**

GridView	214
DetailsView	232
FormView	239
DataGrid	241
DataList	242
Repeater	244

Die bisherigen Beispiele aus Kapitel 7 verwendeten meist das *GridView*-Steuerelement. Daneben gibt es auch andere Listen-Steuerelemente, die an Daten gebunden werden können und ganz spezifische Einsatzbereiche haben. Neben den einfachen Listen wie *DropDown* oder *CheckBoxList* sind das die neuen Steuerelemente *GridView*, *DetailsView* und *FormView*.

Im letzten Teil dieses Kapitels werden noch kurz die Änderungen an den aus ASP.NET 1.x bekannten Steuerelementen *DataGrid*, *DataList* und *Repeater* beschrieben.

Alle diese Steuerelemente lassen sich als reichhaltige Steuerelemente (*Rich Controls*) bezeichnen und bieten eine unglaubliche Fülle von Konfigurationsmöglichkeiten. Aber auch einige knifflige Anforderungen lassen sich damit lösen.

## GridView

Das *GridView*-Steuerelement ist eindeutig als Nachfolger des *DataGrid* konzipiert. Die Eigenschaften und Methoden sind meist identisch benannt und haben dann auch die gleiche Funktion. Allerdings ist das *GridView*-Steuerelement viel mächtiger ausgestattet. Der wesentlichste Unterschied ist aber, dass kein Code nötig ist, um die gängigsten Kodierjobs zu erledigen. In neueren Applikationen sollten Sie ausschließlich das *GridView* verwenden.

Dabei hilft eines der neuen *DataSource*-Controls, an das das *GridView* gebunden werden kann. Außerdem lassen sich die Funktionen *Blättern*, *Sortieren* und *Daten ändern* ohne Code durchführen. Um die einzelnen Bereiche im Detail gestalten zu können, werden umfangreiche Templates unterstützt. Jede Spalte kann für jeden Modus ein eigenes Template beinhalten.

Das Einzige, was man vielleicht beim *GridView* vermissen könnte, ist die fehlende *Insert*-Funktion. Es können also keine neuen Datensätze damit angelegt werden. Dazu gibt es andere Steuerelemente.

## GridView-Attribute

Da es sich beim *GridView*-Steuerelement um ein sehr mächtiges Steuerelement handelt, gibt es folglich auch eine erhebliche Menge an Einstellungsmöglichkeiten. Dies kann mit eigenen Elementen für Templates geschehen, aber auch durch die einfacheren Attribute.

### Allgemeine Attribute

Mit wenigen Attributen lässt sich schon eine umfangreiche Funktionalität erreichen. Wichtig ist vor allem, daß das *GridView* per *DataSourceID* an die Datenquelle gebunden wird. Weiterhin werden zumindest das *Attribut runat=Server*, eine *ID* und Spalten zum Anzeigen benötigt. Wenn der Benutzer zur Laufzeit im Browser nichts sieht, sind entweder keine Daten zum Anzeigen vorhanden oder es wurde vergessene Spalten zu erzeugen.

```
<asp:GridView ID="GridView1" Runat="server"
  AccessKey="A"
  AllowPaging="True"
  AllowSorting="True"
  AutoGenerateColumns="False"
  BorderWidth="1px"
```

Listing 8.1 GridView

```

BackColor="White"
CellPadding="4"
BorderStyle="None"
BorderColor="#CC9966"
DataSourceID="SqlDataSource1">

```

**Listing 8.1** GridView (Fortsetzung)

Im Folgenden werden einige der wichtigsten Attribute des *GridView*-Steuerelements beschrieben. Die Attribute finden sich teils auch in anderen Steuerelementen wieder.

GridView-Attribut	Bedeutung
<i>AllowPaging</i>	Wenn dieses Attribut aktiviert ( <i>true</i> ) ist, wird das automatische Paging aktiviert.
<i>AllowSorting</i>	Mit diesem Attribut aktiviert man die Möglichkeit zum Sortieren der Daten.
<i>AutoGenerateColumns</i>	Ist dieses Attribut aktiviert, werden automatisch die Spalten zur Laufzeit erzeugt. Standardwert ist <i>false</i> .
<i>AutoGenerateDeleteButton</i>	Ist dieses Attribut aktiviert, wird zur Laufzeit automatisch ein Delete-Link oder -Button erzeugt. Damit wird auch die Löschfunktion aktiviert. Achten Sie darauf, dass auch das SQL-Kommando für das Löschen im <i>DataSource</i> -Steuerelement belegt sein muss.
<i>AutoGenerateEditButton</i>	Ist dieses Attribut aktiviert, wird zur Laufzeit automatisch ein Edit-Link oder -Button erzeugt. Damit wird auch die <i>Edit</i> -Funktion aktiviert.
<i>AutoGenerateSelectButton</i>	Ist dieses Attribut aktiviert, wird zur Laufzeit automatisch ein Select-Link oder -Button erzeugt. Damit kann der Benutzer einen Datensatz auswählen.
<i>BackColor</i>	Setzt die Hintergrundfarbe.
<i>BackColorImageURL</i>	Setzt das Hintergrundbild. Es wird im Browser ein <i>Style</i> -Attribut für das Table-Element verwendet.
<i>BorderColor</i>	Setzt die Rahmenfarbe. Das setzt voraus, dass die Rahmenstärke ( <i>BorderWidth</i> ) größer als 0 ist. Mit <i>BorderStyle</i> kann dann noch die Art des Rahmens definiert werden
<i>Caption</i>	Setzt die Überschrift über das gesamte GridView-Steuerelement.
<i>CaptionAlign</i>	Positioniert die Überschrift. Mögliche Werte: <i>Bottom</i> , <i>Left</i> , <i>Right</i> , <i>Top</i> , <i>NotSet</i> .
<i>CellPadding</i>	Definiert den Abstand zwischen dem Inhalt einer Zelle und dem Rahmen in Pixel.
<i>CellSpacing</i>	Definiert den Abstand zwischen den Zellen in Pixel.
<i>DataKeyNames</i>	Damit können die <i>Schlüssel</i> -Werte als Array gesetzt werden, die die anschließenden <i>Select</i> -, <i>Update</i> - oder <i>Delete</i> -Funktionen benötigen. Das sind die Primärschlüssel. Die Werte werden durch Kommata getrennt.
<i>DataMember</i>	Wenn das <i>DataSet</i> , aus dem die Daten kommen, mehr als eine Tabelle enthält, muss die anzuzeigende Tabelle mit <i>DataMember</i> spezifiziert werden.
<i>DataSource</i>	Der Name des <i>DataSource</i> -Objekts, aus dem die Daten kommen.
<i>DataSourceID</i>	Der Name des <i>DataSource</i> Web Controls, aus dem die Daten kommen.
<i>EditIndex</i>	Liest oder schreibt den Index der Reihe, die zu editieren ist.
<i>EmptyDataText</i>	Falls keine Daten anzuzeigen sind, wird diese Zeichenkette dargestellt.
<i>EnableSortingAndPagingCallbacks</i>	Liest oder schreibt einen booleschen Wert. Wenn dieser <i>true</i> ist, wird im Browser ein JScript erzeugt, um <i>Callbacks</i> statt <i>Postbacks</i> durchzuführen.

**Tabelle 8.1** Auszug aus den Attributen des *GridView*-Steuerelements

GridView-Attribut	Bedeutung
<i>GridLines</i>	Mit <i>GridLines</i> wird definiert, ob Linien zwischen den Zellen sind. Mögliche Werte sind <i>Both</i> , <i>Horizontal</i> , <i>Vertical</i> und <i>None</i> . Der Standardwert ist <i>Both</i> , falls nichts angegeben ist.
<i>RowHeaderColumn</i>	Mit diesem Attribut definiert man den Spaltennamen, der eine Reihe identifiziert. Dies wird im normalen Browser nicht verwendet und ist für Lesegeräte gedacht.
<i>SelectedIndex</i>	Setzt oder liest den Index des selektierten Datensatzes.
<i>ShowFooter</i>	Ein boolescher Wert, mit dem die Anzeige der Fußzeile gesteuert wird.
<i>ShowHeader</i>	Ein boolescher Wert, mit dem die Anzeige der Kopfzeile gesteuert wird.
<i>UseAccessibleHeader</i>	Ein boolescher Wert, der angibt, ob die Kopfzeile mit <code>&lt;TH&gt;</code> statt <code>&lt;TD&gt;</code> im Browser angegeben wird.
<i>Visible</i>	Ein boolescher Wert, mit dem die Anzeige des GridView im Browser unterbunden werden kann.
<i>Width</i>	Die Breite des GridView Steuerelements in Pixel.

**Tabelle 8.1** Auszug aus den Attributen des *GridView*-Steuerelements (Fortsetzung)

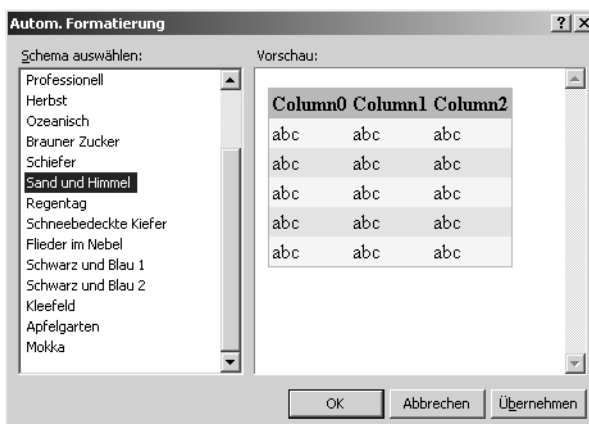
## RowStyle-Element

Die Darstellung der Reihen wird mithilfe von Unterelementen des *GridView*-Steuerelements definiert. Die einzelnen Elemente werden über Attribute mit der gewünschten Formatierung versehen. Es gibt verschiedene Typen-Reihen wie z.B. für die Fuß- oder Kopfzeile.

```
<FooterStyle ForeColor="#330099" BackColor="#FFFFCC"></FooterStyle>
<PagerStyle ForeColor="#330099" HorizontalAlign="Center" BackColor="#FFFFCC"></PagerStyle>
<HeaderStyle ForeColor="#FFFFCC" Font-Bold="True" BackColor="#990000"></HeaderStyle>
```

**Listing 8.2** Listing 8.2:Exemplarische *RowStyle*-Elemente des *GridView*-Steuerelements

Die umfangreichen Möglichkeiten werden noch durch die visuellen Assistenten des Steuerelements unterstützt. Aufgerufen wird dieser mit dem Menüpunkt *automat. Formatierung* im *Aufgaben* Menü des *GridView*-Steuerelements. Es sind umfangreiche Format-Templates vorhanden.



**Abbildung 8.1** Umfangreiche Format-Templates des GridView



Dadurch werden die Style-Elemente erzeugt und entsprechend gesetzt. Im Folgenden finden Sie eine Liste der möglichen Style-Unterelemente des *GridView*-Elements und die Beschreibung ihrer Verwendung.

Style-Typ	Verwendung
<i>AlternatingRowStyle</i>	Damit wird der Style für die abwechselnden Zeilen festgelegt.
<i>EditRowStyle</i>	Damit wird der Style einer Zeile, die sich im Edit Modus befindet, gesteuert.
<i>EmptyDataRowStyle</i>	Damit wird der Style für eine leere Reihe festgelegt, wenn keine Daten vorhanden sind.
<i>FooterStyle</i>	Damit wird der Style der Fußzeile des Steuerelements gesteuert.
<i>HeaderStyle</i>	Damit wird der Style der Spaltenüberschriften im Steuerelement gesteuert.
<i>PagerStyle</i>	Damit wird der Style der Pager-Leiste festgelegt. Die Pager-Leiste zeigt die Steuerung für das Blättern in größeren Datenmengen an.
<i>RowStyle</i>	Damit wird der Style für alle normalen Reihen festgelegt.
<i>SelectedRowStyle</i>	Damit wird der Style für eine selektierte Reihe festgelegt.

**Tabelle 8.2** Style-Template-Elemente

In den Templates wird dann durch zusätzliche Attribute, die sich auf das jeweilige *Style*-Element beziehen, das Design festgelegt.

Attribut	Verwendung
<i>BackColor</i>	Hintergrundfarbe als benannte (Beispiel: <i>red</i> ) oder RGB- Farbe (Beispiel: #FFFFCC).
<i>BorderColor</i>	Rahmenfarbe.
<i>BorderStyle</i>	Rahmenstil. Mögliche Werte sind <i>Dashed</i> , <i>Dotted</i> , <i>Double</i> , <i>Groove</i> , <i>Inset</i> , <i>None</i> , <i>NotSet</i> , <i>Outset</i> , <i>Ridge</i> , <i>Solid</i>
<i>BorderWidth</i>	Rahmenbreite in Pixel.
<i>CssClass</i>	Zu verwendende Stylesheet-Klasse.
<i>Font-Bold</i>	Soll die verwendete Schrift fett oder normal dargestellt werden? Mögliche Werte sind <i>true</i> oder <i>false</i> .
<i>Font-Italic</i>	Soll die Schrift kursiv oder normal dargestellt werden? Mögliche Werte sind <i>true</i> oder <i>false</i> .
<i>Font-Names</i>	Name der verwendeten Systemschrift zur Darstellung im GridView.
<i>Font-Overline</i>	Soll der Text überstrichen oder normal dargestellt werden? Mögliche Werte sind <i>true</i> oder <i>false</i> .
<i>Font-Size</i>	Gibt die Schriftgröße in Pixel an.
<i>Font-Strikeout</i>	Soll der Text durchgestrichen oder normal dargestellt werden? Mögliche Werte sind <i>true</i> oder <i>false</i> .
<i>Font-Underline</i>	Soll der Text unterstrichen oder normal dargestellt werden? Mögliche Werte sind <i>true</i> oder <i>false</i> .
<i>ForeColor</i>	Gibt die Farbe der Schrift an.
<i>Height</i>	Gibt die Höhe in Pixel an.
<i>HorizontalAlign</i>	Die horizontale Ausrichtung in der erzeugten Zelle. Mögliche Werte: <i>Center</i> , <i>Justify</i> , <i>Left</i> , <i>NotSet</i> , <i>Right</i>
<i>VerticalAlign</i>	Die vertikale Ausrichtung in der erzeugten Zelle. Mögliche Werte: <i>Bottom</i> , <i>Middle</i> , <i>NotSet</i> , <i>Top</i>
<i>Width</i>	Breite in Pixel.
<i>Wrap</i>	Gibt an, ob ein Text in der Zelle umgebrochen werden soll. Mögliche Parameter sind <i>true</i> oder <i>false</i> .

**Tabelle 8.3** Attribute der *RowStyle*-Elemente

Viele der Stileinstellungen lassen sich als Element oder alternativ als direktes Attribut des *GridView*-Steuerelements festlegen. Der Name des Attributes setzt sich dann aus dem Style-Typ und dem Attribut zusammen.

```
AlternatingRowStyle-BackColor="#ff3300"
```

Soweit wären die Grundlagen abgehandelt. Im nächsten Abschnitt werden die Funktionen und damit auch die Ereignisse des *GridView*-Steuerelements beschrieben.

## Blättern

Paging ist nun auch ohne Code zu realisieren. Zum Aktivieren wird im *GridView*-Steuerelement das Attribut

```
AllowPaging="True"
```

gesetzt. Das kann auch mit dem Aufgabenmenü und der Checkbox *Paging aktivieren* gemacht werden.

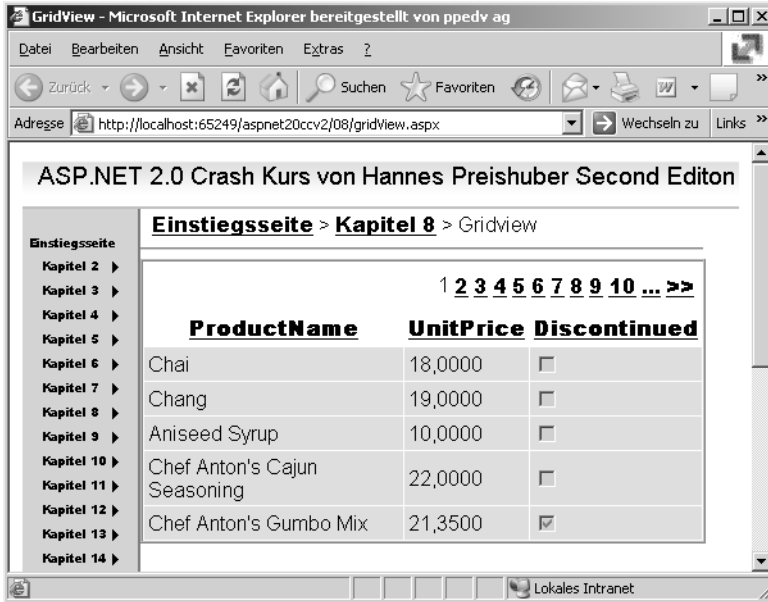
Mit weiteren Attributen wird eingestellt, wie viele Datensätze pro Seite angezeigt werden sollen (*PageSize*) oder wie geblättert werden soll (*Mode*). Selbst die Position (*Position*) der Pagersteuerung (oben oder unten) ist einstellbar.

In einem eigenen Bereich *PagerSettings* werden dazu die Parameter gesetzt. Alternativ können diese, wie in Tabelle 8.4 angegeben, auch direkt als Attribute des *GridView*-Steuerelements eingegeben werden. Im Element *PagerStyle* wird schließlich noch das Design festgelegt.

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1" BorderWidth="2px"
BackColor="White" GridLines="None" CellPadding="3" CellSpacing="1" BorderStyle="Ridge"
BorderColor="White" AllowSorting="True" EnableViewState="False"
AllowPaging="True" PageSize="5">
  <PagerSettings Mode="NumericFirstLast" Position="Top" ></PagerSettings>
  <PagerStyle ForeColor="Black" HorizontalAlign="Right" BackColor="white">
</PagerStyle>
  <RowStyle ForeColor="Black" BackColor="#DEDFDE"></RowStyle>
</asp:GridView>
```

**Listing 8.3** Code in der ASPX-Seite von GridView mit Paging-Funktion

Es ist für die volle Funktion der Seite kein weiterer Code erforderlich.



**Abbildung 8.2** GridView mit Paging-Funktion

Im Unterschied zum *DataGrid*-Steuerelement funktioniert Paging nun auch mit abgeschalteten Viewstate, was die übertragene Datenmenge erheblich reduziert.

### PagerSettings

In der folgenden Tabelle werden die wichtigsten Attribute des *GridView*-Steuerelements erklärt, die das Blättern beeinflussen. Alternativ können die meisten davon auch als Attribut des Unterelements `<PagerSettings>` verwendet werden.

GridView-Attribut	Verwendung
<i>PageIndex</i>	Setzt oder liest die anzuzeigende Seite der Daten.
<i>PagerSettings-FirstPageImageUrl</i>	Gibt den URL des Bildes, der zum Anzeigen des Pagers zum Sprung auf die erste Seite verwendet wird, als String an.
<i>PagerSettings-FirstPageText</i>	Der Text für den Pager zum Navigieren zur ersten Ergebnisseite.
<i>PagerSettings-LastPageImageUrl</i>	Gibt den URL des Bildes, der zum Anzeigen des Pagers verwendet wird, als String an.
<i>PagerSettings-LastPageText</i>	Der Text für den Pager zum Navigieren zur letzten Ergebnisseite.
<i>PagerSettings-Mode</i>	Damit wird einer der vier PagerButtons-Modi gesetzt. Die Collection beinhaltet <i>NextPrevious</i> , <i>NextPreviousFirstLast</i> , <i>Numeric</i> und <i>NumericFirstLast</i> .
<i>PagerSettings-NextPageImageUrl</i>	Gibt den URL des Bildes, der zum Anzeigen des Pagers verwendet wird, als String an.
<i>PagerSettings-NextPageText</i>	Der Text für den Pager zum Navigieren auf die erste Ergebnisseite.
<i>PagerSettings-PageButtonCount</i>	Wenn der <i>Numeric</i> -Modus (siehe <i>PagerSettings.Mode</i> ) gesetzt ist, kann mit diesem Attribut die Anzahl der angezeigten Pager-Links gesteuert werden.

**Tabelle 8.4** Attribute im GridView-Steuerelement für das Paging

GridView-Attribut	Verwendung
<i>PagerSettings-Position</i>	Gibt an, wo der Pager erscheinen soll. Die <i>PagerPosition</i> Collection beinhaltet <i>Top</i> , <i>Bottom</i> und <i>TopAndBottom</i> .
<i>PagerSettings-PreviousPageImageUrl</i>	Gibt den URL des Bildes, der zum Anzeigen des Pagers verwendet wird, als String an.
<i>PagerSettings-PreviousPageText</i>	Der Text für den Pager zum Navigieren zur ersten Ergebnisseite.
<i>PagerSettings-Visible</i>	Der boolesche Wert gibt an, ob der Pager sichtbar sein soll.
<i>PageSize</i>	Damit steuert man, wie viele Ergebnisse auf einer Seite angezeigt werden sollen.

**Tabelle 8.4** Attribute im GridView-Steuerelement für das Paging (Fortsetzung)

## PagerTemplate

Das Erscheinungsbild des Pager-Elements kann selbstverständlich auch beeinflusst werden. Dazu dient das *PagerTemplate*-Element des *GridView*-Steuerelements. Innerhalb des *PagerTemplate* können beliebige HTML- und Webserver-Steuerelemente verwendet werden. Allerdings muss der Inhalt dann auch manuell erzeugt werden. Dazu behandelt man am besten das *RowCreated*-Ereignis des *GridView*-Steuerelements oder deklariert diese in der ASPX-Seite.

Wenn die automatisch erzeugten Ereignisse verwendet werden sollen, müssen die eingesetzten Webserver-Steuerelemente, die das Paging auslösen sollen, mit definierten Parametern gefüttert werden. Dazu können übliche Webserver-Steuerelemente wie *Button* oder *Hyperlink* eingesetzt werden. Das Attribut *CommandName* definiert die grundsätzliche Aufgabe *Page*. Die eigentliche Funktionalität wird durch das Setzen des Attributs *CommandArgument* realisiert. Mögliche Werte sind *First*, *Last*, *Next* und *Prev*.

Über die Attribute *PageIndex*, *PageSize* und *PageCount* kann die Positionsbestimmung durchgeführt werden.

```
<asp:GridView AllowPaging=True Id="GridView1" >
  <PagerTemplate>
  <asp:LinkButton id="cmdPrev" text="<" CommandName="Page" CommandArgument="Prev" runat="Server"/>
  <asp:LinkButton id="cmdNext" text=">" CommandName="Page" CommandArgument="Next" runat="Server"/>
  <br>
  Index <%= GridView1.PageIndex * GridView1.PageSize %> Anzahl <%=GridView1.PageCount %>
  </PagerTemplate>
</asp:GridView>
```

**Listing 8.4** Selbst entworfenes einfaches Pager-Template

## Spalten

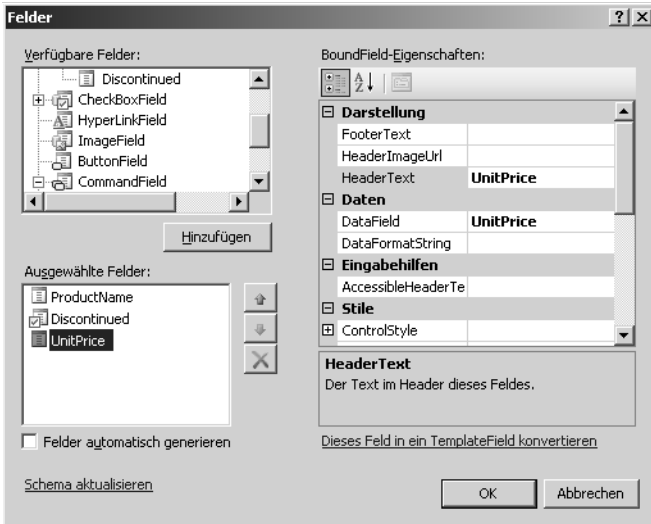
Das *GridView*-Steuerelement kann seine Spalten automatisch erzeugen. Allerdings hat man dann keinerlei Einfluss auf Layout oder Reihenfolge. Also müssen selbst erstellte datengebundene Spalten her. Diese werden auch als *BoundFields* (*gebundene Spalten*) bezeichnet. Zunächst sollte man aber das automatische Erstellen von Spalten abschalten:

```
AutoGenerateColumns="false"
```

### HINWEIS

Das automatische Generieren von Spalten dauert etwas länger, als wenn diese manuell erzeugt werden.

Am besten verwenden Sie den im Kontextmenü *Aufgaben* enthaltenen Assistenten, um Ihre ersten Bound-Fields zu erzeugen. Dazu wählen Sie den Punkt *Spalten bearbeiten*.



**Abbildung 8.3** Spalten im GridView mit dem Assistenten erstellen

Durch Auswahl in der Liste der *verfügbaren Felder* kann man im *GridView* innerhalb des *Column*-Elements pro Spalte ein Element hinzufügen. Je nach Art des Elements kann dies eine *Checkbox* oder ein *CommandField* sein. Dies erscheint dann in der Liste der *ausgewählten Felder*.

Für die reine Anzeige von Daten gibt es das *BoundField*-Element. Mit dem Attribut *DataField* wird die Zuordnung zum Datenfeld vorgenommen. Wenn die Ausgabe formatiert werden soll, kommt der Parameter *DateFormatString* zum Einsatz.

```
<Columns>
  <asp:CommandField ShowSelectButton="True" HeaderText="Kopf1" ></asp:CommandField>
  <asp:CheckBoxField HeaderText="Aktiv" DataField="Discontinued"></asp:CheckBoxField>
  <asp:BoundField HeaderText="Name" DataField="ProductName"></asp:BoundField>
  <asp:BoundField HeaderText="Preis" DataField="UnitPrice" DataFormatString="{0:c}"></asp:BoundField>
</Columns>
```

**Listing 8.5** Manuell definierte Spalten im GridView Control

Im Folgenden werden die möglichen Spaltentypen erläutert:

Spalten-Typ	Verwendung
<i>AutoGeneratedField</i>	Ist das Attribute <i>AutoGenerateColumns</i> auf <i>true</i> gesetzt, wird dieser Column-Typ verwendet, um die Spalten zu erzeugen. Anhand des Datentyps wird automatisch die Darstellung gewählt. So wird ein boolescher Wert durch eine <i>Checkbox</i> dargestellt.
<i>BoundField</i>	Mit einer <i>BoundField</i> -Spalte wird ein einfaches Datenfeld ausgegeben. Im Edit-Modus wird dieses per <i>Textbox</i> dargestellt.

**Tabelle 8.5** Typen manuell definierter Spalten

Spalten-Typ	Verwendung
<i>ButtonField</i>	Mit diesem Steuerelement wird ein Button in jeder Reihe des GridView-Steuerelements angezeigt. Mit dem Attribut <i>ButtonType</i> kann aus den Typen <i>Button</i> , <i>Link</i> oder <i>Image</i> ausgewählt werden. Weitere Attribute setzen den Text oder andere Eigenschaften.
<i>CheckBoxField</i>	Wenn die Daten vom Type <i>Bool</i> oder <i>Bit</i> sind, können diese mit dem <i>CheckBoxField</i> -Steuerelement dargestellt werden.
<i>CommandField</i>	Für das Wechseln zu den vordefinierten Modi <i>Select</i> , <i>Edit</i> , <i>Delete</i> , <i>Insert</i> und <i>Cancel</i> werden <i>CommandFields</i> verwendet. Welche Buttons angezeigt und wie sie dargestellt werden, muss mit zusätzlichen Attributen bestimmt werden.
<i>HyperLinkField</i>	Damit wird ein echter Hyperlink für jede Zeile erzeugt. Die Zieladresse wird aus Daten zusammengesetzt.
<i>ImageField</i>	Ein Bild wird dargestellt.
<i>TemplateField</i>	Mit diesem Steuerelement können Sie die volle Kontrolle übernehmen und mit Hilfe anderer Webserver-Steuerelemente die Spalte »in Eigenleistung« definieren.

**Tabelle 8.5** Typen manuell definierter Spalten (Fortsetzung)

## BoundField

Die gebundene Datenspalte beinhaltet Attribute für die erweiterte Formatierung. Das kann z.B. das Attribut *DataFormatString* sein, das mit Format-Strings die endgültigen Ausgaben beeinflusst.

```
<asp:BoundField HeaderText="Preis" DataField="UnitPrice" DataFormatString="{0:c}"></asp:BoundField>
```

Die umfangreichen Formatierungsregeln können in der Dokumentation nachgelesen werden. Die häufigsten sind *{0:d}* für eine kurzes Datum und *{0:c}* für Währung. Beachten Sie, dass zusätzlich das Attribut *HtmlEncode* auf *false* gesetzt sein muss, sonst wirken diese Formatierungsregeln nicht.

Alle Spalten haben umfangreiche Style-Eigenschaften für *Item*, *Footer Header* und *Control*. Diese wurden hier in der Liste nicht aufgeführt.

Attribut	Verwendung
<i>AccessibleHeaderText</i>	Ist dieses Attribut <i>true</i> , wird der Kopf mit <i>&lt;TH&gt;</i> - anstelle von <i>&lt;TD&gt;</i> -Elementen im Browser dargestellt.
<i>ApplyFormatInEditMode</i>	Dieses Attribut hängt mit der Formatierung über <i>DataFormatString</i> zusammen. Ist <i>ApplyFormatInEditMode</i> auf <i>true</i> gesetzt, werden Formatierungsregeln auch im Edit-Modus berücksichtigt. So können z.B. auch Währungssymbole in der Textbox angezeigt werden.
<i>ConvertEmptyStringToNull</i>	Mit diesem booleschen Wert kann beeinflusst werden, ob ein leerer String in NULL umgewandelt wird. Dies kann wichtig sein, wenn Daten editiert und NULL-Werte in die Datenbank geschrieben werden sollen.
<i>DataField</i>	Der Name des Feldes aus der Datenquelle, der für das Erstellen der Spalte verwendet wird. Der Typ ist <i>String</i> .
<i>DataFormatString</i>	Mit diesem String werden Daten formatiert, bevor sie im Browser angezeigt werden. Das können z.B. Währungen sein. Es wird immer <i>{0:format}</i> angegeben, wobei <i>format</i> für den eigentlichen Format-String steht.
<i>FooterText</i>	Damit wird der Text definiert, der in der Fußzeile dieser Spalte angezeigt werden soll.
<i>HeaderImageUrl</i>	Damit wird ein Verweis auf ein Bild bestimmt, das in der Kopfzeile dieser Spalte angezeigt werden soll.

**Tabelle 8.6** Die Attribute der Bound Field Columns des GridView-Steuerelements

Attribut	Verwendung
<i>HeaderText</i>	Damit wird der Text definiert, der in der Kopfzeile dieser Spalte angezeigt werden soll.
<i>HtmlEncode</i>	Wenn dieser Wert <i>true</i> ist, wird vor dem Anzeigen der Daten auf jede Zeile dieser Spalte das HTML-Encode-Kommando angewandt. Somit werden potentiell schädliche HTML-Elemente in ihre Entitäten übersetzt.
<i>InsertVisible</i>	Wenn dieser Wert <i>true</i> ist, wird diese Spalte auch angezeigt, wenn der <i>Insert</i> -Modus aktiv ist. Der <i>Insert</i> -Modus wird aktuell nur vom <i>DetailsView</i> -Steuerelement unterstützt.
<i>NullDisplayText</i>	Der Text, der statt einem NULL-Wert angezeigt werden soll.
<i>ReadOnly</i>	Wenn dieser Wert <i>true</i> ist, kann der Wert nicht vom Benutzer verändert werden. Im Edit-Modus wird keine Textbox angezeigt, und die Checkbox bleibt deaktiviert.
<i>ShowHeader</i>	Wenn dieser Wert <i>true</i> ist oder das Attribut fehlt, wird der Kopf für diese Spalte angezeigt.
<i>SortExpression</i>	Setzt oder liest den Ausdruck, der verwendet wird, wenn nach dieser Spalte sortiert wird, als String.

**Tabelle 8.6** Die Attribute der Bound Field Columns des GridView-Steuerelements

## Breite der Textbox

Wenn nun der Benutzer in den Bearbeiten-Modus wechselt, werden die Daten im *GridView* in Textboxen angezeigt. Dabei sind alle Eingabefelder gleich breit. Das ist natürlich oft nicht sinnvoll. Mit dem Attribut *Width* oder dem Element *ControlStyle* kann die Breite festgelegt werden.

```
<asp:BoundField HeaderText="Preis" ItemStyle-HorizontalAlign="Right" DataField="UnitPrice"
DataFormatString="{0:c}" ApplyFormatInEditMode="True">
<controlStyle Width="50" ></controlStyle>
</asp:BoundField>
```

**Listing 8.6** Eine gebundene Spalte mit definierter Breite

Die Breite einer »normalen« Spalte wird übrigens mit *ItemStyle* definiert.

## ButtonField

Ein *ButtonField* erzeugt eine Spalte die einen *Button*, *Hyperlink* oder *ImageLink* im *GridView*-Steuerelement enthält.

```
<asp:ButtonField Text="PressME!" CommandName="Press" FooterText="Fuss"></asp:ButtonField>
```

In der folgenden Tabelle werden nur die erweiterten Attribute aufgeführt. Beachten Sie deshalb auch die Tabelle 8.6, deren Attribute ebenfalls gelten.

Attribut	Verwendung
<i>ButtonType</i>	Mit diesem Attribut wird definiert, ob ein <i>Link</i> , <i>Button</i> oder <i>Image</i> dargestellt wird. Fehlt dieses Attribut, wird ein <i>Hyperlink</i> verwendet.
<i>CausesValidation</i>	Mit dem Wert <i>false</i> wird angegeben, dass ein Übertragen mit diesem Button keine Prüfung durch die Validatoren bewirkt. Fehlt dieser Wert, werden die Eingaben validiert.

**Tabelle 8.7** Auszug aus den Attributen des *ButtonField*-Steuerelements

Attribut	Verwendung
<i>CommandName</i>	Damit wird der Name des Commands angegeben. Dieser muss einmalig sein, da in der Ereignis-Methode des GridView-Steuerelements ausgewertet wird, welcher Button gedrückt wurde.
<i>DataTextField</i>	Mit diesem Wert kann gesteuert werden, welches Feld die Beschriftung des Buttons bestimmt.
<i>DataTextFormatString</i>	Mit diesem <i>FormatString</i> wird die Beschriftung des Buttons formatiert, wenn sie aus der Datenbank kommt.
<i>ImageUrl</i>	Damit wird die URL zum Bild festgelegt, wenn die Darstellung als ImageButton erfolgen soll.
<i>Text</i>	Damit wird der Button beschriftet.
<i>ValidationGroup</i>	Bestimmt, zu welcher Gruppe der Button gehört, wenn eine Validierung durchgeführt wird.

**Tabelle 8.7** Auszug aus den Attributen des *ButtonField*-Steuerelements (*Fortsetzung*)

### CheckBoxField

Mit dem *CheckBoxField* kann eine datengebundene Checkbox dargestellt werden. Diese eignet sich natürlich für Felder vom Datentyp *Boolean*. Wenn die Spalte an kein Feld gebunden wird, wird nur im Bearbeitungsmodus eine Checkbox angezeigt.

```
<asp:CheckBoxField HeaderText="auswahl" ></asp:CheckBoxField>
```

Auch hier werden nur die neuen Attribute vorgestellt. Der Rest findet sich in Tabelle 8.6.

Attribut	Verwendung
<i>Text</i>	Der Text als String der neben jeder Checkbox steht.

**Tabelle 8.8** Auszug aus den Attributen des *CheckBox*-Steuerelements

### CommandField

Mit dem *CommandField*-Element wird eine spezielle Spalte angelegt, mit der die Funktionen *Auswahl*, *Editieren*, *Einfügen* und *Löschen* ausgeführt werden. So wird mit dem Attribut *ShowSelectButton* der entsprechende *LinkButton* in der Spalte aktiviert.

```
<asp:CommandField ShowSelectButton="True" HeaderText="Kopf1" ></asp:CommandField>
```

Wenn Sie die Vorzüge von Visual Studio 2005 nutzen wollen, können Sie auch im Menü *Aufgaben* des *GridView*-Steuerelements den Punkt *neue Spalte hinzufügen* auswählen. Es startet dann ein Assistent mit den wichtigsten Optionen zum Anlegen einer Spalte.



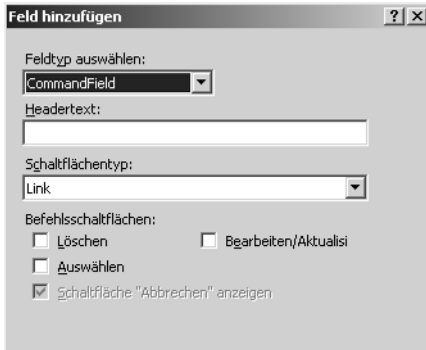


Abbildung 8.4 Assistant-Feld hinzufügen

Das Ergebnis ist dasselbe wie unter der Verwendung des Menüpunktes *Spalten bearbeiten*. Wenn es um das Hinzufügen von Feldern geht, wie z.B. einer Hyperlink Spalte, ist es effektiver den Menüpunkt *neue Spalte hinzufügen* zu verwenden.

In der folgenden Tabelle werden die dabei möglichen Attribute des CommandFields erläutert.

Attribut	Wert
<i>AccessibleHeaderText</i>	Überschrift für barrierefreies Lesen der Webseite
<i>ButtonType</i>	Damit wird aus der ButtonType Collection einer der drei möglichen Typen <i>Button</i> , <i>Image</i> oder <i>Link</i> definiert.
<i>CancelImageUrl</i>	URL des Images für den <i>Cancel</i> -Button.
<i>CancelText</i>	Beschriftung der <i>Cancel</i> -Aktion.
<i>CausesValidation</i>	Boolescher Wert, der angibt, ob die Validierung durchgeführt wird.
<i>DeleteImageUrl</i>	URL des Images für den <i>Delete</i> -Button.
<i>EditImageUrl</i>	URL des Images für den <i>Edit</i> -Button.
<i>EditText</i>	Beschriftung des Links, um einen Datensatz zu editieren.
<i>FooterText</i>	Text in der Fußzeile.
<i>HeaderImageUrl</i>	URL des Images für den Header.
<i>HeaderText</i>	Überschrift für die aktuelle Spalte.
<i>SelectImageUrl</i>	URL des Images für den <i>Auswahl</i> -Button.
<i>SelectText</i>	Text für Beschriftung der <i>Select</i> -Buttons oder -Links.
<i>ShowCancelButton</i>	Boolescher Wert, der bestimmt, ob der Button angezeigt werden soll.
<i>ShowDeleteButton</i>	Boolescher Wert, der bestimmt, ob der Button angezeigt werden soll.
<i>ShowHeader</i>	Boolescher Wert, der bestimmt, ob die Überschrift angezeigt werden soll.
<i>ShowInsertButton</i>	Boolescher Wert, der bestimmt, ob der Button angezeigt werden soll.
<i>ShowSelectButton</i>	Boolescher Wert, der bestimmt, ob der Button angezeigt werden soll.
<i>SortExpression</i>	Gibt den Ausdruck als String an, nach dem sortiert werden soll.
<i>UpdateImageUrl</i>	URL des Images für den <i>Cancel</i> -Button.

Tabelle 8.9 Ausgewählte Attribute der Command Field Column

Attribut	Wert
<i>UpdateText</i>	Textbeschriftung für <i>Update</i> -Kommando.
<i>ValidationGroup</i>	Gibt die Validierungsgruppe an.

**Tabelle 8.9** Ausgewählte Attribute der Command Field Column (Fortsetzung)

**HINWEIS** Es stehen offensichtlich Attribute zur Verfügung, die aktuell im *GridView*-Steuerelement nicht zur Anwendung kommen werden. Das sind z.B. *InsertText* oder *NewText*.

## HyperLinkField

Mit dem *HyperLink*-Feld lassen sich Hyperlinks erzeugen, die aus Daten zusammengesetzt werden können. In der Regel wird damit dem Benutzer eine Möglichkeit geboten, auf eine neue Seite zu kommen, unter Mitnahme des darunter liegenden Datensatzes. In der Praxis kann so eine Detailansicht dargestellt werden. Die Weitergabe z.B. einer ID erfolgt dabei im QueryString in der Form *detail.aspx?id=123*. Sie können auch mehr als ein Feld zur Erzeugung des QueryStrings heranziehen. Der URL-String wird dann im Attribut *DataNavigateUrlFormatString* zusammengesetzt. Die Query-Parameter werden in geschweiften Klammern (»{}«) angegeben und später durch die realen Werte ersetzt. Die 0 steht für den ersten Wert, die 1 für den zweiten, usw.

```
<asp:hyperlinkfield datatextfield="ProductName"
    datanavigateurlfields="ProductID,ProductName"
    datanavigateurlformatstring ="http://www.devtrain.de?artikel={0}&name={1}"
    sortexpression="ProductName" showheader="True" headertext="Product"
    headerstyle-font-names="Arial" headerstyle-font-bold="True"
    itemstyle-font-names="Verdana" itemstyle-font-bold="True" />
```

**Listing 8.7** Hyperlink im GridView-Control

Die wichtigsten Attribute der *HyperlinkField*-Spalte werden in der folgenden Tabelle erklärt.

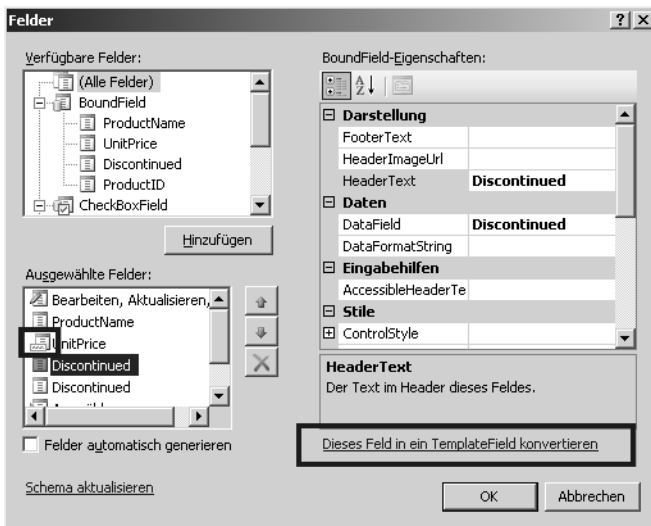
Attribut	Verwendung
<i>DataTextField</i>	Damit wird der angezeigte Text des Links gesteuert. Der zugewiesene String gibt den Feldnamen an.
<i>DataTextFormatString</i>	Mit dem <i>FormatString</i> wird schlussendlich die Ausgabe erzeugt: <i>datatextformatstring="mein Text{0} ende"</i> . Zusätzlich kann so mit dem <i>Format Expressions</i> die Formatierung nach Datum oder Währung vorgenommen werden.
<i>Text</i>	Damit legen Sie einen fixen Text zur Anzeige fest.
<i>DataNavigateUrlFields</i>	Mit diesem String werden die Datenfelder, die zur Bildung des URLs benötigt werden, durch Kommata getrennt definiert.
<i>DataNavigateUrlFormatString</i>	Damit erzeugt man den Navigationslink.
<i>NavigateUrl</i>	Damit kann eine feste Ziel-Adresse definiert werden.
<i>Target</i>	Damit wird das <i>Target</i> -HTML-Attribut eines Hyperlinks gesteuert. Die möglichen Werte wie <i>_blank</i> sind deshalb daran angelehnt. Damit wird ein Link in einem neuen Browser Fenster geöffnet.

**Tabelle 8.10** HyperlinkField-Attribute

## Template-Spalten

Wenn die Möglichkeiten der üblichen Spalten nicht reichen, bleibt noch die Alternative der Verwendung von Templates. Damit haben Sie volle Kontrolle über Design und Layout jeder einzelnen Spalte. Es können HTML-Elemente und auch mehrere Webserver-Steuerelemente darin platziert werden. Auf den ersten Blick gestaltet sich der Einsatz schwierig, da das Ergebnis in der ASPX-Seite leicht unübersichtlich werden kann.

Der einfachste Weg führt über das Aufgabenmenü des *GridView-Steuerelements* und den Menüpunkt *Spalten bearbeiten*. Im Dialog *Felder* kann jede Spalte mit dem Hyperlink *Dieses Feld in ein TemplateField konvertieren* in eine Template-Spalte umgewandelt werden.



**Abbildung 8.5** Felder im GridView in Template Columns umwandeln

Im *GridView*-Steuerelement gibt es dann pro Spalte drei mögliche Template-Bereiche. Im Bereich *ItemTemplate* wird definiert, wie die Spalte im Normalfall dargestellt wird. Für die alternierenden Zeilen gibt es noch das *AlternatingItemTemplate*. Der Inhalt des Bearbeiten-Modus wird im *EditItemTemplate* definiert.

Im Bearbeiten-Modus kommen dann auch eher Webserver-Steuerelemente mit Eingabemöglichkeit in Frage. So wird im folgenden Beispiel eine *TextBox* verwendet. Die Bindung an die Daten erfolgt mit dem neuen Bindungsbefehl *Bind* an die Eigenschaft *Text*. *Bind* erhält als Parameter den Feldnamen und eventuell einen Formatierungsausdruck. Natürlich kann man auch an jede andere Eigenschaft auf diese Weise Daten binden.

Für die Darstellung im *ItemTemplate* wird in diesem Beispiel ein Label Webserver Steuerelement verwendet.

```
<asp:TemplateField HeaderText="Preis">
  <ItemTemplate>
    <asp:Label ID="Label2" runat="server" Text='<%= Bind("UnitPrice", "{0:c}") %>'></asp:Label>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:TextBox ID="TextBox2" runat="server" Text='<%= Bind("UnitPrice", "{0:c}") %>'></asp:TextBox>
  </EditItemTemplate>
  <ItemStyle HorizontalAlign="Right" Width="150px" />
  <ControlStyle Width="50px" />
</asp:TemplateField>
```

**Listing 8.8** Eine Template-Spalte

Für die reine Anzeige reicht auch reiner HTML-Code mit einer Bindung über *Eval*. Dies kann innerhalb eines *ItemTemplate*-Elements statt des vorher angeführten Labels platziert werden.

```
<%# Eval("Preis") %>
```

Es können auch mehrere Datenfelder in einer Spalte dargestellt werden. Selbst Tabellen oder sonstige HTML-Formatierungen können in den Templates umgesetzt werden.

Wenn die Daten noch umgewandelt werden müssen, etwa weil daraus der Link zu einem Bild berechnet werden soll, kann in die Datenbindung auch eine Funktion eingebaut werden, die diese Aufgabe wahrnimmt. Der Rückgabewert wird dann zur Laufzeit angezeigt.

```
<%# UmwandelN(Eval("Preis")) %>
```

## GridView Ereignis Methoden

Das *GridView*-Steuerelement startet eine der eingebauten Funktionen, wenn ein entsprechendes Ereignis eintritt. Ein solches Ereignis kann ein simples Auswählen einer Zeile durch den Benutzer sein. Dann wird die Methode *SelectedIndexChanged* ausgeführt. Technisch ist dafür natürlich ein Postback zum Server nötig. In der Methode am Server werden dann Werte aus dem *GridView* ausgelesen. So kann z.B. mit der Eigenschaft *SelectedValue* auf den gewählten Text zugegriffen werden.

```
Sub GridView1_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs)
    Label1.Text = GridView1.SelectedValue.ToString
End Sub
```

**Listing 8.9** Eine Ereignisbehandlung des GridView-Steuerelements

Im Folgenden werden einige der wesentlichen Attribute des *GridView* beschrieben, die zur Laufzeit aber nur gelesen werden können.

Eigenschaft vom GridView	Bedeutung
<i>BottomPagerRow</i>	Damit erhält man eine Referenz auf die Instanz der <i>GridViewRow</i> , die das Pager Steuerelement beinhaltet.
<i>ColumnFields</i>	Gibt eine Collection von <i>DataControlField</i> -Objekten zurück, die alle Spalten im <i>GridView</i> repräsentieren.
<i>DataKeys</i>	Damit erhält man eine Collection von <i>DataKey</i> -Objekten, die alle Keys des <i>GridView</i> enthält.
<i>FooterRow</i>	Damit erhält man den Fußzeilenbereich des <i>GridView</i> als <i>GridViewRow</i> -Objekt zurück.
<i>HeaderRow</i>	Damit erhält man den Kopfzeilenbereich des <i>GridView</i> als <i>GridViewRow</i> -Objekt zurück, und es lässt sich nach Erstellung des Grid Einfluss auf die Überschriften nehmen.
<i>PageCount</i>	Diese Eigenschaft gibt die Anzahl der Seiten beim Paging zurück. Diese wird aufgrund der Anzahl der Datensätze pro Seite und ihrer Gesamtanzahl errechnet.
<i>Rows</i>	Dies ist eine Auflistung aller Zeilen vom Typ <i>GridViewRow</i> . Diese können per Index angesprochen werden. Dies dient dazu, den Inhalt der Zellen anzusprechen.
<i>SelectedRow</i>	Damit erhält man die aktuell selektierte Zeile. So kann der Inhalt über die darin enthaltene <i>Cells</i> Collection per Index angesprochen und gelesen werden.

**Tabelle 8.11** Ausgewählte Read Only-Eigenschaften des GridView-Steuerelements

Eigenschaft vom GridView	Bedeutung
<i>SelectedDataKey</i>	Damit erhält man das aktuell ausgewählte <i>DataKey</i> -Objekt. Wenn dieses Null ist, ist kein Datensatz selektiert. Mit der Eigenschaft <i>Value</i> kann auf den Wert zugegriffen werden.
<i>SelectedValue</i>	Hat einen ähnlichen Zweck wie <i>SelectedDataKey</i> , ist allerdings kürzer im Code.
<i>SortExpression</i>	Gibt einen String mit dem aktuellen Sortierkriterium zurück.
<i>SortDirection</i>	Gibt die aktuelle Sortierreihenfolge zurück. Mögliche Werte sind: <i>Descending</i> und <i>Ascending</i> .
<i>TopPagerRow</i>	Damit erhält man eine Referenz auf die Zeile, die das Pager Steuerelement beinhaltet.
<i>PagerSettings</i>	Liefert ein Objekt vom Typ <i>PagerSettings</i> . Damit kann dann z.B. auf die Pager-Buttons Einfluss genommen werden.

**Tabelle 8.11** Ausgewählte Read Only-Eigenschaften des GridView-Steuerelements (Fortsetzung)

Diese Werte werden oft benötigt, um in den Ereignismethoden richtig reagieren zu können. So können der aktuelle Datensatz oder die geänderten Daten bestimmt werden.

Weitere wichtige Information findet man naturgemäß in den Parametern der Ereignismethode.

Nachfolgend werden die wichtigsten Ereignisse des *GridView* kurz erläutert.

Ereignis	Bedeutung
<i>PageIndexChanged</i>	Wird ausgeführt, wenn der Benutzer weiter geblättert hat und die Seite gewechselt wurde.
<i>PageIndexChanging</i>	Wird ausgeführt, bevor die Seite gewechselt wird. Dabei werden die Parameter in <i>GridViewSelectEventArgs</i> übergeben, aus denen man mit der Eigenschaft <i>NewSelectedIndex</i> schon den Index der zukünftig angezeigten Seite erhält. Die zweite wichtige Eigenschaft <i>Cancel</i> kann mit dem Parameter <i>true</i> den Seitenwechsel abbrechen.
<i>RowCancelingEdit</i>	Wird ausgeführt, wenn der <i>Cancel</i> -Button gedrückt wird. Dieser wird im Edit-Modus angezeigt. Änderungen an den Daten gehen dann verloren. Über die dabei übergebenen <i>GridViewCancelEventArgs</i> kann man mit der Eigenschaft <i>RowIndex</i> schon den Index der aktuellen Reihe erhalten. Die zweite wichtige Eigenschaft <i>Cancel</i> kann mit dem Parameter <i>true</i> den Abbruch verhindern.
<i>RowCommand</i>	Dieses Ereignis wird ausgeführt, wenn ein Button oder Link im <i>GridView</i> -Steuerelement gedrückt wird. Es wird ein Objekt vom Typ <i>GridViewCommandEventArgs</i> übergeben, das folgende Eigenschaften hat: <i>CommandSource</i> : Eine Referenz auf das Steuerelement vom Typ <i>Object</i> , das das Ereignis ausgelöst hat. <i>CommandName</i> : Der String-Name des Controls, der das Ereignis ausgelöst hat. <i>CommandArgument</i> : Ist in diesem Zusammenhang der <i>RowIndex</i> des gedrückten Buttons.
<i>RowCreated</i>	Wird ausgelöst, wenn eine neue Zeile erzeugt wurde. Erhält als Parameter ein Objekt vom Typ <i>GridViewRowEventArgs</i> . Mit der Eigenschaft <i>Row</i> erhält man ein komplettes Row-Objekt von der neu erzeugten Reihe.
<i>RowDataBound</i>	Dieses Ereignis wird ausgeführt, wenn eine Zeile an ihre Daten gebunden wurde. Dies ist der ideale Ort, um einen Wert noch zu beeinflussen, bevor er angezeigt wird. Mit der Eigenschaft <i>Row</i> erhält man ein komplettes Row-Objekt von der neu erzeugten Zeile.
<i>RowDeleted</i>	Dieses Ereignis wird ausgeführt, bevor eine Reihe gelöscht wird. Dabei wird ein Objekt vom Typ <i>GridViewDeletedEventArgs</i> übergeben. <i>AffectedRows</i> : Die Anzahl der Reihen, die gelöscht wurden. <i>Exception</i> : Das <i>Exception</i> -Objekt, das erzeugt wurde, wenn der Löschvorgang fehlschlug. <i>ExceptionHandled</i> : Wenn die Exception behandelt wurde, sollte diese Eigenschaft auf <i>true</i> gesetzt werden, um die Exception nicht weiter durchzureichen. <i>Keys</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die Primärschlüssel enthält. <i>Values</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die Werte der Primärschlüssel enthält.

**Tabelle 8.12** Auszug aus den Ereignissen des GridView-Steuerelements

Ereignis	Bedeutung
<i>RowDeleting</i>	Wird ausgeführt, bevor die Zeile gelöscht wird. Dabei wird ein Objekt vom Typ <i>GridViewDeletedEventArgs</i> übergeben. Darin sind folgende Eigenschaften enthalten: <i>Cancel</i> : Wenn Sie diesen booleschen Wert auf <i>true</i> setzen, wird die Löschoption abgebrochen. <i>RowIndex</i> : Gibt den Index der Zeile zurück. <i>Keys</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die Primärschlüssel enthält. <i>Values</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die Werte der Spalten enthält.
<i>RowEditing</i>	Wird ausgeführt, bevor die Zeile geändert wird. Dabei wird ein Objekt vom Typ <i>GridViewEditEventArgs</i> übergeben. Darin sind folgende Eigenschaften enthalten. <i>Cancel</i> : Wenn Sie diesen booleschen Wert auf <i>true</i> setzen, wird die Löschoption abgebrochen. <i>NewRowIndex</i> : Gibt den Index der Zeile zurück, die editiert wird oder editiert werden soll. Wenn der Wert <i>-1</i> ist, wird das Editieren abgebrochen.
<i>RowUpdated</i>	Wird ausgeführt, bevor die Zeile geändert wird. Dabei wird ein Objekt vom Typ <i>GridViewUpdatedEventArgs</i> übergeben. Darin sind folgende Eigenschaften enthalten: <i>KeepInEditMode</i> : Wenn Sie diesen booleschen Wert auf <i>true</i> setzen, wird der Edit-Modus beibehalten. <i>RowIndex</i> : Gibt den Index der Zeile zurück. <i>Keys</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die Primärschlüssel enthält. <i>NewValues</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die neuen Werte der Zeile enthält. <i>OldValues</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die alten Werte der Zeile enthält. <i>AffectedRows</i> : Die Anzahl der Zeilen, die geändert wurden. <i>Exception</i> : Das <i>Exception</i> -Objekt, das erzeugt wurde, wenn der Update-Vorgang fehlgeschlagen ist. <i>ExceptionHandled</i> : Wenn die Exception behandelt wurde, sollte diese Eigenschaft auf <i>true</i> gesetzt werden, um die Exception nicht weiter durchzureichen.
<i>RowUpdating</i>	Wird ausgeführt, nachdem die Zeile geändert wurde. Dabei wird ein Objekt vom Typ <i>GridViewUpdateEventArgs</i> übergeben. Darin sind folgende Eigenschaften enthalten: <i>Cancel</i> : Wenn Sie diesen booleschen Wert auf <i>true</i> setzen, wird die Löschoption abgebrochen. <i>RowIndex</i> : Gibt den Index der Zeile zurück. <i>Keys</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die Primärschlüssel enthält. <i>NewValues</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die neuen Werte der Zeile enthält. <i>OldValues</i> : Ein Objekt vom Typ <i>IOrderedDictionary</i> , das die alten Werte der Zeile enthält.
<i>SelectedIndexChanged</i>	Wird ausgeführt, bevor eine andere Zeile selektiert wird.
<i>SelectedIndexChanging</i>	Wird ausgeführt, bevor die Zeile gewechselt wird. Dabei wird ein Objekt vom Typ <i>GridViewSelectEventArgs</i> übergeben. Darin sind folgende Eigenschaften enthalten: <i>Cancel</i> : Wenn Sie diesen booleschen Wert auf <i>true</i> setzen, wird die Operation abgebrochen. <i>NewSelectedIndex</i> : Gibt den Index der Zeile zurück, die gewählt werden soll. Wenn der Wert <i>-1</i> ist, wird keine Zeile selektiert.
<i>Sorted</i>	Wird ausgeführt, wenn die Zeile gewechselt wurde.
<i>Sorting</i>	Wird ausgeführt, bevor eine Sortieraktion durchgeführt wird. Dabei wird ein Objekt vom Typ <i>GridViewSortEventArgs</i> übergeben. Darin sind folgende Eigenschaften enthalten: <i>Cancel</i> : Wenn Sie diesen booleschen Wert auf <i>true</i> setzen, wird die Operation abgebrochen. <i>SortExpression</i> : Gibt den Sortierausdruck als String zurück.

**Tabelle 8.12** Auszug aus den Ereignissen des GridView-Steuererelements (*Fortsetzung*)

## Zelle abhängig von Daten formatieren

Zum Abschluss des *GridView*-Teiles wird ein Beispiel gezeigt, das die Zelle abhängig von den Daten formatiert. Es handelt sich dabei um die Spalte mit dem Preis (*UnitPrice*) die vom Datentyp *Decimal* ist.

Das Ereignis *RowCreated* tritt für jede erstellte Reihe im *GridView* ein. Den Zugriff auf den Wert eines Feldes erhält man per *DataBinder.Eval*. Dieser Funktion übergibt man das *DataItem*-Objekt der aktuellen *Row*. Damit kann man dann einen Wertvergleich, in diesem Fall kleiner als 15, durchführen.

Das Steuerelement findet man mit der Methode *FindControl* oder *Cells*-Auflistung. Das Label *lblPreis* befindet sich in einer Template-Spalte, wird per *FindControl* gefunden und mit *CType* in ein Label umgewandelt. Dieses Label wird dann einfach mit dem Attribut *ForeColor* umgefärbt, wenn die Bedingung zutrifft. Da es verschiedene Arten von Reihen gibt, wird am Anfang noch geprüft, ob es sich um eine normale Datenreihe (*DataRow*) handelt.

```
Protected Sub GridView1_RowCreated(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
    If e.Row.RowType = DataControlRowType.DataRow Then
        Dim lbl As Label = CType(e.Row.FindControl("lblPreis"), Label)
        If DataBinder.Eval(e.Row.DataItem, "unitprice") > 15D Then
            lbl.ForeColor = Drawing.Color.White
        End If
    End If
End Sub
```

Listing 8.10 GridView Ereignis Methode RowCreated



Abbildung 8.6 Werte größer als zehn werden weiß dargestellt

## Client Callback à la AJAX

Eine der absolut »coolsten« Funktionen ist *EnableSortingAndPagingCallbacks*. Wenn dieses Attribut auf *true* gesetzt ist, wird die ASP.NET-Funktion *ClientCallback* verwendet, um die Daten zu aktualisieren. Das heißt, dass nur der Inhalt der HTML-Tabelle ausgetauscht und kein üblicher Postback ausgeführt wird. Details zu dieser Technologie werden in Kapitel 12 beschrieben.

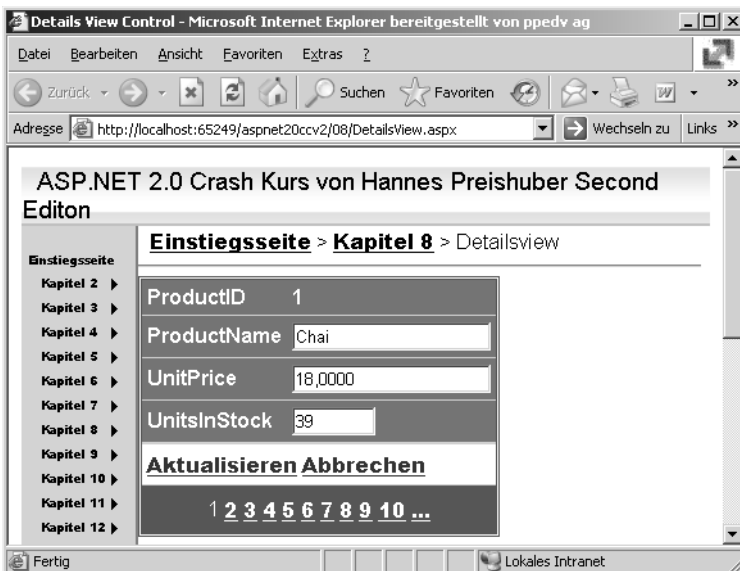
## DetailsView

Das *DetailsView*-Steuerelement ist sozusagen das »GridView mit nur einem Datensatz«. Darüber hinaus können Sie damit auch neue Datensätze anlegen. Das Objektmodell, die Eigenschaften und Attribute sind denen des *GridView*-Steuerelements sehr ähnlich. Es lassen sich die Felder sogar in *Template Columns* umwandeln. Allerdings finden sich diese Felder im Element *<Fields>*, statt wie im *GridView* *<columns>*.

```
<asp:DetailsView ID="DetailsView1" Runat="server" DataSourceID="SqlDataSource1"
AutoGenerateRows="False"
BackColor="White" BorderWidth="1px" GridLines="Vertical" BorderColor="#999999" BorderStyle="None"
CellPadding="3" DefaultMode="Edit" AutoGenerateDeleteButton="True"
AutoGenerateEditButton="True" AutoGenerateInsertButton="True" HorizontalAlign="Left" AllowPaging="True"
>
<Fields>
  <asp:BoundField HeaderText="ProductName" DataField="ProductName" SortExpression="ProductName">
  </asp:BoundField>
  <asp:BoundField HeaderText="UnitPrice" DataField="UnitPrice" SortExpression="UnitPrice">
  </asp:BoundField>
  <asp:CheckBoxField HeaderText="Discontinued" SortExpression="Discontinued" DataField="Discontinued">
  </asp:CheckBoxField>
</Fields>
</asp:DetailsView>
```

**Listing 8.11** DetailsView-Steuerelement

Die Darstellung im Browser erfolgt wie in einem Formular üblich. Eingabefelder sind untereinander positioniert. Darunter befinden sich die Funktionsschaltflächen für *bearbeiten*, *aktualisieren* oder *einfügen*. In diesem Beispiel ist ganz nach *GridView*-Art das Paging aktiviert.



**Abbildung 8.7** DetailsView-Steuerelement im Bearbeiten-Modus

Die Verbindung zur Datenbank wird natürlich mit dem *DataSource*-Steuerelement hergestellt.



```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server" ProviderName="System.Data.SqlClient"
  ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
  SelectCommand="SELECT [ProductName], [UnitPrice], [Discontinued] FROM [Products]">
</asp:SqlDataSource>
```

**Listing 8.12** SQLDataSource-Steuerelement als Datenquelle

Auch Insert- oder Update-Vorgänge werden hier über die entsprechenden Attribute vorgenommen, wie in Kapitel 7 beschrieben.

Auch hier gibt es eine der AJAX-Möglichkeiten über das Attribut *EnablePagingCallbacks*. Wenn dieses auf *true* gesetzt ist, wird die ASP.NET-Funktion *ClientCallBack* verwendet, um die Daten zu aktualisieren. Das heißt, dass nur der Inhalt der Felder ausgetauscht und kein üblicher Postback ausgeführt wird. Dies funktioniert allerdings nicht mit Template-Spalten. Details zu dieser Technologie werden in Kapitel 12 beschrieben.

## DetailsView-Attribute

Natürlich kann auch dieses Steuerelement mit dem Aufgabenmenü nahezu vollständig konfiguriert werden. Trotzdem sind die dabei verwendeten Attribute von Bedeutung. Im Folgenden werden nur die im Unterschied zum *GridView* hinzugekommenen erläutert.

Attribut	Verwendung
<i>AutoGenerateDeleteButton</i>	Dieser boolesche Wert gibt an, ob der <i>Delete</i> -Button automatisch generiert werden soll.
<i>AutoGenerateEditButton</i>	Dieser boolesche Wert gibt an, ob der <i>Edit</i> -Button automatisch generiert werden soll.
<i>AutoGenerateInsertButton</i>	Dieser boolesche Wert gibt an, ob der <i>Insert</i> -Button automatisch generiert werden soll.
<i>AutoGenerateRows</i>	Dieser boolesche Wert gibt an, ob die Formularfelder anhand der Daten automatisch generiert werden sollen.
<i>DataKeyNames</i>	Dies ist eine Auflistung der <i>Key</i> -Felder, die zum Aktualisieren der Daten nötig sind. Mehrere Keys werden durch Kommata getrennt. Die Keys werden dem Benutzer nicht angezeigt.
<i>DataMember</i>	Über <i>DataMember</i> wird die eindeutige Tabelle spezifiziert, die angezeigt werden soll. Dies ist dann von Bedeutung, wenn ein <i>DataSet</i> mit mehreren Tabellen als Datenquelle dient.
<i>DataSource</i>	Jede Art von Objekt, die das Interface <i>System.Collections.IEnumerable</i> beinhaltet, kann als Datenquelle dienen.
<i>DataSourceID</i>	Mit diesem String wird ein vorhandenes DataSource-Steuerelement als Datenquelle bestimmt.
<i>DefaultMode</i>	Das <i>DetailsView</i> -Steuerelement kann in drei Modi arbeiten: <i>Edit</i> , <i>Insert</i> , <i>ReadOnly</i> . Damit wird der Modus ausgewählt, der beim ersten Aufruf angezeigt werden soll.
<i>EmptyDataText</i>	Damit gibt man an, welcher Text angezeigt werden soll, wenn der Inhalt des Feldes NULL ist.
<i>EnablePagingCallbacks</i>	Boolescher Wert, mit dem die Script Callback-Technologie beim Paging verwendet wird. Funktioniert nur ab Internet Explorer 5.5 und nur ohne Template-Spalten.
<i>GridLines</i>	Damit wird gesteuert, welche Trennlinien angezeigt werden. Mögliche Werte sind <i>Both</i> , <i>None</i> , <i>Horizontal</i> oder <i>Vertical</i> .
<i>HorizontalAlign</i>	Damit wird definiert, wie das gesamte Formular positioniert wird. Mögliche Werte sind <i>Center</i> , <i>Justify</i> , <i>Left</i> , <i>NotSet</i> oder <i>Right</i> .
<i>PageIndex</i>	Setzt oder liest den aktuell angezeigten Seitenindex.

**Tabelle 8.13** Attribute des DetailView-Controls

Genau wie beim *GridView*-Steuerelement gibt es in den Command-Columns erweiterte Attribute. Hier werden nur die zusätzlichen Attribute beschrieben.

Attribut	Verwendung
<i>InsertImageUrl</i>	Der URL des Images, das für den <i>Insert</i> -Button angezeigt werden soll.
<i>InsertText</i>	Der Text, der auf dem <i>InsertCommand</i> -Button geschrieben stehen soll.
<i>InsertVisible</i>	Boolescher Wert, der definiert, ob der <i>InsertCommand</i> -Button oder -Link angezeigt werden soll.
<i>NewImageUrl</i>	Der URL des Images, das für den <i>New</i> -Button angezeigt werden soll.
<i>NewText</i>	Der Text, mit dem der <i>NewCommand</i> -Button beschriftet werden soll.
<i>ShowInsertButton</i>	Boolescher Wert, der definiert, ob der <i>Insert</i> -Button angezeigt werden soll.

**Tabelle 8.14** Erweiterte CommandField-Attribute

Diese Attribute beziehen sich auf die erweiterte Möglichkeit, auch neue Datensätze einzufügen.

## Daten einfügen

Auch das Erstellen von neuen Datensätzen ist sehr einfach. Die Datenquelle, also das DataSource-Control, muss entsprechend konfiguriert sein. Es muss deshalb das SQL-Insert-Kommando hinterlegt werden. Der folgende Code wurde somit komplett mit Assistenten erstellt und funktioniert ohne eine Zeile klassischen Programmierens. Einzige Deklarationen in der ASPX-Seite sind notwendig.

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1" BorderWidth="2px"
  BackColor="White" GridLines="None" CellPadding="3" CellSpacing="1" BorderStyle="Ridge"
  BorderColor="White" AllowSorting="True" EnableViewState="true" AllowPaging="True"
  PageSize="5" AutoGenerateColumns="False" OnSelectedIndexChanged="GridView1_SelectedIndexChanged"
  OnRowCreated="GridView1_RowCreated" EnableSortingAndPagingCallbacks=true>
  <PagerStyle ForeColor="Black" HorizontalAlign="Right" BackColor="White"></PagerStyle>
  <RowStyle ForeColor="Black" BackColor="#DEDFDE"></RowStyle>
  <Columns>
    <asp:CommandField ShowEditButton="True" />
    <asp:BoundField DataField="ProductName" HeaderText="ProductName"
      SortExpression="ProductName" />
    <asp:TemplateField HeaderText="UnitPrice" SortExpression="UnitPrice">
      <EditItemTemplate>
        <asp:TextBox ID="TextBox1" runat="server"
          Text='<%=# Bind("UnitPrice") %>'></asp:TextBox>
      </EditItemTemplate>
      <ItemTemplate>
        <asp:Label ID="lblPreis" runat="server" Text='<%=# Bind("UnitPrice") %>'></asp:Label>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:BoundField DataField="Discontinued" HeaderText="Discontinued"
      SortExpression="Discontinued" />
    <asp:BoundField DataField="Discontinued" HeaderText="Discontinued"
      SortExpression="Discontinued" />
    <asp:CommandField ShowSelectButton="True" />
    <asp:HyperLinkField DataNavigateUrlFields="ProductID"
      DataNavigateUrlFormatString="details.aspx?id={0}"
```

**Listing 8.13** Auszug DetailsView mit Insert-Funktion

```

        DataTextField="ProductID" HeaderText="Link" />
    </Columns>
</asp:GridView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    SelectCommand="SELECT [ProductName], [UnitPrice], [Discontinued], [ProductID] FROM [Products]"
    EnableViewState="False"
    ConnectionString="<%= $ ConnectionStrings:NorthwindConnectionString %>"
    DeleteCommand="DELETE FROM [Products] WHERE [ProductID] = @ProductID"
    InsertCommand="INSERT INTO [Products] ([ProductName], [UnitPrice], [Discontinued]) VALUES
        (@ProductName, @UnitPrice, @Discontinued)"
    UpdateCommand="UPDATE [Products] SET [ProductName] = @ProductName, [UnitPrice] = @UnitPrice,
        [Discontinued] = @Discontinued WHERE [ProductID] = @ProductID">
    <DeleteParameters>
        <asp:Parameter Name="ProductID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="ProductName" Type="String" />
        <asp:Parameter Name="UnitPrice" Type="Decimal" />
        <asp:Parameter Name="Discontinued" Type="Boolean" />
        <asp:Parameter Name="ProductID" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
        <asp:Parameter Name="ProductName" Type="String" />
        <asp:Parameter Name="UnitPrice" Type="Decimal" />
        <asp:Parameter Name="Discontinued" Type="Boolean" />
    </InsertParameters>
</asp:SqlDataSource>

```

**Listing 8.13** Auszug DetailsView mit Insert-Funktion (Fortsetzung)

Je nach Darstellungsmodus werden zur Laufzeit im Browser passend die Funktionsbuttons und die Pagingleiste eingeblendet.



**Abbildung 8.8** Einfügen eines neuen Datensatzes

Zwar besitzt das *DetailsView*-Steuerelement auch die Möglichkeit zum Blättern, aber so richtig nützlich ist das nicht. In der Praxis wird eher eine Kombination aus *GridView* und *DetailsView* zum Einsatz kommen. Damit lässt sich dann das *DetailsView* als Steuerelement für Bearbeiten und Einfügen benutzen.

Die Steuerung des anzuzeigenden Datensatzes erfolgt z.B. mithilfe der ID des Datensatzes und Übergabe dieser im QueryString. Dazu fügen Sie einem *GridView*-Steuerelement eine Hyperlinkspalte hinzu, die diesen Wert in der Form *details.aspx?id=ALFKI* enthält. In der Seite, die das *DetailsView*-Steuerelement enthält, wird dann in dem DataSource-Steuerelement die *WHERE*-Bedingung mit einem Querystring-Parameter versehen. Ein ähnlicher Ansatz wird verfolgt, wenn die Detaildatensätze innerhalb der gleichen Seite angezeigt werden sollen. Man spricht dann von *Master-Detail*.

## Master & Detail

Eine Master-Detail-Darstellung kann aus vielen Gründen erfolgen. Eine Rechnung mit Positionen könnte man zum Beispiel so realisieren. Auch wenn die Daten zu umfangreich sind, um in einer Zeile dargestellt zu werden, lässt sich dazu eine Master-Detail-Darstellung verwenden. Auch wenn die Anzahl der Tabellenfelder zu umfangreich ist, um diese in einer Zeile zu editieren, lässt sich in Verbindung eines *GridView*- und eines *FormView*-Steuerelements innerhalb einer Seite alles darstellen.

Im folgenden Beispiel werden die Bestellungen pro Kunde aus der Nordwind-Datenbank angezeigt. Eine Liste erlaubt das Auswählen eines Kunden, dessen Bestellungen angezeigt werden.

ASP.NET 2.0 Crash Kurs von Hannes Preishuber Second Editon

	CompanyName	City	PostalCode	OrderID	OrderDate	Freight	ShipCity
<a href="#">Auswählen</a>	Alfreds Futterkiste	Berlin	12209	10308	18.09.1996 00:00:00	1,6100	México D.F.
<a href="#">Auswählen</a>	Ana Trujillo Emparedados y helados	México D.F.	05021				
<a href="#">Auswählen</a>	Antonio Moreno Taquería	México D.F.	05023				
<a href="#">Auswählen</a>	Around the Horn	London	WA1 1DP				
<a href="#">Auswählen</a>	Berglunds snabbköp	Luleå	S-958 22				
<a href="#">Auswählen</a>	Blauer See Delikatessen	Mannheim	68306				
<a href="#">Auswählen</a>	Blondesddsl père et fils	Strasbourg	67000				

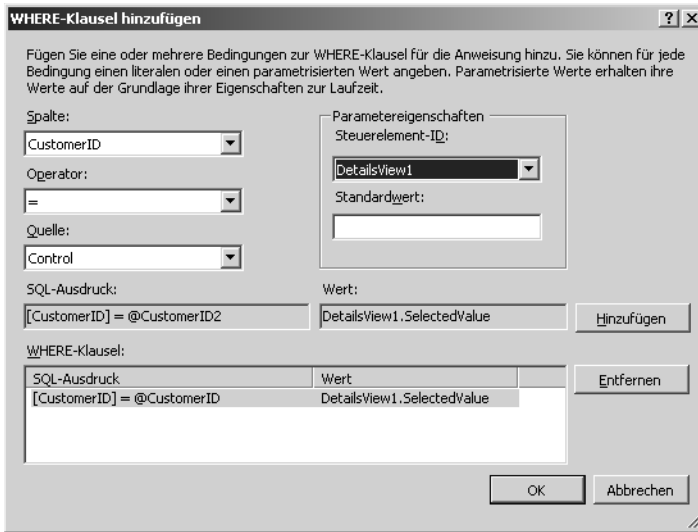
**Bearbeiten Löschen Neu**

1 2 3 4

Abbildung 8.9 GridView und DetailsView

Dazu braucht man zwei DataSource-Steuerelemente. Das erste Steuerelement liest alle Daten aus der Kundentabelle. Das zweite Steuerelement liest die dazu passenden Bestellungen. Die Darstellung erfolgt mittels der Steuerelemente *GridView* und *DetailsView*, so dass die Einzelbestellung auch gleich noch editierbar ist.

Wie immer kann man das sehr einfach mithilfe der visuellen Aufgabenmenüs konfigurieren. Dabei bei der *OrdersSqlDataSource* ein Parameter für die *Where*-Bedingung erstellt, der abhängig vom *GridView* ist – genauer gesagt vom mit *SelectedValue* selektierten Datensatz. Dazu muss allerdings das *GridView*, das die Kunden anzeigt, auch eine Spalte zum *Auswählen* anbieten.



**Abbildung 8.10** Parameter abhängig vom GridView konfigurieren

Zusätzlich werden in der *SQLDataSource* der Bestellungen zwei SQL-Kommandos für *Delete* und *Insert* hinterlegt. Da das *GridView* keine *Insert*-Möglichkeit besitzt, muss das *DetailsView*-Steuerelement verwendet werden. Dies dient dann dazu, die Detaildaten anzuzeigen und – wenn nötig – daran Änderungen vorzunehmen. Um die beiden Datensteuerelemente nebeneinander darzustellen, wird eine HTML-Tabelle mit zwei Spalten verwendet.

```
<table>
  <tr>
    <td style="width: 100px">
      <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
AutoGenerateColumns="False"
      DataKeyNames="CustomerID" DataSourceID="DSKunden" CellPadding="4"
      ForeColor="#333333" GridLines="None">
        <Columns>
          <asp:CommandField ShowSelectButton="True" />
          <asp:BoundField DataField="CompanyName" HeaderText="CompanyName"
SortExpression="CompanyName" />
          <asp:BoundField DataField="City" HeaderText="City" SortExpression="City" />
          <asp:BoundField DataField="PostalCode" HeaderText="PostalCode"
SortExpression="PostalCode" />
        </Columns>
        <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <RowStyle BackColor="#EFF3FB" />
      </asp:GridView>
    </td>
  </tr>
</table>
```

**Listing 8.14** DetailsView verwendet GridView zur Steuerung

```

        <EditRowStyle BackColor="#2461BF" />
        <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
        <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
        <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <AlternatingRowStyle BackColor="White" />
    </asp:GridView>
</td>
<td style="width: 100px" valign="top">
    <asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"
        DataKeyNames="OrderID" DataSourceID="DSOrders" Height="50px"
        Width="125px" AllowPaging="True" CellPadding="4"
        ForeColor="#333333" GridLines="None">
        <Fields>
            <asp:BoundField DataField="OrderID" HeaderText="OrderID" InsertVisible="False"
                ReadOnly="True" SortExpression="OrderID" />
            <asp:BoundField DataField="OrderDate" HeaderText="OrderDate"
                SortExpression="OrderDate" />
            <asp:BoundField DataField="Freight" HeaderText="Freight"
                SortExpression="Freight" />
            <asp:BoundField DataField="ShipCity" HeaderText="ShipCity"
                SortExpression="ShipCity" />
            <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
                ShowInsertButton="True" />
        </Fields>
        <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <CommandRowStyle BackColor="#D1DDF1" Font-Bold="True" />
        <EditRowStyle BackColor="#2461BF" />
        <RowStyle BackColor="#EFF3FB" />
        <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
        <FieldHeaderStyle BackColor="#DEE8F5" Font-Bold="True" />
        <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
        <AlternatingRowStyle BackColor="White" />
    </asp:DetailsView>
</td>
</tr>
</table>
<asp:SqlDataSource ID="DSOrders" runat="server"
    ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
    DeleteCommand="DELETE FROM [Orders] WHERE [OrderID] = @OrderID"
    InsertCommand="INSERT INTO [Orders] ([OrderDate], [Freight], [ShipCity]) VALUES
        (@OrderDate, @Freight, @ShipCity)"
    SelectCommand="SELECT [OrderID], [OrderDate], [Freight], [ShipCity] FROM [Orders]
        WHERE ([CustomerID] = @CustomerID)"
    UpdateCommand="UPDATE [Orders] SET [OrderDate] = @OrderDate, [Freight] = @Freight,
        [ShipCity] = @ShipCity WHERE [OrderID] = @OrderID">
    <DeleteParameters>
        <asp:Parameter Name="OrderID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="OrderDate" Type="DateTime" />
        <asp:Parameter Name="Freight" Type="Decimal" />
        <asp:Parameter Name="ShipCity" Type="String" />
        <asp:Parameter Name="OrderID" Type="Int32" />
    </UpdateParameters>
    <SelectParameters>
        <asp:ControlParameter ControlID="GridView1" Name="CustomerID" PropertyName="SelectedValue"
            Type="String" />
    </SelectParameters>

```

**Listing 8.14** DetailsView verwendet GridView zur Steuerung (Fortsetzung)

```

    <InsertParameters>
      <asp:Parameter Name="OrderDate" Type="DateTime" />
      <asp:Parameter Name="Freight" Type="Decimal" />
      <asp:Parameter Name="ShipCity" Type="String" />
    </InsertParameters>
  </asp:SqlDataSource>
</asp:SqlDataSource ID="DSKunden" runat="server"
  ConnectionString="<%= $ ConnectionStrings:NorthwindConnectionString %>"
  SelectCommand="SELECT [CompanyName], [City], [PostalCode], [CustomerID] FROM
    [Customers]">
</asp:SqlDataSource>

```

**Listing 8.14** DetailsView verwendet GridView zur Steuerung (*Fortsetzung*)

Wieder kommt der Web-Entwickler ohne eine Zeile Quellcode aus. Allerdings ist es nach wie vor möglich, per Relation in einem DataSet eine Master-Detail-Darstellung zu realisieren.

## FormView

Das *FormView*-Steuerelement wird für frei gestaltbare Eingabeformulare verwendet. Neben *Darstellen*, *Einfügen*, *Bearbeiten* und *Löschen* werden aber auch die üblichen Funktionen aus dem *DetailsView* unterstützt. Mit dem *FormView*-Steuerelement wird immer nur ein Datensatz angezeigt.

Das *FormView*-Steuerelement arbeitet ausschließlich mit Templates. Pro Zustand gibt es ein Template, im Gegensatz zum *DetailsView*-Steuerelement, wo pro Spalte jeweils mehrere Templates existieren können. In der folgenden Tabelle werden diese Templates aufgelistet.

Templatetype	Verwendung
<i>EditItemTemplate</i>	Template für den Edit-Modus.
<i>EmptyDataTemplate</i>	Template zur Darstellung eines leeren Datensatzes.
<i>FooterTemplate</i>	Template zur Darstellung der Fußzeile.
<i>HeaderTemplate</i>	Template zu Darstellung des Kopfbereichs.
<i>InsertItemTemplate</i>	Template für den Insert-Modus.
<i>ItemTemplate</i>	Das Standard-Template zur Darstellung der Daten.
<i>PagerTemplate</i>	Template für die Gestaltung des Pager-Bereichs.

**Tabelle 8.15** Die Templates des FormView-Steuerelements

Um die Funktionen für *Bearbeiten*, *Einfügen* und *Löschen* zu aktivieren, müssen dafür entsprechende Buttons manuell angelegt werden. Dabei können in jedem Modus im dazugehörigen Template andere Buttons vorhanden sein.

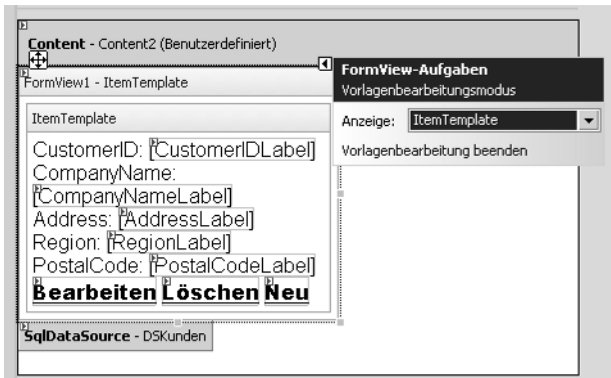
Damit die Buttons auch ohne Code funktionieren, müssen diese definierte Werte in dem *CommandName*-Attribut haben. Je nach gewünschter Funktion sind das *edit*, *cancel*, *update* oder *insert*.

Die Command Buttons müssen natürlich innerhalb des jeweiligen Templates platziert werden.

```
<asp:LinkButton ID="LinkButton2" Runat="server" CommandName="edit">Edit</asp:LinkButton>
```

Die Templates können in der Entwicklungsumgebung visuell bearbeitet werden. Dabei wird jedes Template anders dargestellt. Den Template-Modus wählt man aus dem *FormView-Aufgaben*-menü über *mit Vorlagen bearbeiten* aus. In das Template ziehen Sie aus der Werkzeugleiste ein *Button*-, *HyperLink*- oder *ImageButton*-Steuerelement.

Nach Beendigung muss der Menüpunkt *Vorlagenbearbeitung beenden* gewählt werden.



**Abbildung 8.11** Das ItemTemplate in der Entwicklungsumgebung

Der HTML-Code in der ASPX-Seite für das *FormView*-Steuerelement wird dabei recht umfangreich, da die komplette Darstellung manuell entworfen werden muss. In dieser Hinsicht drängt sich der Vergleich mit dem *Repeater*-Steuerelement auf.

Mit Attributen wird das *FormView*-Steuerelement gesteuert.

```
<asp:FormView ID="FormView1" Runat="server" DataSourceID="SqlDataSource1" DataKeyNames="ProductID"
    AllowPaging="True" BorderStyle="None" BackColor="White" ForeColor="Black"
    BorderWidth="1px"
    GridLines="Vertical" BorderColor="#DEDFDE" CellPadding="4">
  <EditItemTemplate>
    ProductID: <asp:Label Text='<%=# Eval("ProductID") %>' Runat="server" ID="ProductIDLabel1">
      </asp:Label><br />
    ProductName: <asp:TextBox Text='<%=# Bind("ProductName") %>' Runat="server"
      ID="ProductNameTextBox"></asp:TextBox><br />
    ... ReorderLevel: <asp:TextBox Text='<%=# Bind("ReorderLevel") %>' Runat="server"
      ID="ReorderLevelTextBox"></asp:TextBox><br />
    Discontinued: <asp:CheckBox Checked='<%=# Bind("Discontinued") %>' Runat="server"
      ID="DiscontinuedCheckBox" /><br />
  </EditItemTemplate>...
```

**Listing 8.15** FormView-Steuerelement

Auch Paging ist mit dem *FormView*-Steuerelement möglich. Dazu reicht es aus, einen Haken in die Option *Enable Paging* zu setzen. Alles weitere funktioniert vollautomatisch.

```
AllowPaging="True"
```

Aber auch manuell lässt sich der Pager entwerfen. Dazu werden Steuerelemente zu seiner Steuerung in das *PagerTemplate* gezogen. Den Steuerelementen muss die Eigenschaft *CommandName* zugewiesen werden. Je nach eingestelltem Wert *Next*, *Prev*, *Last* oder *First*, hat dieser Button dann die entsprechende Funktion.



```
<PagerTemplate>
  <asp:Button ID="Next" runat="server" Text="Next" CommandName="next"/>
  ..

```

**Listing 8.16** Pager selbst entwerfen

## XML binden

Was ist eigentlich, wenn die Daten aus einer XML-Datei kommen? Dann bietet sich ein *XmlDataSource*-Steuerelement zur Verwendung an. Aber auch die Datenbindung sieht dann im Detail anders aus. Es wird nicht mehr über *Bind* oder *Eval* gebunden, da ja keine Felder im eigentlichen Sinn vorhanden sind.

XML-Abfragen werden mittels XPath durchgeführt. Entsprechend kommt auch der *XPathBinder* zum Einsatz.

```
<asp:XmlDataSource ID="XmlDataSource1" Runat="server" DataFile="~/app_Data/people.xml">
</asp:XmlDataSource>
<asp:FormView ID="FormView1" Runat="server" DataSourceID="XmlDataSource1" AllowPaging="True">
<ItemTemplate>
<asp:Label ID="Label1" Runat="server" Text='<%# XPath("Name/FirstName") %>'></asp:Label><br />
<asp:Label ID="Label2" Runat="server" Text='<%# XPath("Address/City") %>'></asp:Label><br />
</ItemTemplate>
</asp:FormView>

```

**Listing 8.17** XML-Daten werden im FormView-Steuerelement dargestellt

In diesem kleinen Beispiel ist sogar das Paging bereits voll funktionsfähig. Allerdings ist es so nicht möglich geänderte Daten per Update zu schreiben.

## DataGrid

Das *DataGrid*-Steuerelement ist eigentlich überflüssig, da das *GridView*-Steuerelement die Funktion übernimmt. So findet man in Visual Studio dieses Steuerelement nicht mehr in der Werkzeugleiste. In der Summe ist dies vor allem für die weitere Verwendung von bestehen ASPX-Seiten nützlich. Das wahrscheinlich wichtigste neue Attribut ist *DataSourceID*, mit dem an die neuen Datensteuerelemente gebunden werden kann.

```
<asp:DataGrid ID="datagrid1" Runat="Server" AutoGenerateColumns="true" AllowPaging="true" _
  DataSourceID="SqlDataSource1">
</asp:DataGrid>

```

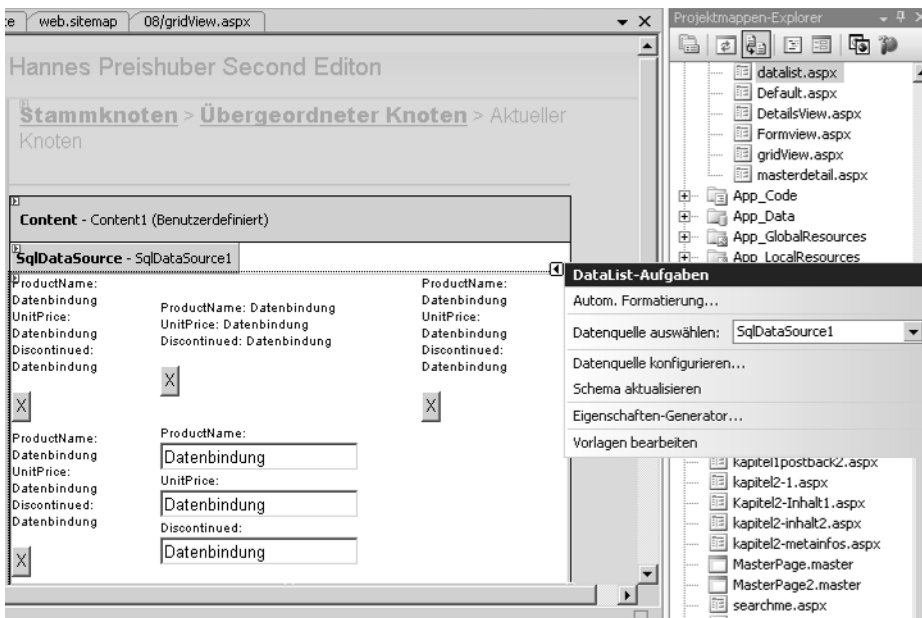
**Listing 8.18** DataGrid Webserver-Steuerelement.

Nach wie vor muss aber für die erweiterten Funktionen wie Paging oder Sorting manuell Code implementiert werden. Wenn dieser fehlt, kann man im Browser auch nicht weiterblättern. Die Pager-Steuerung wird allerdings angezeigt.

# DataList

Das DataList-Steuerelement ist seit ASP.NET 1.0 vorhanden und wird auch weiterhin eingesetzt. Der größte Vorteil der DataList ist, dass in einer Zeile mehrere Datensätze dargestellt werden können.

Auch das *DataList*-Steuerelement kann jetzt mit dem Attribut *DataSourceID* an eine DataSource gebunden werden. Mit dem Menüpunkt *Schema aktualisieren* aus dem *DataList-Aufgabenmenü* werden auch gleich die Spalten erzeugt. Der visuelle Designer ist damit wesentlich mächtiger geworden, als in früheren Versionen von Visual Studio.



**Abbildung 8.12** DataList wird in der Entwicklungsumgebung gestaltet

Mit dem Attribut *RepeatColumns* wird angegeben, wie viele Datensätze in einer Zeile wiederholt werden sollen. Auch die Ausrichtung kann mit dem Attribut *RepeatDirection* über die Werte *Horizontal* und *Vertical* gesteuert werden. Mit dem Attribut *RepeatLayout* kann zwischen der Table- und Flow-Darstellung gewechselt werden. Diese Einstellungen finden Sie im Eigenschaftsdialog oder direkt im Code in der ASPX-Seite mit IntelliSense.

Die Darstellung wird im *ItemTemplate* mit HTML-Elementen und ASP.NET-Steuerelementen erledigt. Dabei können über die Datenbindung mit `<# ... %>` Tabelleninhalte ausgegeben werden. Auch zur Steuerung von anderen Attributen – wie beispielsweise für ein Bild die *ImageSource* – kann ein Datenfeld herangezogen werden.

```
<asp:DataList ID="DataList1" Runat="server" DataSourceID="SqlDataSource1" RepeatColumns="3"
    RepeatDirection="Horizontal" RepeatLayout="Table">
  <ItemTemplate>
    ProductName: <asp:Label Text='<%# Eval("ProductName") %>' Runat="server"
        ID="ProductNameLabel"></asp:Label><br />
    UnitPrice:
    <asp:Label Text='<%# Eval("UnitPrice") %>' Runat="server" ID="UnitPriceLabel"></asp:Label>
    <br />Discontinued:
    <asp:Label Text='<%# Eval("Discontinued") %>' Runat="server"
        ID="DiscontinuedLabel"></asp:Label>
    <br /><br />
  </ItemTemplate>
  <ItemStyle ForeColor="Navy" Font-Names="Arial" Font-Size="X-Small"></ItemStyle>
</asp:DataList>
```

**Listing 8.19** DataList-Steuerelement im Einsatz

Wird *RepeatLayout* nicht angegeben, wird der Wert *Table* verwendet. Die Ausgabe im Browser erfolgt dann mit HTML-Table-Elementen.

Die *DataList* kann auch in einen Bearbeiten-Modus versetzt werden. Um die Darstellung dafür und für andere Zwecke zu steuern, gibt es noch weitere Templates. Diese können auch in der Entwicklungsumgebung visuell editiert werden. Die Templates wurden bereits im vorhergehenden Teil beschrieben. Es stehen zur Verfügung:

- *AlternatingItemTemplate*
- *EditItemTemplate*
- *FooterTemplate*
- *HeaderTemplate*
- *ItemTemplate*
- *SelectedItemTemplate*
- *SeparatorTemplate*

Das *DataList*-Steuerelement kann somit auch für *Bearbeiten*-, *Auswählen*- und *Löschen*-Aktionen verwendet werden.

Die Buttons für die Aktionen müssen manuell aus der Werkzeuggeste eingfügt werden. Die Steuerung, welche Aktion dann letztendlich durch den Button ausgeführt wird, erfolgt mit dem Attribut *CommandName*. Diese müssen entsprechend mit *edit*, *delete* oder *select* benannt werden.

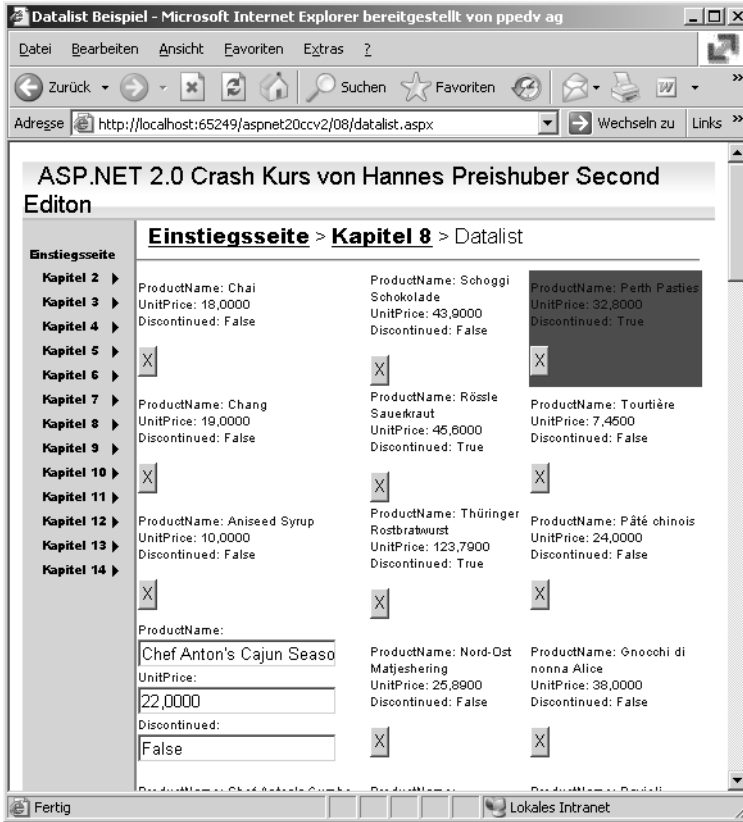


Abbildung 8.13 Select und Edit im DataList-Control

Daten-Paging ist mit diesem Steuerelement nicht direkt möglich und muss manuell implementiert werden.

## Repeater

Wenn es darum geht, volle Kontrolle über den erzeugten HTML-Code zu erhalten, führt kein Weg am *Repeater*-Steuerelement vorbei. Auch dieses Steuerelement ist bereits seit ASP.NET 1 vorhanden, wurde aber auch um das Attribut *DataSourceID* erweitert. Das folgende Beispiel zeigt die Verwendung eines Repeaters.

In diesem Beispiel werden Artikel dargestellt. Sobald ein Artikel nicht mehr im Sortiment ist, soll er durchgestrichen werden.

Das Tabellenfeld *discontinued* enthält die dafür nötige Information als booleschen Wert. Dieses wird nicht direkt gebunden. Stattdessen wird die Funktion *Strike* aufgerufen, die je nach Feldinhalt einen Text zurückliefert. Der Text wird in das *Style*-Attribut eines *Font*-Elements eingebaut und formatiert dann die Browser-Ausgabe. Die Datenbindung erfolgt mit dem Ausdruck *Eval*.

```
<script runat="server">
Function Strike(ByVal aktiv As String) As String
    If aktiv = "False" Then
        Return ""
    End If
End Function
```

Listing 8.20 Repeater wird an DataSource gebunden

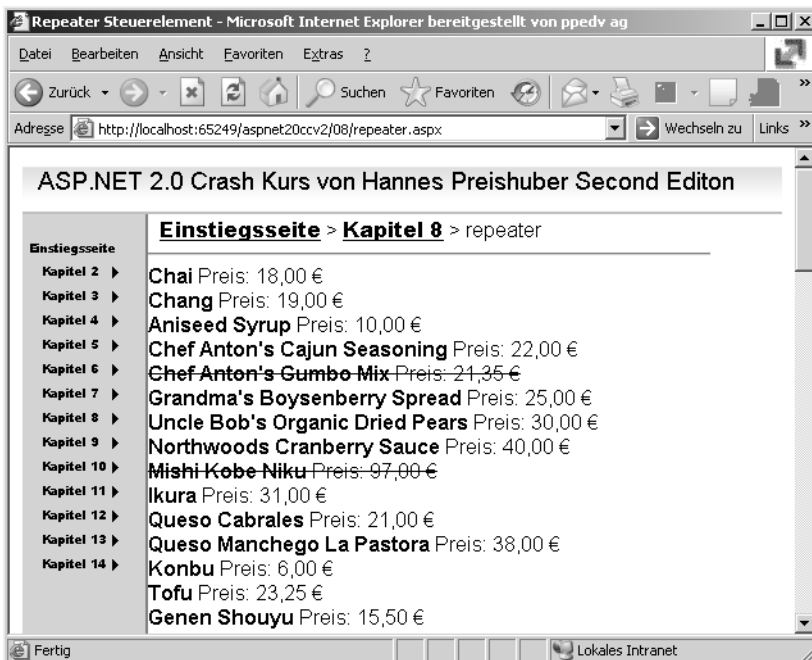
```

Else
    Return "text-decoration: line-through"
End If
End Function
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <asp:Repeater ID="Repeater1" Runat="server" DataSourceID="SqlDataSource1">
        <ItemTemplate>
            <font style='<%=#Strike(Eval("discontinued"))%>' >
                <b><%=eval("productname") %></b> Preis:
            <%=Eval("unitprice", "{0:c}")%></font>
<br />
        </ItemTemplate>
    </asp:Repeater>
    <asp:SqlDataSource ID="SqlDataSource1" Runat="server" ProviderName="System.Data.SqlClient"
        ConnectionString="<=%$ ConnectionStrings:AppConnectionString1 %>" SelectCommand="SELECT_
            [ProductName], [UnitPrice], [Discontinued] FROM [Products]">
    </asp:SqlDataSource>

```

**Listing 8.20** Repeater wird an DataSource gebunden (Fortsetzung)

Wer sich bisher die Daten-Steuerelemente angesehen hat, sollte keine Probleme mit dem Verstehen des Code-Beispiels haben. Achten Sie darauf, dass selbst der Zeilenumbruch nach dem Datensatz manuell erzeugt werden muss. Entweder Sie packen alles in eine HTML-Tabelle und beenden einfach die *TableRow* nach jedem Datensatz, oder Sie verwenden ein `<BR>`-Tag.



**Abbildung 8.14** Durchgestrichene Einträge mit Repeater

Der Repeater bietet auch Ereignisse an, wobei hier nur kurz auf *ItemCommand* hingewiesen sei. Damit kann man Ereignisse der eingebauten Buttons verarbeiten.

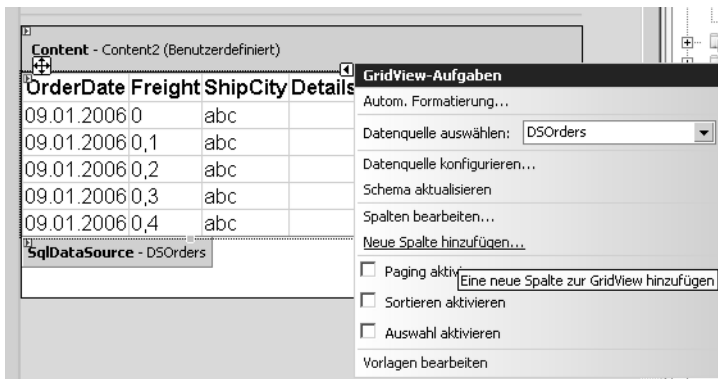
## Verschachtelte Steuerelemente

Zum Abschluss dieses Kapitels beschäftigen wir uns noch mit dem Verschachteln zweier Daten-Steuerelemente. Dies ist eine Möglichkeit, die im Windows-Umfeld so nicht vorhanden ist. Mit ASP.NET 2.0 lassen sich Verschachtelungen von zwei Webserver-Steuerelementen auf unterschiedliche Art durchführen. So kann in ein Bearbeiten-Template eine DropDown-Liste zur Wertauswahl eingebunden werden oder die Master-Detail-Darstellung innerhalb eines *GridView*-Steuerelements verschachtelt werden.

Genau Letzteres wird im folgenden Beispiel durchgeführt.

Ein *GridView*-Steuerelement zeigt Bestellungen und enthält gleichzeitig die Bestellpositionen. Da eine Bestellung mehrere Bestellpositionen beinhalten kann, ist dies nicht direkt möglich.

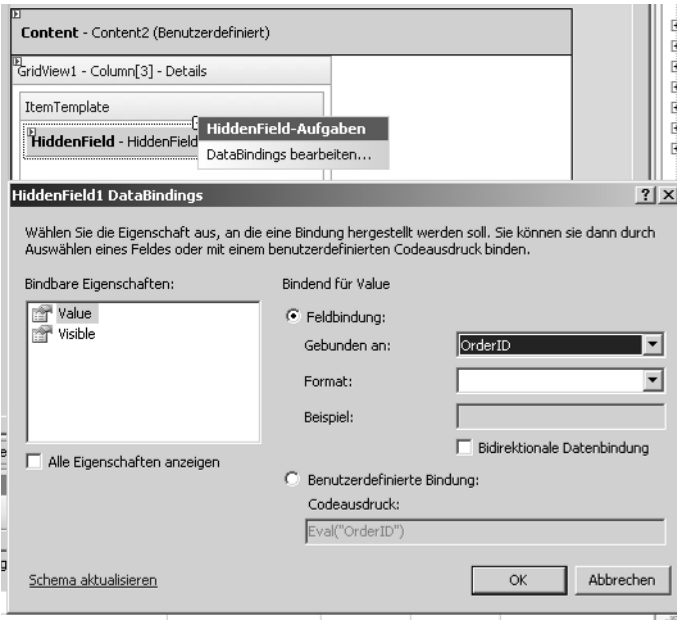
Dazu erstellen Sie zuerst ein *GridView*-Steuerelement, das die Bestellungen enthält. Fügen Sie eine Vorlagen-spalte hinzu. Diese wird später die Bestelldetails enthalten.



**Abbildung 8.15** Alle Bestellungen im GridView-Steuerelement

Im nächsten Schritt wählen sie aus dem *GridView-Aufgabenmenü* den Punkt *Vorlagen bearbeiten*. Damit wird die letzte Spalte »Details« in den Template Modus versetzt, so dass Sie diese direkt bearbeiten können. Natürlich lässt sich dies auch alles in der Quellansicht durchführen.

Ziehen Sie nun aus der Werkzeugleiste ein *HiddenField*-Steuerelement in das Template. Wählen Sie aus dem Aufgabenmenü des *HiddenField*-Steuerelements *DataBindings bearbeiten*. Dann können Sie im *DataBindings*-Dialog die Feldbindung zu *OrderID* aktivieren. Diese ID verwenden wir gleich zur Steuerung einer zweiten *SQLDataSource*, die zum Ermitteln der Detail-Datensätze dient. Das funktioniert aber nur, wenn sich die Steuerelemente in der gleichen Hierarchieebene, hier im Template, befinden.



**Abbildung 8.16** HiddenField-Datenbindung bearbeiten

Als Nächstes ziehen Sie ein *Repeater*-Steuerelement in das *ItemTemplate* und erzeugen eine neue Datenquelle. Dies konfigurieren Sie so, dass bei der Auswahl der *WHERE*-Bedingung das HiddenField als *Control-Parameter* dient. Am Ende muss Ihr erzeugter Code in der ASPX-Seite so aussehen.

```
<asp:TemplateField HeaderText="Details">
  <ItemTemplate>
    <asp:HiddenField ID="HiddenField1" runat="server" Value='<%# Eval("OrderID") %>' />
    <asp:Repeater ID="Repeater1" runat="server" DataSourceID="DSDetails"></asp:Repeater>
    <asp:SqlDataSource ID="DSDetails" runat="server"
      ConnectionString="<%= $ ConnectionStrings:NorthwindConnectionString %>"
      SelectCommand="SELECT [ProductID], [UnitPrice], [Quantity] FROM [Order Details]
        WHERE ([OrderID] = @OrderID)">
      <SelectParameters>
        <asp:ControlParameter ControlID="HiddenField1" Name="OrderID" PropertyName="Value"
          Type="Int32" />
      </SelectParameters>
    </asp:SqlDataSource>
  </ItemTemplate>
</asp:TemplateField>
```

**Listing 8.21** Die Detail-Spalte aus der Bestellsicht

Nun müssen Sie nur noch in das *Repeater*-Steuerelement die Felder einfügen, die darzustellen sind. Um eine bessere Ansicht zu bekommen, wird noch eine HTML-Tabelle pro Datensatz erzeugt. Dazu wird das *HeaderTemplate* und *FooterTemplate* mit einem öffnenden und einem schließenden HTML-Table-Element versehen. Im *ItemTemplate* des *Repeater*-Steuerelements werden dann *<TR>* und *<TD>* Elemente für die Reihen und Zellen implementiert.

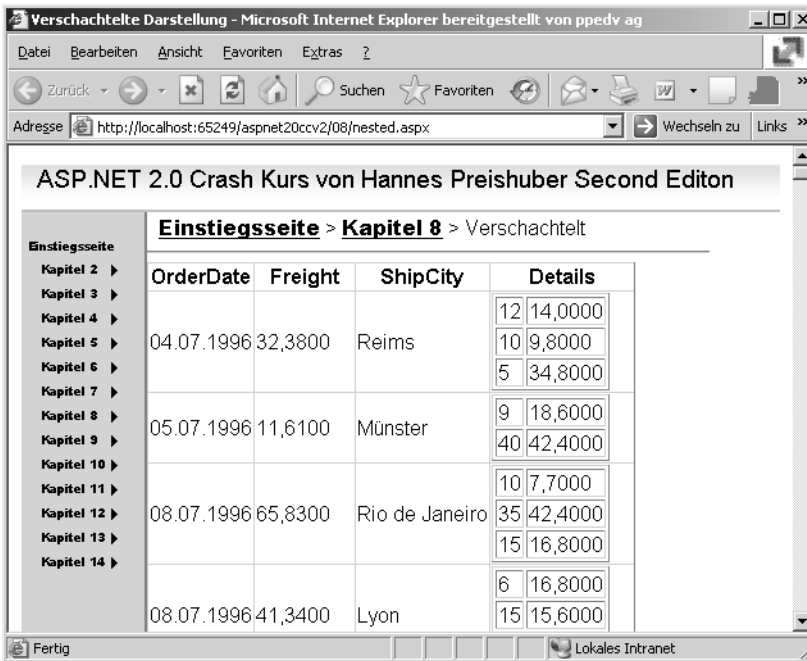
```

<asp:Repeater ID="Repeater1" runat="server" DataSourceID="DSDetails">
  <HeaderTemplate>
  <table border="1">
  </HeaderTemplate>
  <FooterTemplate>
  </table>
  </FooterTemplate>
  <ItemTemplate>
  <tr>
  <td><%=#Eval("quantity")%></td>
  <td><%=#Eval("unitprice")%></td>
  </tr>
  </ItemTemplate>
</asp:Repeater>

```

**Listing 8.22** Die Darstellung der einzelnen Bestellpositionen

Am Ende soll natürlich das Endergebnis noch gezeigt werden. Wieder wurde eine Aufgabe völlig ohne Code in kürzester Zeit gelöst.



**Abbildung 8.17** Verschachtelte Master-Detail-Darstellung



## Kapitel 9

# Trace, Debug und Config

### **In diesem Kapitel:**

Ablaufverfolgung	250
Debug	254
Fehler aufzeichnen	256
IIS-Konfiguration	257
Websiteverwaltungs-Tool	258
Konfigurations-API	261
Kompilierung	263
Health Monitoring	265

Um es mit Ironie zu sagen: Nur Schwächlinge machen Fehler. Natürlich sind wir keine Schwächlinge und machen deshalb keine Fehler. Damit ist Debuggen für uns selbstverständlich überflüssig. Für alle anderen nicht so perfekten Entwickler bringt ASP.NET allerdings einiges, was das Leben beim Fehler suchen leichter machen kann.

Fehlersuche ist nicht nur debuggen. Es stehen auch andere Methoden zur Fehlersuche zur Verfügung. Wenn nur eine Webanwendung auf einem Server läuft, ist es noch ziemlich einfach, diese zu konfigurieren und zu überwachen. Aber speziell in Hosting-Umgebungen ließ ASP.NET bislang einige Wünsche bezüglich Überwachung und Diagnose offen.

## Ablaufverfolgung

Ablaufverfolgung auch Tracing genannt, wurde schon in ASP.NET 1 eingeführt. Damit kann ziemlich detailliert verfolgt werden, was beim Seitenaufbau vor sich geht. Zusätzlich kann man im Code per `Trace.Write` und `Trace.Warn` selbst in das Trace-Fenster schreiben.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Trace.Warn("Warnung", "Das ist die Warnung")
    Trace.Write("Write", "Das ist per Trace.write geschrieben")
End Sub
```

**Listing 9.1** In den Trace schreiben

Alternativ kann man auch selbst einen mithörenden Trace-Listener schreiben, der diese Ausgaben umleitet. Dabei ist diese Trace-Ausgabe nur aktiv, wenn das *Tracing* aktiviert ist.

## Page Tracing

Um die Ablaufverfolgung für eine einzelne ASPX-Seite zu aktivieren, setzen Sie die entsprechenden Einstellungen `Trace=true` in der *Page*-Direktive.

```
<%@ Page Language="VB" MasterPageFile="~/masterpage2.master" Title="Trace" Trace="true" %>
```

Beim so genannten Page Tracing wird die Trace-Ausgabe unten an die erzeugte HTML-Seite angehängt.

Die Ablaufverfolgung liefert umfangreiche Informationen über die Steuerelement-Hierarchie, deren jeweilige benötigte Größe in der Seite und die zeitliche Abfolge. Speziell Letzteres ist wichtig um Flaschenhalse, die die Anwendung bremsen, ausfindig zu machen. Wichtig ist auch, dass die Codezeilen, die sich auf Trace beziehen, eben nur ausgeführt werden, wenn Ihre Anwendung im Trace-Modus ist. Dies können Sie auch mit einer Webanwendung machen, die live auf einem Webserver läuft, ohne zusätzlich z.B. einen Debugger installieren zu müssen.

The screenshot shows a browser window with the address `http://localhost:65249/aspnet20ccv2/09/trace.aspx`. The main content is a list of trace events:

Event	Start Time	End Time	Duration
aspx.page End PreLoad	0,0035096639377351		0,000031
aspx.page Begin Load	0,00352866076554422		0,000019
Warnung Das ist die Warnung	0,00397005764699145		0,000441
Write Das ist per Trace.write geschrieben	0,00399464177709737		0,000025
aspx.page End Load	0,00402872432110785		0,000034
aspx.page Begin LoadComplete	0,00404827987914665		0,000020
aspx.page End LoadComplete	0,00406615924649641		0,000018
aspx.page Begin PreRender	0,00408320051850165		0,000017
aspx.page End PreRender	0,005421918148815		0,001339
aspx.page Begin PreRenderComplete	0,00545600069282548		0,000034
aspx.page End PreRenderComplete	0,00547527688574945		0,000019
aspx.page Begin SaveState	0,0479055298927657		0,042430
aspx.page End SaveState	0,0702346247916984		0,022329
aspx.page Begin SaveStateComplete	0,0702804406705321		0,000046
aspx.page End SaveStateComplete	0,0702997168634561		0,000019
aspx.page Begin Render	0,0703175962308059		0,000018
aspx.page End Render	0,0750204285740227		0,004703

Below the log is a table titled 'Steuerelementstruktur' (Control Structure):

UniqueID des Steuerelements	Typ	Wiedergabegröße in Bytes (einschl. untergeordnete)	ViewState-Größe in Bytes (ausschl. untergeordnete)	ControlState-Größe in Bytes (ausschl. untergeordnet)
__Page	ASP_09_trace.aspx	67103	0	0
ctl00	ASP.masterpage2_master	67103	0	0
ctl00\$ctl00	System.Web.UI.LiteralControl	127	0	0
ctl00\$ctl01	System.Web.UI.LiteralControl	49	0	0
ctl00\$kopf	System.Web.UI.HtmlControls.HtmlHead	975	0	0
ctl00\$css1	System.Web.UI.HtmlControls.HtmlLink	78	0	0
ctl00\$Metainfos	System.Web.UI.WebControls.ContentPlaceHolder2	2	0	0
ctl00\$Metainfos\$ctl00	System.Web.UI.LiteralControl	2	0	0
ctl00\$ctl02	System.Web.UI.LiteralControl	14	0	0
aspnetForm	System.Web.UI.HtmlControls.HtmlForm	65918	0	0
ctl00\$ctl03	System.Web.UI.ResourceBasedLiteralControl	575	0	0
ctl00\$Menu1	System.Web.UI.WebControls.Menu	41727	11852	56
ctl00\$ctl04	System.Web.UI.LiteralControl	30	0	0
ctl00\$SiteMapDataSource1	System.Web.UI.WebControls.SiteMapDataSource0	0	0	0
ctl00\$ctl05	System.Web.UI.ResourceBasedLiteralControl	274	0	0

Abbildung 9.1 Trace in Seite ausgeben lassen

## Ablaufverfolgung auf Anwendungsebene

Um die Ablaufverfolgung in der `web.config` für alle Seiten der Anwendung zu aktivieren, wird das Trace-Element verwendet. Den Inhalt des so genannten *Trace Stacks* ruft man im Browser mithilfe der Seite `Trace.axd` auf. Darin sind dann auch mehrere Traces enthalten.

In dem Element `Trace` kann mit dem Attribut `enabled` das Tracing an- und ausgeschaltet werden. Aus Sicherheitsgründen sollte die Ausgabe des Traces nur lokal erreichbar sein. Das erreicht man mit dem Attribut `localOnly`. In der Version 1.x von ASP.NET stoppte das Tracing, wenn die in `requestLimit` eingestellte Anzahl von Requests erreicht war. Mit dem neu hinzugekommenen Attribut `mostRecent` stehen immer die aktuellsten Traces zum Abruf bereit. Natürlich muss dafür der entsprechende Wert auf `true` gesetzt sein.

```
<trace
enabled="true"
pageOutput="false"
localOnly="true"
mostRecent="true"
requestLimit="50"
traceMode="SortByTime"
writeToDiagnosticsTrace="true" />
```

Listing 9.2 Konfiguration von Trace in `web.config`

http://localhost:65249/aspnet20ccv2/09/trace.axd - Microsoft Internet Explorer bereitgestellt von ppedv ag

Adresse <http://localhost:65249/aspnet20ccv2/09/trace.axd>

## Anwendungsüberwachung

### aspnet20ccv2

[ [Aktuelle Nachverfolgung löschen](#) ]  
 Physikalisches Verzeichnis: D:\WebSites\aspnet20ccv2\

Anforderungen an die Anwendung						Verbleibend: 0
Nr.	Anforderungszeitpunkt	Datei	Statuscode	Verb		
1	09.01.2006 19:25:06	/style2.css	200	GET	<a href="#">Details anzeigen</a>	
2	09.01.2006 19:25:07	/09/trace.aspx	200	POST	<a href="#">Details anzeigen</a>	
3	09.01.2006 19:25:07	/style2.css	200	GET	<a href="#">Details anzeigen</a>	
4	09.01.2006 19:25:07	/WebResource.axd	200	GET	<a href="#">Details anzeigen</a>	
5	09.01.2006 19:25:07	/WebResource.axd	200	GET	<a href="#">Details anzeigen</a>	
6	09.01.2006 19:25:07	/WebResource.axd	200	GET	<a href="#">Details anzeigen</a>	
7	09.01.2006 19:25:07	/WebResource.axd	200	GET	<a href="#">Details anzeigen</a>	
8	09.01.2006 19:25:07	/WebResource.axd	200	GET	<a href="#">Details anzeigen</a>	
9	09.01.2006 19:25:07	/WebResource.axd	200	GET	<a href="#">Details anzeigen</a>	
10	09.01.2006 19:25:12	/09/xxx.aspx	404	GET	<a href="#">Details anzeigen</a>	

Microsoft .NET Framework-Version: 2.0.50727.42; ASP.NET-Version: 2.0.50727.42

Lokales Intranet

Abbildung 9.2 Application Tracing

Wenn in der Page-Deklaration `Trace=false` definiert ist, wird diese Seite generell aus dem Tracing ausgeschlossen.

## Trace aus Objekten

Wenn Ihr Code in Businessobjekten ausgelagert ist, Sie aber dennoch Trace-Informationen benötigen, kommt der Namespace `System.Diagnostics` zum Einsatz. Dieser beinhaltet einen eigenen Trace Stack.

Darin enthalten ist wiederum ein `Trace`-Objekt mit einer `Write` Funktion. Diese ist genau umgekehrt parametrisiert wie der normale Page Trace. Auch eine Warn-Funktion, allerdings mit dem Namen `TraceWarning` ist vorhanden.

Unabhängig davon funktioniert das Schreiben in den normalen Trace mithilfe des `HttpContext.Current`-Objekts. Das folgende Beispiel erzeugt eine `shared` (VB) oder auch `static` (C#) Funktion, die ohne Instanziierung verwendet werden kann und in den Trace schreibt.

```
Public Class tracesample
    Public Shared Sub nop()
        System.Diagnostics.Trace.Write("Nix passiert ausser System.diagnostic.trace.write", "NOP")
        HttpContext.Current.Trace.Write("NOP2", "nix passiert2")
        System.Diagnostics.Trace.TraceWarning("Warnung")
    End Sub
End Class
```

Um die Trace-Meldungen von *System.Diagnostics* aus dem Objekt weiterzuleiten, muss mit dem Compiler-Switch *Trace* kompiliert werden. Diese Einstellung kann in der *web.config* im Bereich *System.CodeDom* vorgenommen werden. Weiterhin muss im Bereich *System.Diagnostics* der *TraceListener* auf den *WebPageTraceListener* umgeleitet werden, der auch die Page-Trace-Informationen abhandelt.

```
<system.codedom>
  <compilers>
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
      type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.3600.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" warningLevel="1" compilerOptions="/d:TRACE" />
  </compilers>
</system.codedom>
<system.diagnostics>
<trace autoflush="true" indentsize="4">
  <listeners>
    <add name="webListener" type="System.Web.WebPageTraceListener, System.Web, Version=2.0.3500.0,
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
  </listeners>
</trace>
</system.diagnostics>
```

**Listing 9.3** Listing 9.3: Compiler-Anweisung in *web.config*, um Trace zu aktivieren

In der ASPX-Seite wird dann in den Trace Stack geschrieben und die Funktion NOP (für *no operation*) aufgerufen.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
  Trace.Warn("Warnung", "Das ist die Warnung")
  tracesample.nop()
  Trace.Write("Write", "Das ist per Trace.write geschrieben")
End Sub
```

**Listing 9.4** In der ASPX-Seite wird der Trace beschrieben

Nur wenn dies so komplett konfiguriert ist, werden auch die *System.Diagnostic*-Meldungen per *Trace.axd* im Browser angezeigt.

Event Name	Message	Time
aspx.page	begin load	0,16007448712057
Warnung	Das ist die Warnung	0,188263871525571
NOP	Nix passiert ausser System.diagnostics.trace.write	0,190735414696561
NOP2	nix passiert2	0,191061992515809
WebDev.WebServer.EXE	Ereignis 0: Warnung	0,215534655940909
Write	Das ist per Trace.write geschrieben	0,215694452786597
aspx.page	End Load	0,216622723126872

**Abbildung 9.3** Drei Trace-Meldungen kommen aus dem Geschäftsobjekt

## Trace-Nachrichten routen

Mit dem Attribut *writeToDiagnosticsTrace* in der *web.config* können *Trace*-Nachrichten, die vom *System.Diagnostics.Trace* erzeugt werden, weiter geroutet werden. So können aus externen Code-Objekten *Trace*-Nachrichten auch in einen externen *TraceListener* umgeleitet werden.

In der ASPX-Seite wird mit dem *WebPageTraceListener*, wenn alle *Trace*-Nachrichten eingegangen sind, das Ereignis *TraceFinished* ausgelöst. Anschließend kann dieses in der Seite behandelt und dort z.B. die Auflistung der *Trace*-Nachrichten ausgegeben werden. Aber auch ein Weiterreichen an andere Logging-Mechanismen ist denkbar.

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Trace.Warn("MeinTrace", "Start")
    Trace.Write("MeinTrace", "Ende")
    AddHandler Trace.TraceFinished, AddressOf Me.OnTraceFinished
End Sub
Sub OnTraceFinished(ByVal sender As Object, ByVal e As TraceContextEventArgs)
    Response.Write("<h1>Trace ausgeben</h1>")
    For Each tr As TraceContextRecord In e.TraceRecords
        Response.Write(String.Format("<b>{0}</b> {1}<br />", tr.Category, tr.Message))
    Next
End Sub
```

**Listing 9.5** Trace per Code ausgeben

Die folgende Auflistung zeigt die in *System.Diagnostics* vorhandenen *TraceListener*.

- *FileLogTraceListener*
- *DefaultTraceListener*
- *EventLogTraceListener*
- *TextWriterTraceListener*
- *ConsoleTraceListener*
- *XMLWriterTraceListener*
- *DelimetedListTraceListener*
- *WebPageTraceListener*

## Debug

Der visuelle Debugger von Visual Studio wartet wieder mit einer Menge Verbesserungen auf: Nicht nur *Edit* & *Continue* funktionieren wieder so, wie schon in Visual Basic 6.0, sondern auch das Auslesen von Variablen während des Debuggings ist sehr einfach geworden. Es genügt, mit der Maus auf eine Variable zu zeigen, um dann durch die Objektstruktur zu navigieren. Sogar einzelne Variablen kann man sofort ändern, egal ob es sich um eine einfache String-Variablen oder um eine komplexe XML-Struktur handelt.

Der Debugger startet automatisch, wenn man in einer ASPX-Seite **F5** drückt. Es wird dann ein neues Browserfenster geöffnet. Haltepunkte werden mit **F9** gesetzt. Falls die Anwendung noch nie im Debug-Modus gestartet wurde, kann es sein, dass von Visual Studio eine Rückfrage kommt, ob eine Änderung in der *web.config* vorgenommen werden darf. Für das Debuggen muss folgender Eintrag vorhanden sein.

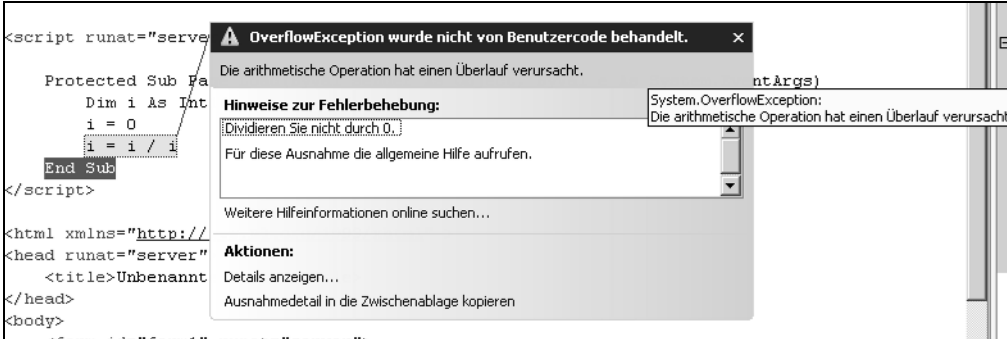


Abbildung 9.4 Der Debugger von Visual Studio 2005

```
<compilation debug="true" />
```

Man kann auch einen aktuellen Prozess an den Debugger anhängen. Dazu wählen Sie im Menü *Debuggen* den Punkt *Prozess anhängen* aus. Diese Funktion ist in der Express-Edition nicht verfügbar.

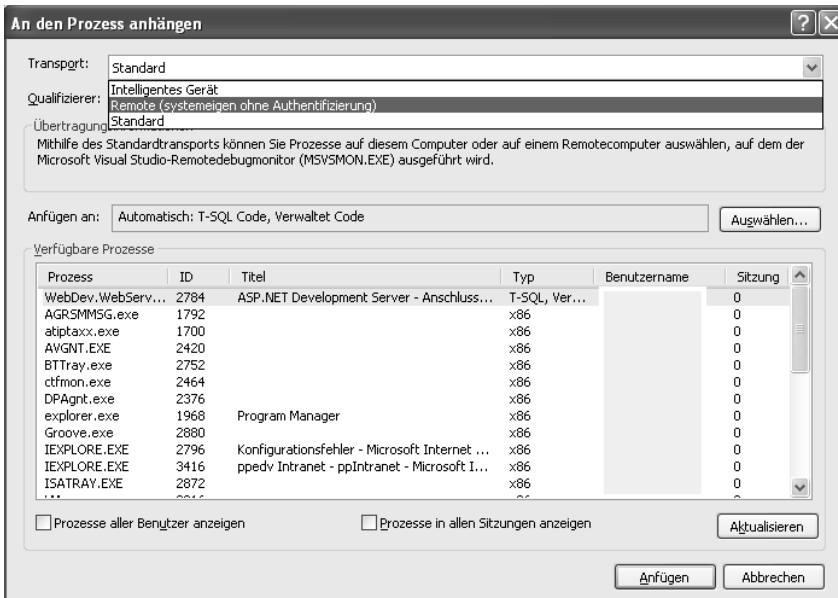


Abbildung 9.5 Debug-Prozesse überwachen

**HINWEIS** Häufig werden Fehler beim Debuggen dadurch verursacht, dass das Konto des aktuell angemeldeten Benutzers nicht Mitglied der Debugger-Gruppe ist. Außerdem muss das Debuggen generell aktiviert sein. Überprüfen Sie dies im Kontextmenü des Projektes im Projektmappen-Explorer mit dem Menüpunkt *Eigenschaftsseiten*. In *Startoptionen* muss die Checkbox bei *Debugger ASP.NET* gesetzt sein.

Es muss darüber hinaus in der *Page*-Direktive das Debugging für die Seite mit dem Attribut *Debug* aktiviert sein. Wenn das Attribut fehlt, ist der Standardwert *false*, das Debugging also deaktiviert.

```
<%@ Page Language="VB" Debug="true"%>
```

## Ausnahmen

Die Fehlerbehandlung ist mithilfe des *Try-Catch*-Konstruktes nun sehr einfach und übersichtlich möglich. Unbehandelte Fehler lösen eine so genannte Exception aus. Diese lassen sich während des Debuggens nun sehr komfortabel lesen und lösen.

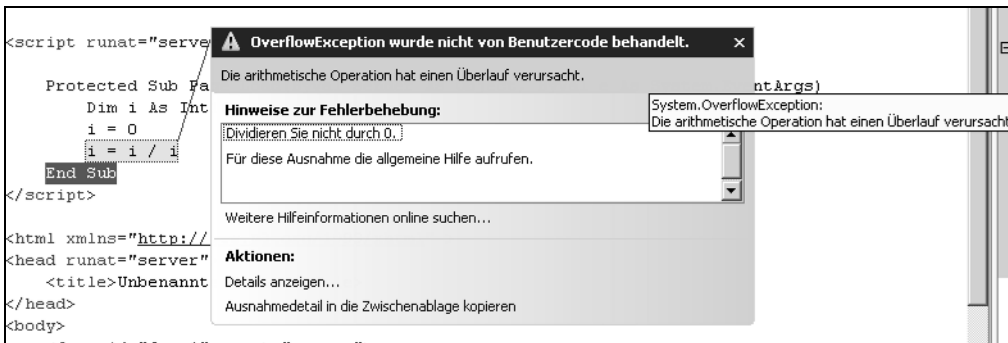


Abbildung 9.6 Unbehandelte Ausnahme

## Fehler aufzeichnen

Ein sehr einfache Möglichkeit Web-Administratoren über Fehler der Webanwendung zu informieren, ist über die Fehler-Methode der *global.asax*. Diese Datei darf es nur einmal im Stammverzeichnis der Anwendung geben. Erzeugen können Sie sie, indem Sie *Neues Element hinzufügen* und die Vorlage *Globale Anwendungsklasse* auswählen. Falls diese nicht vorhanden ist, ist die *global.asax* bereits erzeugt worden.

Die Methode *Application\_Error* wird immer ausgeführt, wenn ein unbehandelter Fehler auftritt. Darin kann dann z.B. mit einem Zweizeiler eine E-Mail versandt werden, die den Fehler enthält. Dafür bietet das Server-Objekt die *GetLastError* Methode.

```
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    Dim smtpc As New Net.Mail.SmtpClient
    smtpc.Send("server@ppedv.de", "webmaster@ppedv.de", "Fehler aufgetreten", _
        Server.GetLastError.ToString())
End Sub
```

Listing 9.6 Fehlerüberwachung per E-Mail

Außerdem haben Sie die Möglichkeit in der *web.config* Fehlerseiten zu konfigurieren. Diese werden aufgerufen, sobald ein Fehler auftritt. Damit sieht der Benutzer eine sinnvolle und dem spezifischen Code entsprechende Seite. Außerdem kann diese ASPX-Seite wieder Code enthalten, der bestimmte Maßnahmen, wie das Versenden von E-Mails, ausführt.

Dies geschieht alles im Element *customErrors*. Mit dem Attribut *mode* lässt sich festlegen, dass ein lokaler Aufruf am Server den echten Fehler anzeigt. Je Fehlercode, z.B. 404 für eine nicht vorhandene Seite, lässt sich eine eigene Fehlerseite deklarieren.



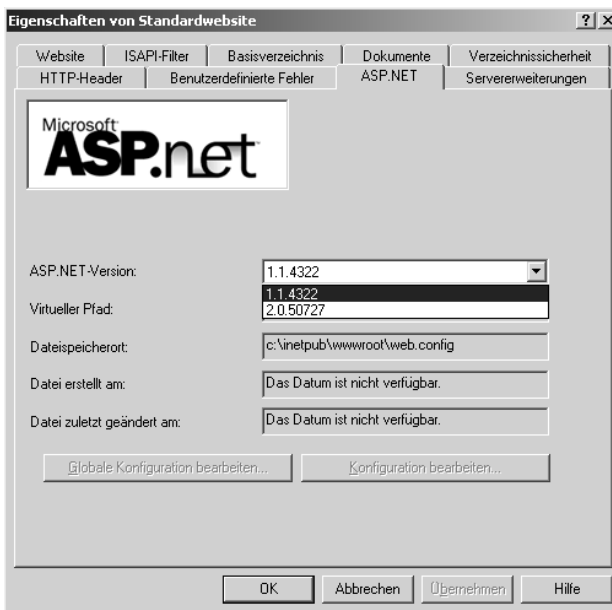
```
<customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.aspx">
  <error statusCode="403" redirect="NoAccess.aspx" />
  <error statusCode="404" redirect="FileNotFound.aspx" />
</customErrors>
```

**Listing 9.7** Fehlerseiten in *web.config* konfigurieren

Um den Benutzer nicht zu verschrecken, sollten Sie so z.B. auf einer 404er Fehlerseite eine Seitenübersicht und/oder einen Suchdialog anbieten.

## IIS-Konfiguration

Erste Anlaufstelle zur Konfiguration der Internet Information Services war schon immer die MMC (Management Console) mit dem entsprechenden IIS-Snap-In. Wenn Sie einen Webserver verwenden, befindet sich der Menüpunkt in *Administration Internet-Informationdienste*. Bei Windows XP befindet sich dieselbe Administrationsoberfläche in der Systemsteuerung unter *Verwaltung*.



**Abbildung 9.7** Umstellen der verwendeten ASP.NET-Version

Neu hinzugekommen ist dabei der Reiter *ASP.NET*, unter dem sich die Konfigurationsmöglichkeiten für das Web befinden.

Im ersten Dialog ist es möglich, für jede Website die verwendete ASP.NET-Version einzustellen.

Dies funktioniert zu jeder Zeit, also auch nachträglich. Den gleichen Effekt können Sie mit dem Kommandozeilentool *aspnet\_regiis* erzielen. Auf einem Webserver können so problemlos mehrere Versionen von ASP.NET parallel betrieben werden. IIS 6 bietet Application Pools an. Damit kann man Webanwendungen zusammenfassen und in einem gemeinsamen Speicherbereich laufen lassen. Achten Sie darauf, dass nie zwei Webanwendungen mit verschiedener .NET Framework Version im gleichen Pool arbeiten. Dies führt 100%ig, meist nicht sofort, zu einem Laufzeitfehler.

In den weiteren Dialogen, die mit der Schaltfläche *Konfiguration bearbeiten* aufrufbar sind, können dann weitere Details festgelegt werden. Diese werden ausnahmslos in der *web.config* der Anwendung gespeichert.

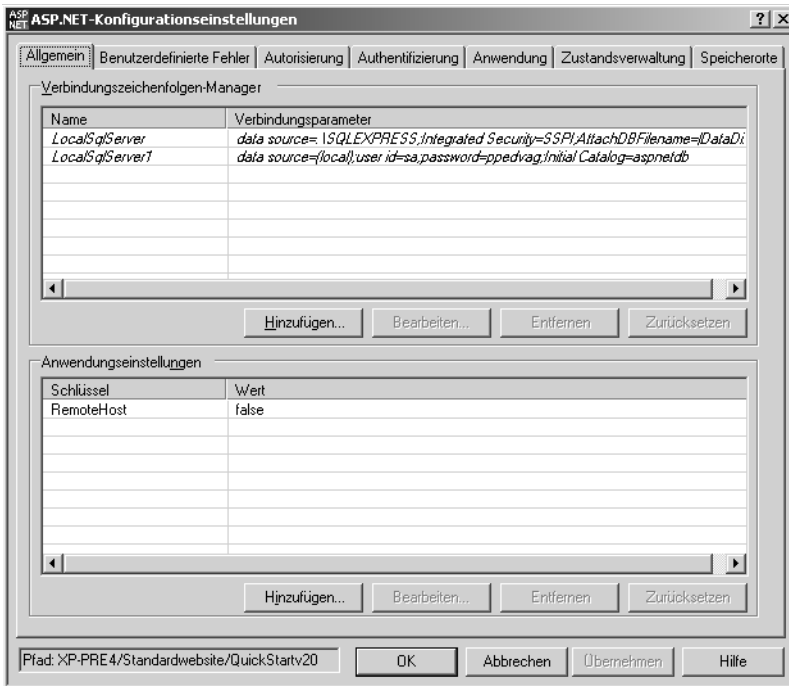


Abbildung 9.8 Authentication-Dialog in der MMC

Falls dieses ASP.NET-Snap-In nicht in den Reitern sichtbar sein sollte, haben Sie wahrscheinlich den IIS nach dem .NET Framework oder dem Visual Studio installiert. Starten Sie in dem Fall auf der Kommandozeile das Tool *aspnet\_regiis* mit dem Parameter *-i*.

## Websiteverwaltungs-Tool

Heutzutage lässt sich fast jeder Wireless Access Point per Web-Interface konfigurieren. Da wurde es höchste Zeit, dass auch Webseiten per Web-Administration konfigurierbar werden. Mit ASP.NET 2.0 ist das jetzt möglich.

Der Aufruf kann entweder per Menüpunkt in Visual Studio (ASP.NET-Konfiguration) oder direkt im Web-Browser erfolgen.

Aus Sicherheitsgründen wird dabei eine zweite Instanz des mitgelieferten Webdevelopment Servers auf einem zufälligen Port gestartet. Der Aufruf ist nur direkt am Server möglich.

Es gibt drei Bereiche, die konfiguriert werden können. Dabei wird direkt die *web.config* der Anwendung verändert. Wenn keine *web.config* vorhanden ist, wird sie beim ersten Aufruf erzeugt.

Leider wird dabei auch das *configuration* Element mit einem Namespace (*xmlns*) »verschönert«. Dann funktioniert in der Entwicklungsumgebung IntelliSense in der *web.config* nicht mehr. Entfernen Sie einfach das Attribut *xmlns* samt Wert.

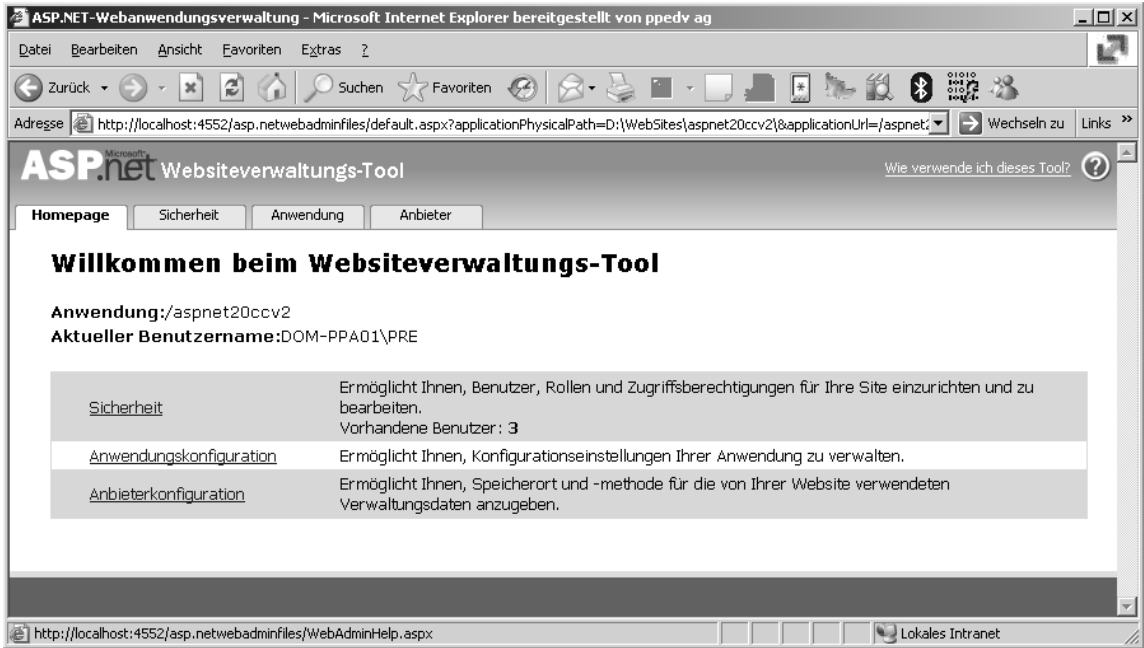


Abbildung 9.9 Startseite des Websiteverwaltungs-Tools

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
```

Das Administration Tool wurde selbst mit ASP.NET 2.0 programmiert und befindet sich samt Source-Code im folgenden Verzeichnis:

```
C:\WINDOWS\Microsoft.Net\Framework\v2.0.50727\ASP.NETWebAdminFiles
```

Microsoft hat nicht vorgesehen dieses Werkzeug auch in Produktivumgebungen einzusetzen. Vermutlich sprachens Sicherheitsbedenken dagegen. Mit ein wenig Geschick kann man aber das Verzeichnis manuell im IIS als virtuelles Verzeichnis freigeben. Wichtig ist, dass dieses mit integrierter Windows Authentifizierung arbeitet. Wenn man remote darauf zugreifen möchte, ist sogar noch eine Änderung im Code der Datei *WebAdminPage.cs* nötig. Dort muss die Abfrage, ob der Aufruf lokal erfolgt, ausdokumentiert werden.

```
application.Context.Request.IsLocal
```

Zu empfehlen ist diese Vorgehensweise allerdings nicht.

Auf der Registerkarte *Sicherheit* lassen sich Benutzer und Rollen anlegen und verwalten. Dazu wurde im Kapitel 6 bereits einiges ausgeführt. Bemerkenswert ist dabei noch die Funktion *Zugriffsregeln*. Damit lassen sich Zugriffsrechte auf Verzeichnisse definieren oder – anders ausgedrückt – Unterverzeichnisse schützen.

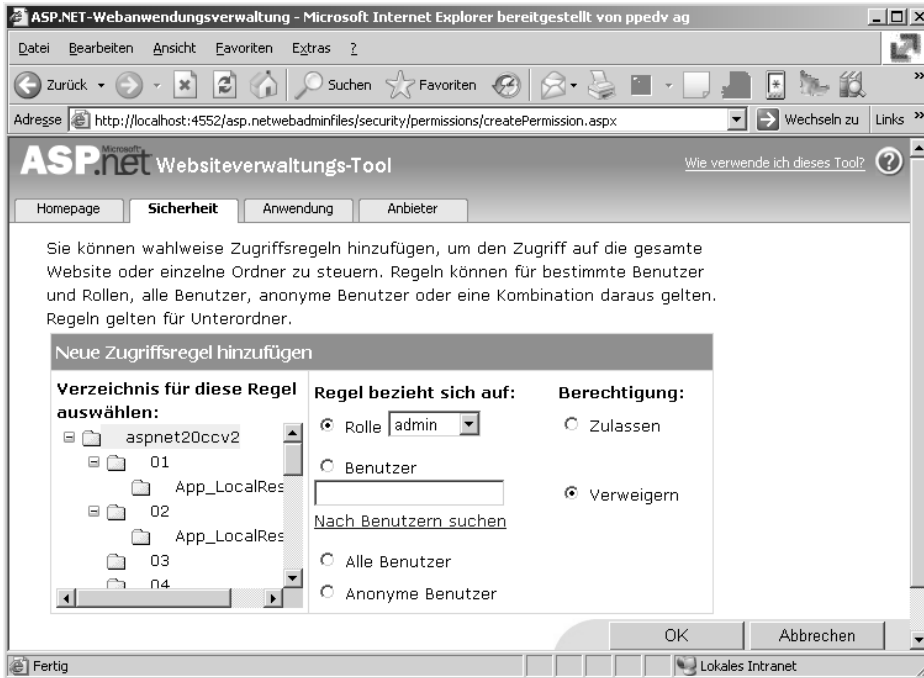


Abbildung 9.10 Die Sicherheitseinstellungen

Damit können für jeweilige Unterverzeichnisse Regeln pro Benutzer und Rollen angelegt werden, die dann eine *web.config* mit einem *authorization* Element speichern.

```
<system.web>
  <authorization>
    <allow roles="admin" />
    <deny users="Dog" />
  </authorization>
</system.web>
```

Listing 9.8 *web.config* aus Unterverzeichnis

Die meisten anwendungsspezifischen Einstellungen können im Reiter *Anwendung* vorgenommen werden. Die Einstellungen zum Versenden von E-Mails werden z.B. von den Login Controls verwendet.

```
<httpRuntime enable="false" />
```

Alternativ kann eine Anwendung durch Anlegen der Datei *app\_offline.htm* im Anwendungsverzeichnis kurzfristig »still« gelegt werden.

Wenn die Anwendung damit Offline genommen wird, wird in der *web.config* ein Eintrag erzeugt.

Darüber hinaus lässt sich mit der Konfigurations-API jede Einstellung auch programmatisch setzen.

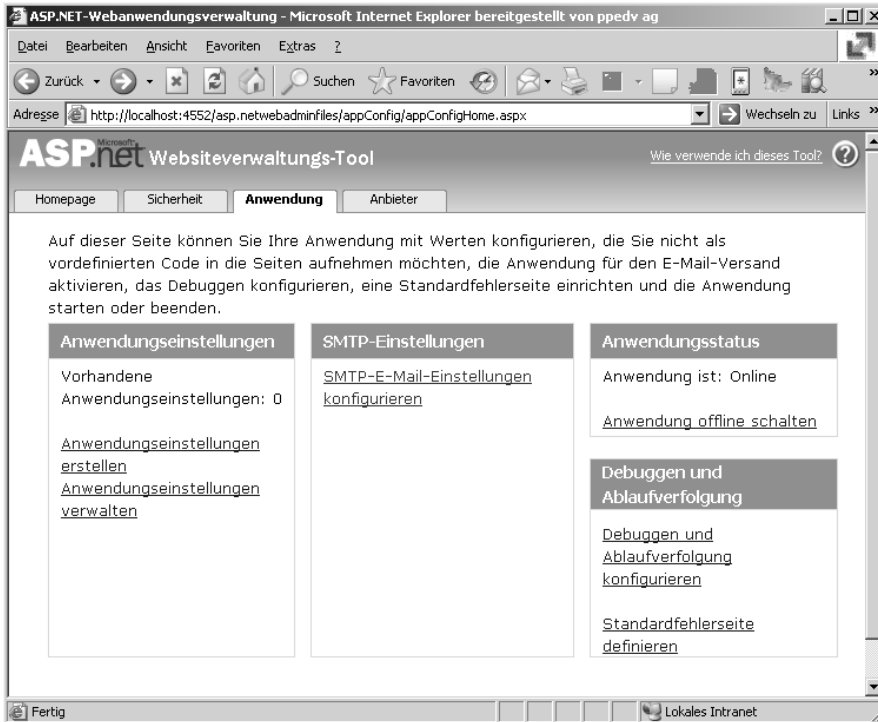


Abbildung 9.11 Allgemeine Application Settings

## Konfigurations-API

Mit der neuen API können Sie die Möglichkeiten zur Anwendungskonfiguration erweitern. Dies kann für die Installation oder Automatisierung sehr nützlich sein. Dabei muss man nicht direkt XML-Dateien lesen und schreiben. Außerdem muss man sich nicht um Sperren oder den Neustart der Webanwendung kümmern. Natürlich lassen sich damit auch ganze Gruppen von Servern remote-verwalten.

Die API für Webanwendungen befindet sich unter anderem im Namespace *System.Web.Configuration* in der Klasse *WebConfigurationManager*. Es gibt noch einen *ConfigurationManager*, der hier im Buch auch bereits verwendet worden ist. Letzter ist eigentlich für das Schreiben und Lesen von *app.config*-Dateien gedacht, funktioniert aber auch mit *web.config*. Der *WebConfigurationManager* bietet mehr webspezifische Methoden an. Dazu muss aber zuerst der Namensraum mit *Imports* referenziert werden.

```
<%@ Import Namespace="System.Web.Configuration" %>
```

Mit dem Befehl *OpenWebConfiguration* kann dann eine Konfiguration geöffnet werden. Dabei muss als Parameter der Name im IIS der Webanwendung eingegeben werden. Da es praktisch möglich ist, mehrere virtuelle Webserver zu betreiben, muss eventuell noch die laufende Nummer des Webserver vorangestellt werden.

```
WebConfigurationManager.OpenWebConfiguration("~/", "pfad", "Subpfad", "server")
```

Sogar die *machine.config*, auch von entfernten Servern, lässt sich auf ähnliche Weise öffnen.

```
WebConfigurationManager.OpenMachineConfiguration(null, "remoteserver", "user", "password")
```

## Bereich lesen

Im folgenden Beispiel sollen die Umleitungen aus dem Bereich *UrlMappings* aufgelistet und in der Folge auch ergänzt werden. Für die Darstellung wird dazu lediglich ein *GridView*-Steuerelement gewählt, dem die *UrlMappings*-Auflistung übergeben wird.

In einer *web.config* gibt es mehrere Hauptbereiche, die mit dem Befehl *GetSectionGroup* referenziert werden können. Die *UrlMappings* befinden sich in *System.Web*. Da die Rückgabe gleich in einem Objekt vom Typ *UrlMappingsSection* abgelegt wird, kann man die dort vorhandene Funktion *Mappings* verwenden, um die Liste für *GridView* zu erhalten.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myconf As Configuration = WebConfigurationManager.OpenWebConfiguration("~/")
    Dim myUrls As UrlMappingsSection
    Dim myGroup As SystemWebSectionGroup = myconf.GetSectionGroup("system.web")
    myUrls = myGroup.UrlMappings
    GridView1.DataSource = myUrls.UrlMappings
    GridView1.DataBind()
End Sub
```

**Listing 9.9** Anzeige im Browser der *UrlMappings* aus *web.config*

Die Anwendung erhält weiterhin zwei Textboxen, in die man neue URL-Umleitungen eintragen kann. ASP.NET erfordert immer eine Tilde »~« zur URL, die den Anwendungsstamm in der Eingabe darstellt.



**Abbildung 9.12** *UrlMappings* werden neu hinzugefügt

## Bereich speichern

Im nächsten Schritt wird ein zusätzlicher Eintrag in die URL-Umleitungsliste eingetragen. Auch dazu kommt wie vorher beschrieben der *WebConfigurationManager* zum Einsatz. Im vorherigen Bild kann man die Spalte *LockItem* erkennen, die Auskunft darüber gibt, ob jemand diese Daten zum Schreiben vorgesehen hat. Auch der Sperrmechanismus kommt aus der API.

Zunächst aber wird im folgenden Beispiel an die Auflistung der Mappings mithilfe des *Add*-Befehls eine neue Zuordnung erzeugt. Die Werte werden den Textboxen entnommen, die vom Benutzer per Eingabe gefüllt wurden.

Bevor der Schreibvorgang ausgeführt werden kann, wird geprüft, ob der Bereich sperrbar ist. Dann wird mit *Save* die Änderung zurückgeschrieben.

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myconf As Configuration = WebConfigurationManager.OpenWebConfiguration("~/")
    Dim myUrls As UrlMappingsSection
    Dim myGroup As SystemWebSectionGroup = myconf.GetSectionGroup("system.web")
    myUrls = myGroup.UrlMappings
    myUrls.UrlMappings.Add(New UrlMapping(txtUrl.Text, txtZiel.Text))
    If Not myUrls.LockItem Then
        myconf.Save()
    End If
End Sub
```

**Listing 9.10** Speichern der Änderungen in *web.config*

Die Änderung ist umgehend für die Anwendung ohne Neustart wirksam. Dies ist natürlich nur eine kurzer Einblick in die Möglichkeiten der Configuration-Schnittstelle.

## Kompilierung

Mit ASP.NET 1.x wurden Webanwendungen in eine DLL kompiliert. Diese liegt dann im *bin*-Verzeichnis der Webanwendung. Beim ersten Aufruf einer ASPX-Seite springt dann der JIT-Compiler an und erstellt den eigentlichen Binärcode. Das Verfahren hatte seine Schwächen, speziell bei umfangreicheren Webseiten.

Da jetzt die eigentlichen Web-Projektdateien wegfallen, muss auch keine große DLL erzeugt werden. Es gibt für jeden Bedarf die richtige Art, seine Webpräsenz zu kompilieren.

### Automatische Kompilierung

Am einfachsten ist es, gar keine Vorgaben zu machen. Beim ersten Aufruf einer ASPX-Seite wird diese dann automatisch kompiliert. Beim nächsten Aufruf wird sie aus dem Cache geladen.

### Vorkompilierung

Aus verschiedenen Gründen kann es Sinn machen, Webseiten vorzukompilieren. Dabei können Fehler im Code schneller, eben zur Kompilierzeit, erkannt werden.

Ein gewichtigeres Argument ist aber sicher die Performance, da die Wartezeiten beim ersten Aufruf entfallen.

Am besten ist aber die neue Funktion, mit deren Hilfe Sie sogar ASPX-Seiten komplett in eine DLL kompilieren können. Dadurch muss man den HTML-Ausgangscode nicht mehr weitergeben.

Wenn Sie Visual Web Developer Express verwenden, steht dieser Menüpunkt nicht zur Verfügung.

Mit dem Menüpunkt *Erstellen/Website erstellen* wird eine normale Kompilierung durchgeführt. Um eine Webseite für die Weitergabe zum Produktionsserver vorzubereiten, gibt es ebenfalls in dem Menü *Erstellen* den Unterpunkt *Website veröffentlichen*. Damit wird das Projekt neu erstellt und gleich zum Produktionsserver kopiert.

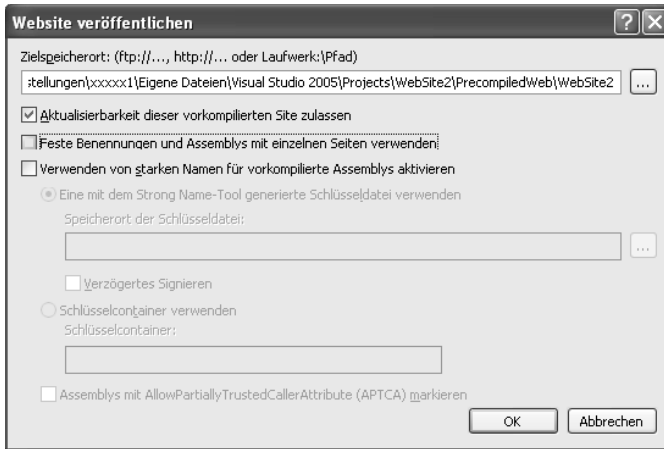


Abbildung 9.13 Website kompilieren und weitergeben

Wenn die Option *Aktualisierbarkeit dieser vorkompilierten Site zulassen* deaktiviert ist, wird der HTML-Code aus den ASPX-Seiten entfernt und in Assemblies (DLL's) einkompiliert. Alle ASPX-Dateien sind dann leer und können nicht mehr vom Kunden oder Website-Betreiber verändert werden. Die DLL-Dateien liegen dann wie bisher im *bin*-Verzeichnis, pro Webseite und Klassendatei gibt es jeweils eine eigene Datei.

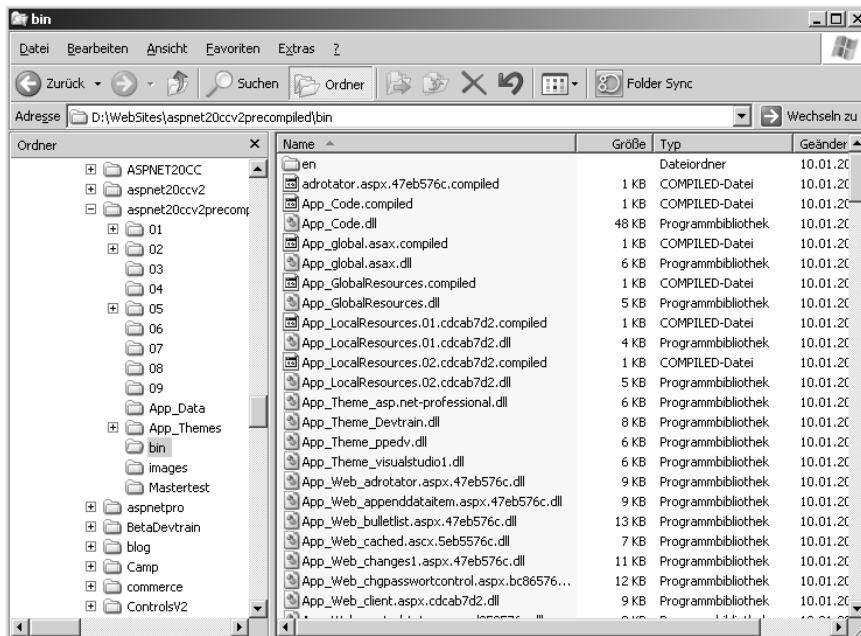


Abbildung 9.14 Das fertig erstellte Web-Projekt

## Visual Studio 2005 Web Deployment Projects

Microsoft hat offensichtlich erkannt, dass die Möglichkeiten zum Kompilieren der Web Site nicht ausreichend sind und liefert als kostenlosen Download einen Zusatz aus, mit dem sich die MSBuild Jobs genauer konfigurieren lassen. Dies funktioniert nicht mit der Express Edition.



Im Projektmappen Explorer kann ein neuer Projekttyp *Web Deployment Project* (Web Verteilungs Projekt) hinzugefügt werden. Darin werden die Verteilungsoptionen gesetzt, die dann den Menüpunkt *Website veröffentlichen* ersetzen.

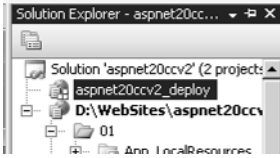


Abbildung 9.15 Web Deployment Projekt

Damit lassen sich auch erweiterte Aufgaben wie das Ersetzen von ConnectionStrings in der web.config im Build Prozess automatisieren.

### Kommandozeilenkompilierung

Wer lieber manuell vorgeht oder die Visual Web Developer Express Edition verwendet, kann per Kommandozeilen-Operationen den ASP.NET-Compilervorgang auch per Hand anstoßen. Das Kommando dafür lautet *aspnet\_compiler*. Mit dem Parameter *-v* gefolgt vom Namen des virtuellen Verzeichnisses wird dieses kompiliert.

```
aspnet_compiler -v virtualPath
```

Weitere zusätzliche Parameter erlauben erweiterte Möglichkeiten.

Brandneu hinzugekommen ist das Tool *aspnet\_merge* mit dem sich mehrere Assemblies zu einem Assembly zusammenfügen lassen, das dann wie gehabt ins BIN Verzeichnis gelegt wird.

### Visual Studio 2005 Web Application Projects

Ein weiteres Add On zu Visual Studio 2005 erlaubt es wieder mit Web Projekten zu arbeiten. Zum Zeitpunkt der Drucklegung dieses Buches ist dieses nur als Preview verfügbar.

## Health Monitoring

Mit Health Monitoring wurde in ASP.NET 2.0 ein völlig neuer Mechanismus geschaffen, um Webanwendungen zu überwachen. Dank eines Provider-Konzeptes können die Nachrichten an verschiedene Systeme geschickt werden.

- Windows Management Instrumentation (WMI)
- Windows Ereignisanzeige
- Windows Event Tracing
- Performance Counter
- SQL Datenbank logging
- E-Mail
- Log files

Zusätzlich lassen sich eigene Provider entwickeln.

Das Verhalten der Provider lässt sich dabei per *web.config* konfigurieren. Es sind in der globalen *web.config* bereits die drei Provider *EventLogProvider*, *WmiWebEventProvider* und *SQLWebEventProvider* vorkonfiguriert. Dabei verwendet der letztere in der Default Einstellung erwartungsgemäß *SQLExpress* und darin die Datenbank *aspnetdb*. Um diese Funktion grundsätzlich zu aktivieren, muss im Element *healthmonitoring* erst das Attribut *enabled* auf *true* gesetzt werden. Damit arbeitet das Monitoring auch umgehend entsprechend den Einstellungen in der globalen *web.config* und schreibt die Fehler ins Eventlog. Daneben gibt es weitere Unterelemente.

```
<healthMonitoring
  Enabled="true"
  heartbeatInterval="Zeit Interval">
  <bufferModes>...</bufferModes>
  <providers>...</providers>
  <eventMappings>...</eventMappings>
  <profiles>...</profiles>
  <rules>...</rules>
</healthMonitoring>
```

**Listing 9.11** Health Monitoring Bereich aus *web.config*

## Rules

Um nun Fehlermeldungen in die SQL-Datenbank umzuleiten, müssen entsprechende Regeln im Rules-Element angelegt werden. Eine neue Regel wird mit dem *Add* Element erzeugt und mit dem Attribute *name* benannt. Entscheidend ist, dass das Attribut *provider* auf einen konfigurierten Health Monitor Provider verweist.

```
<rules>
  <add name="Alle Fehler" eventName="All Errors" provider="SqlWebEventProvider"
    profile="Default" minInstances="1" maxLimit="Infinite" minInterval="00:01:00"
    custom="" />
</rules>
```

**Listing 9.12** Alle Fehler werden zusätzlich in die Datenbank geschrieben

Die Anzeige der Fehler kann dann in einer Webseite ganz einfach mit dem GridView-Steuerelement erfolgen. Die möglichen Parameter im Attribut *eventName* sind per *EventMapping* vordefiniert.

## EventMappings

Die Ereignisse, die auftreten können sind in dem *EventMapping* Element in der globalen *web.config* vordefiniert. Es können auch eigene EventMappings erzeugt werden.

- All Events
- Heartbeats
- Application Lifetime Events
- Request Processing Events
- All ErrorsInfrastructure Errors
- Request Processing Errors
- All Audits
- Failure Audits
- Success Audits

Die ist nur ein Auszug aus den unglaublich reichhaltigen Möglichkeiten des Health Monitorings.

## Kapitel 10

# Die Client-Seite

### **In diesem Kapitel:**

Script Jobs	268
Registrieren von Client Scripts	272
Client Callbacks	276

Eigentlich endet der Job des ASP.NET Entwicklers an den Grenzen des Webservers. Der erzeugte HTML Code, der dann an den Client gesendet wird und schlussendlich eine Darstellung im Browser bewirkt, ist nicht mehr im Job Profil enthalten. Aber die Praxis zeigt natürlich, dass die Verantwortung nicht abgescho-ben werden kann. Also ist der ASP.NET-Programmierer für alles zuständig, letztendlich auch für das Gesche-hen im Browser. Neben HTML und CSS ist es hauptsächlich JScript-Code der geschrieben werden muss und sich gänzlich der OO-Welt entzieht. All das musste bisher weitestgehend als einfacher Text behandelt wer-den.

Erst mit ASP.NET 2.0 gibt es im *Page*-Objekt eine Zusammenfassung der wichtigsten Funktionen in der *ClientScript*-Klasse.

Will der Entwickler in die Königsklasse der Webentwicklung einsteigen, bleibt es ihm trotzdem nicht erspart, sich intensiv mit HTML und JScript auseinanderzusetzen.

JScript ist eine interpretierte Sprache, die optisch Java ähnelt. Richtig heißt JScript eigentlich *ECMA Script*, weil sich ECMA um die Normierung dieser Sprache kümmert.

ASP.NET baut einfach JScript in den Response Stream, der zum Browser gesendet wird, als Text ein. Wenn der Browser die Seite vollständig erhalten hat, werden von diesem eventuell enthaltene Scripts ausgeführt. Eines der Probleme dabei ist, wie solche Code-Fragmente am Server und Client miteinander kommunizie-ren. Es müssen Daten zwischen Server und Client ausgetauscht werden.

Um diese Daten vom Server in ein Client-Script zu bringen, kann ein Cookie verwendet werden, oder das Script wird gleich so am Server aus ASP.NET zusammengesetzt, dass die Variablen richtig gefüllt sind.

```
<% Response.Write("<script language=jscript>alert('am Server ist es: " + Date.Now + "');</script>")%>
```

Auf diese Weise kann man die benötigten Scripts eigentlich auch direkt in die ASPX-Seite schreiben. Daraus können sich aber zur Laufzeit Konflikte ergeben. Ein Script, das z.B. in einem User-Steuerelement enthalten ist, würde mehrfach in die Seite geladen werden, wenn das User-Steuerelement mehrfach vorhanden ist. Ganz nebenbei ist solcher Code schwer zu lesen und zu pflegen.

Um mit solchen Situationen umgehen zu können, gibt es in ASP.NET 2.0 einige Hilfsfunktionen, die hier erläutert werden.

Darüber hinaus gibt es einige spezielle Tricks, die bisher manuell mit JScript erledigt werden mussten und jetzt in ASP.NET 2.0 bereits eingebaut sind.

## Script Jobs

Mit Client Scripts lässt sich aus einer Browseranwendung eine Rich-Client-Anwendung machen. Beinahe alles, was eine Windows-Anwendung so benutzerfreundlich macht, ist auch mit JScript zu verwirklichen.

### Fokus setzen

Einer der häufigen Wünsche des Entwicklers ist es, den Fokus auf ein bestimmtes Eingabefeld zu setzen. Der Eingabecursor soll also in einer Textbox blinken, wenn der Benutzer die Seite aufruft.

Die einfachste Methode ist es, etwas über das Attribut *defaultfocus* des Form-Elements zu realisieren. Als Parameter wird einfach der Name des Steuerelements übergeben. Dies wird in Kapitel 5 beschrieben.

```
<form id="form1" runat="server" defaultfocus="TextBox2" >
```

Die Textbox besitzt auch direkt eine Eigenschaft *Focus*, die das gleiche Ziel erreicht. Im Browser wird ein JScript erzeugt, das den Focus setzt:

```
<script type="text/javascript">
<!--
WebForm.AutoFocus('TextBox2');// -->
</script>
```

**Listing 10.1** Erzeugtes JScript im Browser

Dabei wird eine Script-Bibliothek verwendet, die ASP.NET unter Verwendung eines http-Handlers zur Verfügung stellt. Mit dem Aufruf von *WebResource.axd* unter Angabe von Parametern wird so das Script geladen.

```
<script src="/aspnet20ccv2/WebResource.axd?d=hgKcZ7Seun2_Y2qx1SAh0w2&amp;t=632662676484602144" _
type="text/javascript"></script>
<script src="/aspnet20ccv2/WebResource.axd?d=v9IeoZ7x2EQPzp_suIkUiw2&amp;t=632662676484602144" _
type="text/javascript"></script>
<script src="/aspnet20ccv2/WebResource.axd?d=0F9j0Qjcv1NoH3JiL6pJhw2&amp;t=632662676484602144" _
type="text/javascript"></script>
```

**Listing 10.2** Auszug aus HTML Source: Script-Referenz im Browser

Der Vorteil dieser Methode ist, dass die Script-Bibliotheken nicht im Verzeichnis der Anwendung vorhanden sein müssen.

Leider funktioniert das im Zusammenhang mit Masterseiten nicht so einfach, da in der Inhalts-ASPX-Seite kein Form-Element mehr vorhanden ist. Auch das Steuerelement hat eine Eigenschaft *DefaultFocus*, dazu muss per *FindControl* erst das Steuerelement gefunden werden, dem man den Fokus geben will. Einfacher geht es aber mithilfe der Eigenschaft *UniqueID*, wie dies im nächsten Abschnitt gleich beschrieben wird.

```
Page.Form.DefaultFocus = TextBox2.UniqueID
```

## Default Button (Standardschaltfläche)

Wenn auf einem Formular mehrere Buttons vorhanden sind, möchte man steuern, welche Button-Ereignismethode ausgeführt wird, wenn der Benutzer Eingabe drückt. Dazu deklariert man den Default-Button (die Standardschaltfläche).

Genau wie beim Setzen des Fokus kann man dies mit dem zusätzlichen Attribut *defaultbutton* direkt im Form-Element realisieren.

```
<form id="form1" runat="server" defaultbutton="Button2">
```

Etwas schwieriger wird es auch hier im Zusammenspiel mit Masterseiten. Zum einen gibt es natürlich kein Form-Element in der Content-Seite, zum anderen sind die IDs, die erzeugt werden, zu diesem Zeitpunkt unbekannt.

Ein angenommener *Button1* wird so in der ASP.NET Steuerelement-Hierarchie vielleicht zu einem `Button` `ctl00$ContentPlaceHolder1$TextBox2`.

Für das erste Problem kann man über das *Page*-Objekt auf das Formular zugreifen und so den Default-Button setzen. Das zweite Problem löst sich, wenn man der Eigenschaft *UniqueID* ein wenig Aufmerksamkeit schenkt. *UniqueID* entspricht der erzeugten ID genau, und sie ist es, die wir benötigen, wenn wir im *Page\_Load*-Ereignis den Default-Button setzen wollen.

```
Page.Form.DefaultButton = Button2.UniqueID.ToString
```

**ACHTUNG**

Auch ein Panel-Steuerelement besitzt die Eigenschaft *DefaultButton*.

## Position wieder finden

Wenn die Webseite mehr Inhalt hat und damit ein Scrollen erfordert, ist es sehr ärgerlich, wenn sich der Benutzer nach einem Postback wieder am Anfang der Seite findet. ASP.NET 2.0 unterstützt das Zurückkehren zur alten Seitenposition automatisch. Das ist z.B. bei der Darstellung einer langen Liste mit einem *GridView*-Steuerelement sehr nützlich. Dabei hilft das Attribut in der Page Direktive *MaintainScrollPositionOnPostBack="true"*.

## Client Click

Webserver-Steuerelemente verfügen nur über Server-Ereignismethoden. Was auf den ersten Blick absolut sinnvoll erscheint, reicht aber manchmal doch nicht. Wenn ein Button beispielsweise eine Löschaktion ausführen soll, wäre eine sofortige Rückfrage an den Benutzer durchaus wünschenswert. Eine Rückfrage kann im Browser per JScript Funktion *Confirm* durchgeführt werden.

Man braucht dazu also auch ein *Client*-Ereignis. Einige der Steuerelemente besitzen jetzt ein *OnClientClick*-Attribut. Diesem wird entweder der Funktionsname der Client-Funktion oder direkt ein JScript übergeben.

Wenn der Rückgabewert *false* ist, werden der Postback und damit z.B. eine anstehende Löschoperation abgebrochen.

Im Browser wird ein PopUp-Fenster angezeigt, das mit *OK* oder *Abbrechen* bestätigt werden muss.



**Abbildung 10.1** Löschvorgang wird mit PopUp bestätigt

Das JScript wird im nächsten Beispiel direkt in die ASPX-Seite geschrieben. Je nach Eingabe wird dann *true* oder *false* zurückgeliefert. Das Attribut *OnClick* des Buttons wurde mit *return pruefe();* belegt. Auf diese Weise wird der Rückgabewert auch in der Seite ausgewertet und dementsprechend der Postback ausgeführt oder abgebrochen.

```
<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "gelöscht"
    End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <script language="ecmascript">
        function pruefe()
        {
            if(confirm('wirklich löschen'))
            {
                return true;
            }
            return false;
        }
    </script>
    <asp:Label ID="Label1" Runat="server" Text="" EnableViewState="false"></asp:Label>
    <asp:Button ID="Button1" Runat="server" Text="delete" OnClientClick="return pruefe();"
        OnClick="Button1_Click" />
</asp:Content>
```

**Listing 10.3** Delete-Aktion mit Client Script prüfen

Es soll hier nicht verheimlicht werden, dass es auch kürzer geht.

```
<asp:Button ID="Button2" runat="server" OnClientClick="return confirm('wirklich löschen?');"
    Text="löschen" />
```

So ganz reizt dieses Beispiel die Möglichkeiten von ASP.NET 2.0 noch nicht aus. Deshalb betrachten wir als Nächstes die Klasse *Page.ClientScript*, die noch mehr Funktionalität besitzt.

## Registrieren von Client Scripts

Aus Webseiten sind clientseitige Scripts nicht mehr wegzudenken. Grundsätzlich ist das eigentlich ein Thema außerhalb des Themas Objektorientierung und auch außerhalb von ASP.NET. Da Entwickler dies aber häufig benötigen, bietet ASP.NET einige Hilfsfunktionen an, die nun in *Page.ClientScript* zusammengefasst sind.

Unter ASP.NET 1.x hat man diese Funktionen, wie z.B. *RegisterClientScriptBlock*, direkt aufgerufen. Aus Kompatibilitätsgründen geht das auch weiterhin. Allerdings löst jetzt eine komplett neue Klasse namens *ClientScriptManager* diese Funktionen ab. Die Entwicklungsumgebung von Visual Studio 2005 unterschlängelt aus diesem Grund die alte Form dieser Verwendung grün.

Das Objekt *ClientScript* ist eine Instanz in der Page des *ClientScriptManagers* und wird für den Zugriff darauf verwendet.

### RegisterClientScriptBlock

Um ein Script in die Webseite zu schreiben, kommt *RegisterClientScriptBlock* zum Einsatz. Damit wird ein Script, versehen mit einem eindeutigen Schlüssel, in den HTML-Bereich geschrieben. Da sich die Position des Blocks in der erzeugten HTML-Seite nach dem Form-Element und vor den anderen Seitenelementen befindet, können auch HTML-Code, Java Applets und was sonst noch notwendig ist, auf diese Weise in die Seite geschrieben werden.

Die Methode steht zweifach überladen zur Verfügung. Die erste Variante erwartet drei Parameter: Den Typ (*me.GetType* für Vb und *this* für C#), den Schlüssel und das Script als String.

Die zweite Zeile aus folgendem Beispiel-Code setzt den vierten Parameter auf *true*. Somit werden automatisch die umschließenden Script-Tags erzeugt.

```
ClientScript.RegisterClientScriptBlock(Me.GetType, "neueMethode", "alert('test');", True)
ClientScript.RegisterClientScriptBlock(Me.GetType, "neueMethode1", "<b>test</b>")
```

**Listing 10.4** Die beiden Möglichkeiten *RegisterClientScriptBlock* zu verwenden

Durch den Schlüssel wird verhindert, dass ein Script mit dem gleichen Schlüsselnamen ein zweites Mal in die Seite geladen wird. Allerdings bezieht sich die Eindeutigkeit nur auf den gleichen Script-Typen. Wenn das Script mit einer anderen Funktion als *RegisterStartupScript* registriert wird, kann der gleiche Schlüssel ein zweites Mal vergeben werden.

#### HINWEIS

Bei den Schlüsselnamen wird zwischen Klein- und Großschreibung unterschieden.

### RegisterStartupScript

Ein Script, das nach dem Laden der Seite automatisch ausgeführt werden soll, nennt man ein *StartupScript*. Dies wird in der Browserausgabe immer an das Ende der Seite vor dem schließenden Form-Element platziert. Dies geschieht damit sichergestellt ist, dass alle Elemente auf der Seite auch vorhanden sind, auf die sich das Script per DOM eventuell bezieht.

Wenn schon andere Scripts, die von ASP.NET automatisch generiert worden sind, in der Seite stehen, wird der automatisch erzeugte, umschließende Script-Block gleich weiterverwendet.



Auch *RegisterStartupScript* existiert mit zwei Überladungen, wobei der vierte Parameter, falls *true*, ebenfalls die Script-Elemente erzeugt.

```
ClientScript.RegisterStartupScript(Me.GetType, "neueMethode", "alert('start');", True)
ClientScript.RegisterStartupScript(Me.GetType, "neueMethode1", "<b>test</b>")
```

**Listing 10.5** Die beiden Verwendungsmöglichkeiten von *RegisterStartupScript*

Wie im Beispiel zu sehen ist, kann man damit auch beliebige Textblöcke ausgeben.

## RegisterClientScriptInclude

Wenn das Script etwas größer ist oder es öfter verwendet werden soll, legt man es am besten in eine eigene Datei. Dies ist nicht nur wegen der Wiederverwendbarkeit von Vorteil, sondern wird auch besser vom Browser gecacht.

Um das Script zu laden, muss ein Script-Block mit einem Verweis auf die Script-Datei in die Seite eingebaut werden. Dies geschieht mit *RegisterClientScriptInclude*.

```
ClientScript.RegisterClientScriptInclude(Me.GetType, "include2", "meinfile.js")
```

Im Browser wird dann ein *script*-Element mit einem *src*-Verweis erzeugt.

```
<script src="meinfile.js" type="text/javascript"></script>
```

## RegisterClientScriptResource

Die Script-Bibliotheken, die ASP.NET für seine Standardfunktionen im Browser benötigt, werden mit der Datei *WebResource.axd* geladen. Dabei wird das gewünschte Script per QueryString angegeben. Um selbst einen Ladevorgang einzuleiten, wird die Funktion *RegisterClientScriptResource* verwendet. Der zweite Parameter definiert den Namen des Scripts, das in den Browser geladen werden soll.

```
ClientScript.RegisterClientScriptResource(Me.GetType, "Ressource.js")
```

Der vorherige Befehl erzeugt dann im Browser den folgenden Code.

```
<script src="WebResource.axd?a=paawbbr_i&r=Ressource.js&t=632299799954973904" type="text/javascript"></script>
```

## RegisterHiddenField

HiddenFields sind *HTML-INPUT*-Elemente mit dem Zusatz *Hidden*. So sind sie zwar in der Seite vorhanden, aber für den Benutzer unsichtbar. Verwendung finden diese meist, um Daten weiter zu transportieren. Der ViewState wird z.B. auf diese Weise gespeichert. So kann auch ein eigenes HiddenField per Code erzeugt werden.

```
ClientScript.RegisterHiddenField("feld", "wert")
```

Das so erzeugte HTML-Element wird vor dem ViewState in die Seite eingefügt.

```
<input type="hidden" name="feld" value="wert" />
```

### HINWEIS

HiddenFields sind nicht geeignet, um sensitive Informationen vor dem Benutzer zu verbergen.

## RegisterOnSubmitStatement

Manchmal soll bei einem Form Submit ein JScript ausgeführt werden. Das ist speziell bei der Verwendung von Masterseiten nicht so einfach, weil das Form-Element in der ASPX-Seite nicht vorhanden ist. Mit *RegisterOnSubmitStatement* wird das Problem umgangen. Als Parameter benötigt die Methode den Schlüssel und das Script. Der umschließende Script-Block wird automatisch erzeugt.

```
ClientScript.RegisterOnSubmitStatement(Me.GetType, "Key1", "alert('submit');")
```

Damit man auch mehrere Scripts beim Submit der Seite ausführen kann, ist der erzeugte Code im Browser etwas aufwändiger. Es wird im Form-Element auf eine JScript-Funktion *WebForm\_OnSubmit* verwiesen, in der dann der eigentliche Code steht.

```
<form method="post" action="/ASPNET20CC/10/clientScript.aspx"
  onsubmit="javascript:return WebForm_OnSubmit();" id="__aspnetForm" autocomplete="on">
...
<script type="text/javascript">
<!--
function WebForm_OnSubmit() {
  alert('submit');
  return true;
}
// -->
</script>
```

**Listing 10.6** Ein JScript wird bei Submit ausgeführt

## RegisterArrayDeclaration

Wenn Sie ein Array an den Client übergeben müssen, kommt die Funktion *RegisterArrayDeclaration* zum Einsatz. Pro Element im Array wird die Funktion einmal aufgerufen. Je nach JScript-Datentyp wird der Wert in Hochkommas gestellt.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
  ClientScript.RegisterArrayDeclaration("sex", "'frau'")
  ClientScript.RegisterArrayDeclaration("sex", "'mann'")
  ClientScript.RegisterArrayDeclaration("sex", "'ding'")
End Sub
```

**Listing 10.7** Ein Array wird am Client erzeugt.

Im Browser sieht der erzeugte JScript-Code dann wie folgt aus:

```
<script type="text/javascript">
<!--
var sex = new Array('frau', 'mann', 'ding');
// -->
</script>
```

**Listing 10.8** Ein Array wird an den Client gesendet

## RegisterExpandoAttribute

Ein echter Exot ist das *Expando*-Attribut. Damit kann man zur Laufzeit HTML-Objekten Methoden hinzufügen. Es lässt sich so ein unselektierbarer Text auf der Webseite darstellen.

```
ClientScript.RegisterExpandoAttribute("SPAN1", "unselectable", "on", True)
```

Das setzte allerdings den Einsatz von JScript voraus, VBScript unterstützt dies nicht.

## Prüffunktionen

Die weiteren Funktionen prüfen, ob ein Script bereits mit einer der bisher beschriebenen Methoden registriert wurde. Da diese sehr einfach und in der Anwendung ähnlich sind, erfolgt die Beschreibung in kurzer Form.

### IsClientScriptBlockRegistered

Mit der Funktion *IsClientScriptBlockRegistered* prüft man, ob das Script mit diesem Schlüssel schon registriert ist. Der Schlüssel ist case-sensitiv (Groß-/Kleinschreibung wird unterschieden). Der erste Parameter *Typ* muss identisch mit dem bei der Registrierung verwendeten sein. Die Rückgabe ist vom Datentyp *Boolean*.

```
If ClientScript.IsClientScriptBlockRegistered(Me.GetType, "neueMethode1")) then ..
```

### IsClientScriptIncludeRegistered

Mit der Funktion *IsClientScriptIncludeRegistered* wird geprüft, ob das Script mit diesem Schlüssel schon registriert ist. Die Rückgabe ist vom Datentyp *Boolean*.

```
ClientScript.IsClientScriptIncludeRegistered(Me.GetType, "meinfile")
```

### IsOnSubmitStatementRegistered

Die Funktion *IsOnSubmitStatementRegistered* prüft, ob das Script mit diesem Schlüssel schon registriert ist. Die Rückgabe ist vom Datentyp *Boolean*.

```
ClientScript.IsOnSubmitStatementRegistered(Me.GetType, "key1")
```

## IsStartupScriptRegistered

Mit der Funktion *IsStartupScriptRegistered* prüft man, ob das Script mit diesem Schlüssel schon registriert ist. Die Rückgabe ist vom Datentyp *Boolean*.

```
ClientScript.IsStartupScriptRegistered(Me.GetType, "key1")
```

## Rückgaben mit Get

Einige der Funktionsnamen im ClientScriptManager beginnen mit Get und liefern Daten zurück. Auszugsweise werden folgende kurz beschrieben:

### GetPostBackEventReference

Wenn aus einer Client-Aktion eine Server-Methode ausgeführt werden soll, ohne dass es sich um die üblichen Webserver-Steuerelemente handelt, kann die Funktion *GetPostBackEventReference* helfen. Damit bekommt man die ID eines Server-Steuerelements und kann diese in ein JScript einbauen. So kann dieses Script am Client eine Postback Funktion eines Server-Steuerelements auslösen. Im folgenden Beispiel wird nach fünf Sekunden eine Button-Methode aufgerufen und damit der Button faktisch gedrückt.

```
<script language="javascript">
window.setTimeout(myTimer,5000);
function myTimer()
{
    <%= ClientScript.GetPostBackEventReference(Button1, nothing) %>
}
</script>
```

**Listing 10.9** Button Klick wird simuliert

Unter ASP.NET 1.x wurde dies oft manuell durchgeführt, indem man die HiddenFields (EventTarget und EventArguments) manipuliert hat. Dies führt jetzt aus Sicherheitsgründen zu einem Laufzeitfehler. Der Hinweis ist dann, dass man *EnableEventValidation* in der Page Direktive deaktivieren muss.

### GetPostBackClientHyperlink

Funktioniert wie *GetPostBackEventReference*, fügt allerdings an den Beginn der erzeugten Jscript-Funktion ein *javascript:* ein.

### GetCallbackEventReference

Damit erhält man eine Referenz auf eine clientseitige Funktion. Diese Funktion wird in Zusammenarbeit mit der Funktionalität Client Callbacks verwendet.

## Client Callbacks

Den alten Hasen unter Ihnen mag Remote Scripting noch etwas sagen – bei ASP.NET suchte man das bisher vergeblich. Mit der Version 2 wird dieses Feature nun unter dem Namen *Script Callbacks* eingeführt. In den ersten Versionen von ASP.NET 2.0 wurde die Funktion als *ClientCallback* bezeichnet. Kurz gesagt lässt sich

damit eine Funktion am Server ausführen und deren Rückgabe verarbeiten, ohne dass für den Benutzer sichtbar die Seite verlassen wird. Also Callback statt Postback.

Das Flackern beim Seitenwechsel fällt damit weg, und die übertragene Datenmenge sinkt. Voraussetzung ist allerdings ein Browser wie IE 6, der XMLHTTP unterstützt.

Einige der Webserver-Steuerelemente haben die Fähigkeit, dieses Feature für manche Ihrer Funktionalitäten zu nutzen, beispielsweise das GridView und das FormView Control.

Im hier beschriebenen Beispiel wird damit eine Chat-Anwendung realisiert.

## Chat

Ein Chat erlaubt es mehreren Benutzern, miteinander zu kommunizieren. Alle teilen sich einen Chat. Deshalb könnte die Chat-Historie in einer *Application*-Variablen oder in einem Cache-Objekt gespeichert werden. Die letztere Methode erlaubt es, bei längerer Inaktivität den Chatverlauf einfach zu löschen. Jeder Benutzer wählt einen Alias, der über eine Textbox erfasst wird. Der eigene Kommentar zum Chat wird ebenfalls in einer Textbox erfasst. Der Chatverlauf wird in einer mehrzeiligen Textbox angezeigt. Die Chateinträge kann man in einem Array speichern.

Das Absenden eines Textes und das Erneuern der Anzeige des Chatverlaufs würden normalerweise einen Postback erfordern.

Da nicht alle Benutzer dauernd und kontinuierlich schreiben, ist das weniger das Problem. Allerdings möchten alle Benutzer sofort oder wenigstens ohne große Zeitverzögerung neue Eingaben der anderen Benutzer sehen. Dazu ist ein Refresh-Intervall von 10 Sekunden durchaus angebracht. Genau diese Refresh-Funktion implementiert der folgende Chat als Callback-Funktion. Damit wird der erzeugte Datenverkehr geringer, und die Webseite gestaltet sich benutzerfreundlicher. Ganz nebenbei wird das Ganze ohne den Einsatz von Frames durchgeführt.

Weil der Code einfach gehalten werden soll, wird jedes Mal der komplette Chat-Text übertragen. Für den produktiven Einsatz sollte nur der neu hinzugekommene Text geladen werden. Dies setzt eine Timestamp-Verwaltung und -Prüfung voraus.

---

**ACHTUNG** Bevor Sie loslegen, überlegen Sie sich gut die Namen der Funktionen und Variablen. Da auch JScript im Spiel ist, muss auch die Groß-/Kleinschreibung berücksichtigt werden. Die Fehlersuche ist sehr aufwändig.

---

## Server-Code

Um eine Webseite grundsätzlich mit einer Callback-Funktion auszustatten, muss diese in das Interface *ICallbackEventHandler* eingebunden werden. Dieses Interface definiert nur zwei Funktion, das *RaiseCallbackEvent*- und *GetCallbackResult*- Ereignis.

```
Partial Class chat_aspx
    Implements ICallbackEventHandler
...
Oder...
<%@ Implements Interface="System.Web.UI.ICallbackEventHandler" %>
```

**Listing 10.10** Callback-Funktion wird in Seite implementiert

Dabei ist es egal, ob Sie dies für eine ASP.NET-Seite oder ein Benutzer-Steuerelement implementieren.

---

**ACHTUNG** Schnittstellennamen sind case-sensitiv.

---

Das eben implementierte Interface erzwingt die Definition einer Methode *RaiseCallbackEvent* im Code. Dabei ist ähnlich wie bei einem Webserver-Steuerelement ein *EventArgs* nötig, indem dann der Inhalt der Nachricht des Clients als Zeichenkette geliefert wird.

```
Public Sub RaiseCallbackEvent(ByVal eventArgument As String) _  
    Implements ICallbackEventHandler.RaiseCallbackEvent  
    ...'ein wenig Logik  
End Sub
```

**Listing 10.11** Die noch leere Ereignis-Methode für den Callback

Weiterhin muss für die Rückgabe des Wertes noch *GetCallbackResult* als Funktion implementiert werden.

```
Public Function GetCallbackResult() As String _  
    Implements ICallbackEventHandler.GetCallbackResult  
    ...'ein wenig Logik  
End Function
```

**Listing 10.12** Die noch leere Ereignis-Methode für den *GetCallbackResult*

Außerdem werden noch drei Client Scripte benötigt, wovon zwei den Aufruf zum Server handhaben und eine die Rückgabe verarbeitet.

## Die Chat-Seite

Wie in der Einleitung schon beschrieben, werden einige Textboxen in der Webseite benötigt. Eigentlich werden nur normale HTML INPUT-Elemente benötigt, da am Server kein direkter Zugriff auf die Steuerelemente durchgeführt wird. Für unser Beispiel kommen allerdings ASP.NET Textbox-Steuerelemente zum Einsatz. Es wird dadurch ein wenig komplizierter, weil die UniqueIDs der erzeugten Steuerelemente benötigt werden, um diese später per JScript lesen und schreiben zu können. Normale HTML-Elemente behalten auch bei Masterpages ihre ID.

---

**ACHTUNG** Verwenden Sie für das Absenden der Chat-Nachricht kein ASP.NET Server Control, da dieses einen Postback auslöst.

---

Egal ob per Link oder einem Button, es wird über den Funktionsnamen im Attribut *onClick* die Funktion *ServerCallBack()* aufgerufen.

```

<table><tr><td style="width: 100px" valign="top">
<asp:TextBox ID="txtVerlauf" Runat="server" Width="447px" Height="385px" TextMode="MultiLine"></_
asp:TextBox></td>
<td style="width: 100px" valign="top" bgcolor="#ff9933">
Hier geben Sie Ihren Alias ein&nbsp;nbsp;nbsp;
<asp:TextBox ID="txtAlias" Runat="server" Width="71px" Height="22px" EnableViewState="false"></_
asp:TextBox>
<br /></td></tr>
<tr><td style="width: 100px" valign="top">
<asp:TextBox ID="txtChat" Runat="server" EnableViewState="false"></asp:TextBox>
<input id="Button2" type="button" value="chat!" onclick="ServerCallBack();" />
</td><td style="width: 100px">
</td></tr></table>

```

**Listing 10.13** Das Design des Chats

## Client-Code am Server erzeugen

Der nächste wichtige Teil passiert am Client in mehreren Scripten. Man kann die Scripte auch direkt in die Seite schreiben. In unserem Beispiel werden die Scripte am Server erzeugt und per *RegisterClientScriptBlock* registriert. Da die Client-Scripts auf die IDs der erzeugten Server-Steuerelemente zugreifen müssen, ist es sinnvoll, diese IDs per *UniqueID*-Eigenschaft auszulesen. Speziell mit Masterpages weichen die erzeugten IDs von den benannten IDs ab.

In der Regel werden am Client drei Funktionen benötigt:

Funktion	Verwendung	Name im Beispiel
<i>Callback</i>	Diese Funktion wird vom Benutzer aufgerufen, indem er auf einen Button drückt. Dieser Button hat diese Funktion z.B. mit dem Attribut <i>OnClientClick</i> als Client-Script hinterlegt. Die CallBack-Funktion ruft dann die vom Server automatisch erzeugte CallBack-Funktion auf. Als Parameter werden Control-Wert und Control-ID übergeben.	<i>ServerCallback</i>
<i>Return</i>	Wenn der Code am Server fertig ausgeführt wurde, wird am Client die <i>Return</i> -Funktion gestartet. Dieser werden die Parameter Ergebnis und Kontext übergeben. Mit diesen Werten kann dann die Darstellung im Browser per HTML DOM geändert werden.	<i>ReturnHandler</i>
<i>Error</i>	Wenn ein Fehler im CallBack-Mechanismus aufgetreten ist, wird diese Funktion gestartet. Die beiden Parameter sollten den Fehler als Text und den Kontext zurückliefern.	<i>Fehler</i>

**Tabelle 10.1** Die drei Client-Funktionen für Callback

Am besten erzeugt man die benötigten Scripts im *Page\_Load*-Ereignis.

Die Funktion *ServerCallback* löst den Callback aus. Die eigentliche Rückruffunktion mit dem Namen *genjs-Callback* wird innerhalb des *ServerCallback*-Script-Blocks aufgerufen. Dabei wird der eingegebene Chattext übergeben sowie das Control, aus dem dieser stammt. Die Funktion *genjsCallback* wird automatisch generiert. Wie das funktioniert, wird weiter unten beschrieben.

Nachdem am Server alles abgearbeitet worden ist, wird am Client die Funktion *ReturnHandler* ausgeführt. Der Rückgabewert wird dann zum Füllen des Chatverlaufs verwendet.

Die Funktion mit dem Namen *Fehler* erhält als Parameter den Fehlercode und den Kontext, der den Fehler verursacht hat.

Nachdem die drei Scripts mithilfe von String-Operationen zusammengesetzt worden sind, werden diese mit *RegisterClientScriptBlock* in die Seite eingebaut.

```
Dim jsReturn As String
Dim jsError As String
Dim jsCallback As String
jsError = "function Fehler(result, context){alert('Fehler:'+result+context);}"
jsReturn = "function ReturnHandler(result, context)" &
"{var verlauf=document.getElementById('" & txtVerlauf.UniqueID & "');" & _
"verlauf.value=result;" & _
"verlauf.scrollTop=9999;" & _
"}"
jsCallback = "function ServerCallback()" &
"var cont =document.getElementById('" & txtChat.UniqueID & "');" &
"var alias =document.getElementById('" & txtAlias.UniqueID & "');" & _
"var sendback =alias.value+' '+cont.value;" & _
"cont.value='';" & _
"genjsCallback(sendback,cont.id);}"
ClientScript.RegisterClientScriptBlock(Me.GetType, "Error", jsError, True)
ClientScript.RegisterClientScriptBlock(Me.GetType, "Return", jsReturn, True)
ClientScript.RegisterClientScriptBlock(Me.GetType, "ServerCallback", jsCallback, True)
```

**Listing 10.14** Die Hilfs-Scripts werden erzeugt und registriert

## CallbackReference am Server

Der zentrale Teil ist mit drei Zeilen Code abgearbeitet. Dazu dient die Funktion *GetCallbackEventReference*. Diese liefert einen String zurück, mit dem die Callback-Funktion *genjsCallback* zusammengesetzt wird. Dieses Script wird dann im Client mit *RegisterClientScriptBlock* eingebunden.

```
Dim scrPart As String = GetCallbackEventReference(Me, "arg", "ReturnHandler", "ctx", "Fehler")
Dim scrBack As String = "function genjsCallback(arg,ctx){" + scrPart + "};}"
ClientScript.RegisterClientScriptBlock(Me.GetType(), "genjsCallback", scrBack, True)
```

**Listing 10.15** Callback Handler definieren

Die Funktion *GetCallbackEventReference* existiert in vier Überladungen, die alle eine Zeichenkette zurückliefern.

```
GetCallbackEventReference(control,argument,clientCallback,context)
GetCallbackEventReference(control,argument,clientCallback,context,clientErrorCallback)
GetCallbackEventReference(control,argument,clientCallback,context,clientErrorCallback,useAsync)
GetCallbackEventReference(target, argument, clientCallback, context, clientErrorCallback)
```

**Listing 10.16** Die vier möglichen Parametrisierungen von *GetCallbackEventReference*

Parameter	Bedeutung
<i>Control</i>	Das Control, das das Callback-Event implementiert. Ein passender Wert für den Parameter ist <i>Me</i> .
<i>Target</i>	Die ID des Controls, an das der Callback zurückgesandt werden soll.
<i>Argument</i>	Mit diesem Parameter wird ein String übergeben, der den Namen für den <i>EventArgs</i> -Parameter darstellt.

**Tabelle 10.2** Die Parameter der *CallbackEventReference*-Funktion



Parameter	Bedeutung
<i>clientCallback</i>	Der Name der Funktion als String, die nach dem erfolgreichen Callback im Client ausgeführt wird.
<i>Context</i>	Ein String, der den Kontext als Namen darstellt, der an den Client zurückgegeben wird.
<i>clientErrorCallback</i>	Der Funktionsname als String, der nach dem fehlerhaften Callback im Client ausgeführt wird.
<i>useAsync</i>	Boolescher Wert der angibt, ob der Rückruf Ansynchron ausgeführt werden soll.

**Tabelle 10.2** Die Parameter der *CallBackEventReference*-Funktion (Fortsetzung)

## CallBack-Logik

Die Ereignis-Methode *RaiseCallBackEvent* wird beim eigentlichen CallBack ausgeführt. Verwechseln Sie das nicht mit einem Postback. Im Parameter *eventArgument* folgt in unserem Beispiel der eingegebene Chattext. Es gibt in diesem Beispiel zwei Modi. Falls keine Eingabe erfolgt ist, wird der Callback durch einen Timer am Client ausgelöst. Dann wird ein leerer Parameter übergeben. Im anderen Fall hat der Benutzer einen Text eingegeben und auf den Button gedrückt.

Es wird ein *Cache*-Objekt angelegt, das gemessen ab der letzten Benutzung 10 Minuten gültig ist und mit den Eingabedaten aktualisiert wird.

```
Public Sub RaiseCallBackEvent(ByVal eventArgument As String) Implements _
    ICallbackEventHandler.RaiseCallBackEvent
    Dim ChatText As String
    If eventArgument <> "" Then
        If Cache("chattext") Is Nothing Then
            ChatText = eventArgument + vbCrLf
            Cache.Add("chattext", ChatText, Nothing, Date.MaxValue, New TimeSpan(0, 10, 0), _
                CacheItemPriority.Normal, Nothing)
        Else
            Cache("chattext") = CStr(Cache("chattext")) + eventArgument + vbCrLf
        End If
    End If
End Sub
```

**Listing 10.17** Die CallBack-Methode am Server

Die *GetCallBackResult*-Funktion liefert einen String zurück, der am Client per JScript weiter verarbeitet wird. Auf diese Weise wird dann der Chatverlauf inhaltlich aufbereitet.

```
Public Function GetCallBackResult() As String Implements ICallbackEventHandler.GetCallBackResult
    Return CStr(Cache("chattext"))
End Function
```

**Listing 10.18** Die Rückgabe-Funktion

## Timer

Für den Einsatz von Callback-Funktionen ist dies zwar nicht mehr wichtig, aber um den Chat zu kompletieren, kommt noch ein Timer hinzu. Damit wird die Anzeige in regelmäßigen Abständen erneuert. Allerdings ist die Lösung ein wenig trickreich, sodass sie durchaus interessant ist. Zuerst wird mithilfe von JScript ein

Timer gebaut, der alle 10.000 Millisekunden (10 Sekunden) ausgeführt wird. Gleichzeitig wird die eigentliche Callback-Funktion *genjsCallBack* mit einem Leerstring als Parameter aufgerufen.

```
jsPoll = "function timer(){window.setTimeout('timer();',10000);genjsCallBack('');}timer();"
```

**Listing 10.19** Ein Timer mit JScript

Hier beginnt der erste Trick. Zuerst wird die Timer-Funktion deklariert und erst danach der erste Aufruf von *genjsCallBack* ausgeführt.

Außerdem ist es wichtig, dass dieser JScript-Code der absolut letzte auf der Webseite ist. Jeder andere JScript-Code muss vorher ausgeführt werden, da sonst eventuell nötige Deklarationen für den Callback-Mechanismus noch nicht gemacht sind. Das Ziel ist also ein HTML-Code, der ungefähr so aussieht:

```
<script type="text/javascript">
<!--
var pageUrl='/ASPNET20CC/10/ChatClientCallBack.aspx';
WebForm_InitCallback();
function timer()
{
window.setTimeout('timer();',10000);
genjsCallBack('');
}
timer();
var ct100_Menu1_Data = new Object();
ct100_Menu1_Data.disappearAfter = 500;
// -->
</script>
</form>
</body>
</html>
```

**Listing 10.20** Quellcode im Browser

Deshalb muss die Funktion *RegisterStartupScript* verwendet werden. Aber vergessen Sie nicht, diese erst zuletzt in der *Page\_Load*-Funktion aufzurufen!

```
...
ClientScript.RegisterStartupScript(Me.GetType, "TimerEvent", jsPoll, True)
End Sub
```

**Listing 10.21** Listing 10.21:Das TimerEvent wird als letztes JScript registriert

## Der fertige Chat

Nun kann es schon losgehen mit dem ersten Chat. Man verwendet dazu am besten zwei Browser-Instanzen, um den Effekt auch richtig demonstrieren zu können.

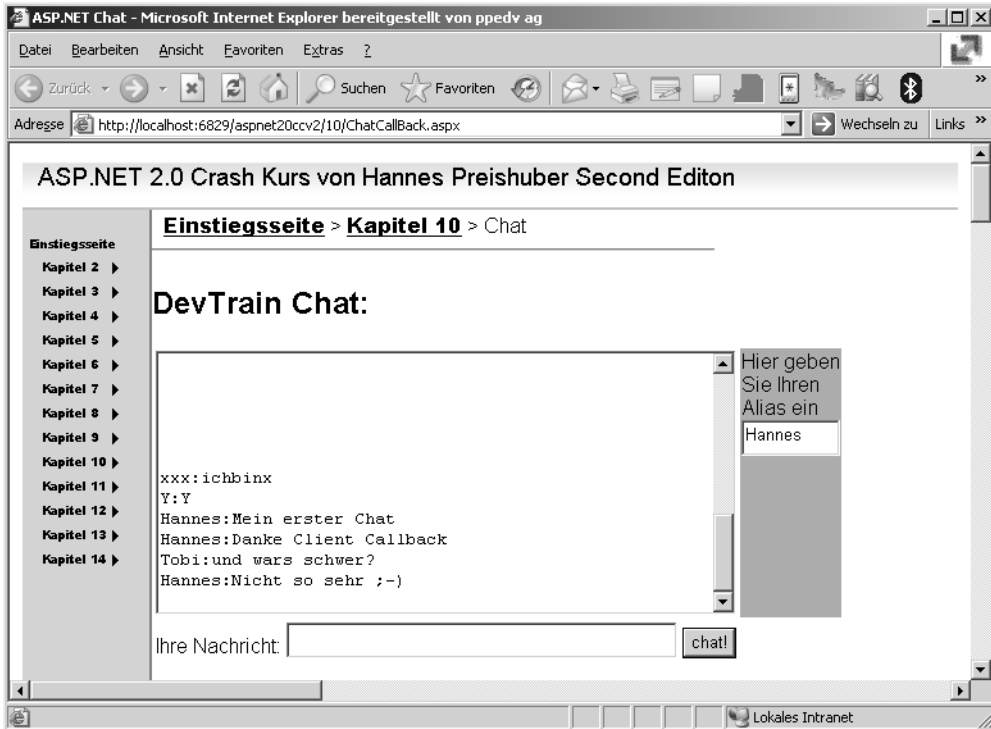


Abbildung 10.2 Der fertige Chat

Da der Callback-Mechanismus auf der MSXML-Engine aufsetzt, funktioniert dieser nur im Internet Explorer. Auch andere Steuerelemente, wie das *TreeView*-Steuerelement, machen für das Nachladen von Knoten davon Gebrauch.

Wenn ein Browser dieses Feature nicht unterstützt, muss dies in der Programmlogik berücksichtigt werden. Dazu verwendet man eine *If*-Verzweigung auf die boolesche Eigenschaft *SupportsCallback*:

```
Request.Browser.SupportsCallback
```

Man kann natürlich den Chat und seine Features noch hier und da verfeinern.

Trotzdem keine klassischen Postbacks mehr ausgeführt werden, ist die vom Client zum Server und umgekehrt übertragene Datenmenge nicht gering. Es wird jedes Mal der komplette ViewState gesendet, um die Integrität der Seite bewahren zu können. Das Deaktivieren des ViewStates in der Page Direktive mit dem Attribut *enableViewState* reduziert hierbei den Traffic erheblich, gerade wenn wie im vorigen Beispiel alle 5 Sekunden ein Callback ausgeführt wird. Dann sehen die im Hintergrund übertragenen Daten in etwa so aus.

```
__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=%2FwEPDwUKLTkwOTAxNTcONGQAQULY3RsMDAKTWvudTEPD2QFHkVpbN0aWVnc3N1aXR1XEthcG10ZWwgMTBcQ2hhdGQ0L84g0INiyIeKsCI8Hte3618DFg%3D%3D&ct100$ContentPlaceHolder1$txtVerlauf=&ct100$ContentPlaceHolder1$txtAlias=&ct100$ContentPlaceHolder1$txtChat=&__CALLBACKID=__Page&__CALLBACKPARAM=&__EVENTVALIDATION=%2FwEWBAKvWn0rCgKqu573BgKxtZ05DQkykNCMDwXjxZuH%2F3qJRjbeTgCwsD01zYwx
```

Listing 10.22 Mit Fiddler aufgezeichneter http-Verkehr



## Kapitel 11

# Konfiguration mit web.config

### **In diesem Kapitel:**

Allgemeines	286
Änderungen	286
Neue Bereiche	291

## Allgemeines

Die Datei *web.config* steuert die Konfiguration der Webanwendung. In früheren Webanwendungen wurde vieles über den Webserver konfiguriert. Dies ist speziell in gehosteten Umgebungen oft gar nicht möglich. Um diese Probleme zu umgehen, wird nun die Konfiguration über eine Datei im XML-Format vorgenommen. Ein weiterer Vorteil dieses Verfahrens besteht darin, dass die fertige Anwendung samt Konfiguration einfach per Copy & Paste eingesetzt werden kann. Änderungen können auch während des laufenden Betriebs vorgenommen werden.

Die Einstellungen werden mithilfe von Elementen vorgenommen. Oft hat ein Element Unterelemente oder kann per Attribute weiter konfiguriert werden. Die Bereiche hängen meist eng mit einem Steuerelement oder einer Klasse zusammen. Viele dieser Bereiche wurden hier im Buch schon an verschiedenen Stellen beschrieben.

Aus Platzgründen wird nicht die gesamte *web.config* erläutert, sondern nur die Änderungen zu ASP.NET 1.x sowie die neu hinzugekommenen Bereiche.

## Änderungen

Die gesamte Auflistung der Konfigurationsmöglichkeiten kann in der Datei *machine.config.comments* nachgelesen werden, zu finden in *WINDOWS\Microsoft.Net\Framework\v2.0.50727\config*. Dort sind alle möglichen Bereiche samt Kommentierung vorhanden. Gegenüber dem .NET Framework 1.x wurden Teile der Konfiguration aus der *machine.config* herausgenommen und in eine eigene *web.config*-Datei gelegt. Dementsprechend gibt es auch die Datei *web.config.Comments*.

## ClientTarget

Mit *ClientTarget* werden Browserkennungen (*User Agent*) in eine Kurzform umgewandelt und gemappt. Der Eintrag *downlevel* wird nun mit dem User-Agent *Generic Downlevel* verknüpft.

```
<clientTarget >  
  <add alias="downlevel" userAgent="Generic Downlevel" />  
</clientTarget>
```

**Listing 11.1** Ausschnitt aus *ClientTarget*

## Globalization

Mit dem *Globalization*-Element werden die Sprach- und Ländereinstellungen für die gesamte Anwendung vorgenommen. Das neue Attribut *enableClientBasedCulture* erlaubt es, die Einstellungen vom Browser überschreiben zu lassen. Wenn das Attribut *true* ist, werden aus dem Header die *accept-language*-Einstellungen ausgelesen. Damit werden dann die *culture*- und *uiCulture*-Werte überschrieben. Der Standardwert ist allerdings *false*. Ebenfalls neu ist das Attribut *enableBestFitResponseEncoding* mit dem Standardwert *false*.

Mit dem Attribut *ResponseHeaderEncoding* kann der ResponseHeader mit einem abweichenden Encoding versehen werden.

```
<globalization
  culture="DE"
  uiCulture="DE"
  enableClientBasedCulture="true"
  enableBestFitResponseEncoding = "false"
  fileEncoding="utf-8"
  requestEncoding="utf-8"
  responseEncoding="utf-8"
  responseHeaderEncoding="utf-8"
  resourceProviderFactoryType = "" />
```

**Listing 11.2** Globalization-Element

Dieser Konfigurationsbereich ist in ASP.NET 1.x bereits vorhanden.

## HttpHandlers

Mit den *HttpHandlers* können für bestimmte Dateien oder Dateitypen definierte Behandlungen durch Klassen durchgeführt werden. Mit *Add*, *Remove* oder *Clear* werden die einzelnen Handler angelegt oder wieder entfernt. Mit dem Attribut *verb* können die einzelnen Anforderungsmethoden *GET*, *POST* und *HEAD* angegeben werden, für die der Handler gültig sein soll. Das Sichten der Trace-Einträge ist auf diese Weise konfiguriert.

```
<httpHandlers>
  <clear />
  <add verb="*" path="trace.axd" type="System.Web.Handlers.TraceHandler" />
```

**Listing 11.3** Auszug aus *web.config*

In diesem Bereich wird auch der Zugriff auf die zahlreichen neuen Dateierweiterungen gesteuert. Der Download von MDB-Dateien wird z.B. unterbunden, indem ein *http forbidden*-Code an den Client gesendet wird.

```
<add verb="*" path="*.mdb" type="System.Web.HttpForbiddenHandler" />
<add verb="GET,HEAD,POST" path="*" type="System.Web.DefaultHttpHandler" />
<add verb="*" path="*" type="System.Web.HttpMethodNotAllowedHandler" />
```

**Listing 11.4** Weiterer Auszug aus *HttpHandlers*-Element

Mit dem optionalen Attribut *validate* wird gesteuert, ob der entsprechende Handler sofort oder erst bei Bedarf geladen wird. Da der *HTTPForbiddenHandler* mit hoher Wahrscheinlichkeit von jeder Anwendung benötigt wird, ist dieser z.B. mit dem Wert *true* vorkonfiguriert, um im Fehlerfall eine schnellere Ausführung zu erreichen.

## HttpModules

HTTP-Module sind Erweiterungen der ASP.NET-Pipeline. Jede Anforderung muss durch diese Pipeline hindurchgeleitet werden. Es sind einige neue Module in der *web.config* vorkonfiguriert. Diese Klassen werden dann von ASP.NET als Objekte instanziiert.

```
<httpModules>
  <add name="OutputCache" type="System.Web.Caching.OutputCacheModule" />
  <add name="Session" type="System.Web.SessionState.SessionStateModule" />
  <add name="WindowsAuthentication" type="System.Web.Security.WindowsAuthenticationModule" />
  <add name="FormsAuthentication" type="System.Web.Security.FormsAuthenticationModule" />
  <add name="PassportAuthentication" type="System.Web.Security.PassportAuthenticationModule" />
  <add name="RoleManager" type="System.Web.Security.RoleManagerModule" />
  <add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule" />
  <add name="FileAuthorization" type="System.Web.Security.FileAuthorizationModule" />
  <add name="AnonymousIdentification" type="System.Web.Security.AnonymousIdentificationModule" />
  <add name="Profile" type="System.Web.Profile.ProfileModule" />
  <add name="ErrorHandlerModule" type="System.Web.Mobile.ErrorHandlerModule, System.Web.Mobile,
    Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
</httpModules>
```

Listing 11.5 Auszug aus *web.config*

## HTTPRuntime

Mit *HttpRequest* wird auf die Ausführung der Anwendung Einfluss genommen. Dieser Bereich kann auch in einer *web.config* in einem Unterverzeichnis existieren.

Dies war schon in ASP.NET 1.x ein oft veränderter Bereich, da hier das Attribut *maxRequestLength* enthalten ist. Diese Einstellung beschränkt auch die Datei-Upload-Größe. Diese ist beschränkt auf 4.096 KByte, oder mit anderen Worten: Bei 4 MB Upload ist die Obergrenze erreicht.

```
<httpRuntime
  executionTimeout="110"
  maxRequestLength="4096"
  requestLengthDiskThreshold="256"
  useFullyQualifiedRedirectUrl="false"
  minFreeThreads="8"
  minLocalRequestFreeThreads="4"
  sppRequestQueueLimit="5000"
  enableVersionHeader="true"
  requireRootedSaveAsPath="true"
  enable="true" />
```

Listing 11.6 Auszug aus *machine.config*

Auch im Bereich *HttpRequest* kommen einige neue Attribute hinzu.

Attribut	Verwendung
<i>Enable</i>	Wenn dieser boolesche Wert <i>false</i> ist, wird die Anwendung nicht geladen und jede Anforderung mit einer 404er-Meldung quittiert.
<i>compilationTempDirectory</i>	Das Verzeichnis für die temporären Dateien.
<i>maxWaitChangeNotification</i>	Die Zeit in Sekunden, die maximal vergehen darf, bis nach einer Änderung die Anwendung neu gestartet wird.
<i>requestPriority</i>	Die Priorität der Anforderung in drei Stufen: <i>Normal</i> , <i>High</i> und <i>Critical</i> .
<i>requestLengthDiskThreshold</i>	Definiert in KByte den temporären Speicherplatz beim Upload.

Tabelle 11.1 Neue Attribute im *HttpRequest*-Element



Attribut	Verwendung
<code>requireRoutedSaveAsPath</code>	Dieser boolesche Wert gibt an, ob beim Upload einer Datei der Pfad ein Stammpfad sein muss.
<code>waitChangeNotification</code>	Die Zeit in Sekunden, die gewartet wird, bis nach einer Änderung die Anwendung neu startet.

**Tabelle 11.1** Neue Attribute im *HTTPRuntime*-Element (Fortsetzung)

## Pages

Alles, was sich in einer Seite in der *Page*-Deklaration einstellen lässt, kann auch direkt in der *web.config* mit dem *Pages*-Element gesteuert werden. Deshalb gibt es auch einige neue Attribute, die z.B. Master Pages oder Themes betreffen.

```
<pages
  buffer="true"
  enableSessionState="true"
  enableViewState="true"
  enableViewStateMac="true"
  autoEventWireup="true"
  validateRequest="true">
```

**Listing 11.7** Voreinstellung des *Pages*-Elements aus *web.config*

Die *Pages*-Eigenschaften können in der *web.config* der Anwendung überschrieben und ergänzt werden. Eine interessante Möglichkeit ist das Registrieren von Namensräumen; dies muss dann nicht mehr per *Imports* in der Seite selbst geschehen. Es wird einfach das *namespaces*-Element angelegt und per *Add*-Unterelement der *Namespace* eingebunden.

Auch Benutzer-Steuerelemente kann man auf ähnliche Art und Weise vorregistrieren. Das Unterelement dazu heißt *controls*.

Mit *TagMapping* ist es möglich, zur Kompilierzeit Mappings zwischen Klassen herzustellen. Dies ist für die Entwicklung von Steuerelementen von Bedeutung.

```
<pages
  buffer="true"
  enableSessionState="true"
  enableViewState="true"
  enableViewStateMac="true"
  autoEventWireup="true"
  master="~/masterpage2.master"
  smartNavigation="true"
  pageBaseType="typename, assembly"
  userControlBaseType="typename"
  validateRequest="true">
  <namespaces >
    <add namespace="System.IO"/>
  </namespaces>
  <controls>
    <add tagPrefix="uc" assembly="irgendwas" namespace="wasanderes"/>
  </controls>
  <tagMapping >
    <add tagName="?" mappedTagName="?" />
  </tagMapping>
</pages>
```

**Listing 11.8** Erweitertes *Pages*-Element in *web.config*

## WebRequestModules

Der Bereich *WebRequestModules* findet sich im Konfigurationsbereich *System.Net*. Eigentlich wird dieser nicht zwingend in der *web.config* verwendet. Trotzdem wird an dieser Stelle auf eine Änderung hingewiesen: Es kommt ein neuer Handler für FTP-Requests hinzu.

```
<webRequestModules>
  <add prefix="http" type="System.Net.HttpRequestCreator, System, Version=2.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
  ...
  <add prefix="ftp" type="System.Net.FtpWebRequestCreator, System, Version=2.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
</webRequestModules>
```

**Listing 11.9** Auszug aus *machine.config*

Dieser Bereich wird von der *WebRequest*-Klasse benutzt. Mit dieser kann am Server aus ASP.NET-Code heraus eine externe Webseite aufgerufen werden. Nun kommt auch die Möglichkeit hinzu, FTP-Abrufe auf diese Weise zu gestalten.

```
Dim request As WebRequest = WebRequest.Create("ftp://ftp.devtrain.de/samples.zip")
```

Die Klasse *WebRequest* befindet sich im Namesraum *System.Net*.

## DefaultProxy

Ebenfalls aus dem Bereich *System.Net* kommt das Element *DefaultProxy*. Wenn sich der Webserver hinter einer Firewall befindet, benötigen Programme oft einen Proxy-Eintrag. Diesen können Sie hier in der *machine.config* vornehmen. Auch hier gibt es einige neue Einträge, die sich auf die Konfigurations-Scripts des Proxy-Servers beziehen.

```
<defaultProxy>
  useDefaultCredentialsForScriptDownload="true"
  scriptDownloadInterval="60"
  scriptDownloadTimeout="60"
  <proxy proxyaddress="Http://proxy:8080"/>
    <bypasslist>
      <add address="bypassRegexString" />
    </bypasslist>
    <module
      type="System.Net.WebProxy, System, Version=2.0.0.0, Culture=neutral, _
        PublicKeyToken=b77a5c561934e089"
    />
</defaultProxy>
```

**Listing 11.10** Auszug aus einer Proxy-Konfiguration

Die Voreinstellung in der *machine.config* ist nun *usesystemdefault="true"*. Dann werden die Einstellungen aus der Netzwerk-Konfiguration verwendet.

# Neue Bereiche

Für einige Namespaces sind auch neue Konfigurationsbereiche hinzugekommen, die Sie in den folgenden Abschnitten beschrieben finden:

## Compilation

Der Bereich *Compilation* ist nicht neu, aber es gibt einige neue Unterelmente.

```
<compilation
  tempDirectory = "Verzeichnis"
  debug = "false"
  strict = "false"
  explicit = "true"
  batch = "true"
  urlLinePragmas = "false"
  batchTimeout = "900"
  maxBatchSize = "1000"
  maxBatchGeneratedFileSize = "1000"
  numRecompilesBeforeAppRestart = "15"
  defaultLanguage = "vb"
  assemblyPostProcessorType = "" />
```

**Listing 11.11** *Compilation*-Element mit möglichen Attributen

Wenn es Probleme zur Entwurfszeit der Anwendung gibt, die nur zeitweise auftreten und auf fehlende Objekte hinweisen, sollten Sie Batch-Kompilierung auf *false* setzen. Für eine laufende Anwendung bewirkt der Wert *true* eine erhöhte Geschwindigkeit.

## CodeSubDirectories

Mit dem Element *CodeSubDirectories* werden Namen von Unterverzeichnissen im Unterelement *CodeSubDirectory* benannt. Dies bewirkt, dass für jedes derartig benannte Verzeichnis ein eigenes Assembly erzeugt wird.

## BuildProvider

Im Element *BuildProvider* wird für die einzelnen Datei-Erweiterungen festgelegt, welcher Provider die Kompilierung durchführt. Auf diese Art wurde auch das nicht besonders elegante Feature *Datei ausschließen* implementiert.

```
<add extension=".exclude" type="System.Web.Compilation.IgnoreFileBuildProvider" />
```

Eine Datei mit der Erweiterung *.exclude* wird so nicht kompiliert.

## Caching

Der Caching-Bereich enthält drei Unterelmente: *Cache*, *outputCache* und *sqlCacheDependency*. Caching wurde schon sehr detailliert in Kapitel 3 behandelt, weswegen es an dieser Stelle nicht weiter ausgeführt wird, als das folgende Beispiel zu zeigen:

```
<cache
  cacheAPIEnabled="true"
  disableDependencies="false"
  disableExpiration="false"
  disableMemoryCollection="false"
/>
```

**Listing 11.12** Caching-Konfiguration

## ConnectionStrings

Im Bereich `ConnectionStrings` werden in ASP.NET 2.0 die Verbindungszeichenfolgen zentral abgelegt. Mit `Add` kann eine neuer `ConnectionString` per *name* deklariert werden. Im Attribut `ConnectionString` wird dann die Verbindungszeichenfolge deklariert und bei nur Bedarf der Datenbank Provider im Attribut `ProviderName` die Klasse spezifiziert.

Es existiert bereits ein vorkonfigurierter `ConnectionString` zu einer `SQLExpress`-Datenbank. Diese heißt `aspnetdb` und liegt im `app_data` Verzeichnis der Anwendung. Dies wird von verschiedenen Providern wie `Membership` oder `Profile` benutzt.

```
<connectionStrings>
  <clear />
  <add name="LocalSqlServer" connectionString="data source=.\SQLEXPRESS;Integrated_
    Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true" _
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

**Listing 11.13** StandardConnectionString aus `machine.config`

Wenn Sie den `ConnectionString` in Ihrer lokalen Webanwendung überschreiben möchten, können Sie entweder mit `clear` alle `ConnectionStrings` löschen oder mit `remove` einen einzelnen.

## ExpressionBuilders

Die ASP.NET Expressions sind ebenfalls in der `machine.config` konfiguriert. Aktuell sind dies drei Einträge im Bereich `expressionBuilders`.

```
<expressionBuilders>
  <add expressionPrefix="Resources" type="System.Web.Compilation.ResourceExpressionBuilder" />
  <add expressionPrefix="ConnectionStrings" type="System.Web.Compilation.Connection_
    StringsExpressionBuilder" />
  <add expressionPrefix="AppSettings" type="System.Web.Compilation.AppSettingsExpressionBuilder" />
</expressionBuilders>
```

**Listing 11.14** Auszug aus `web.config`

Eigene Expressions müssen hier konfiguriert werden.

## httpCookies

Wenn Cookies im http-Header versendet werden, kann dies ein Sicherheitsproblem darstellen. Mit dem Bereich *httpCookies* kann darauf Einfluss genommen werden. Das Attribut *httpOnlyCookies* erlaubt es client-seitigen Scripts, den Zugriff auf das Cookie zu untersagen. Dies wird mit dem Wert *true* erreicht.

Per *requireSSL* wird das Übertragen des Cookies bei einer unverschlüsselten Verbindung unterbunden.

```
<httpCookies
  httpOnlyCookies="false"
  requireSSL="false"
/>
```

**Listing 11.15** Auszug aus *web.config*

Das dritte mögliche Attribut ist *domain*. Damit wird die Gültigkeit des Cookies für die Anwendungsdomäne eingeschränkt.

## HostingEnvironment

Speziell im Hosting-Umfeld wartet ASP.NET 2.0 mit einigen wichtigen Neuerungen auf. Deshalb gibt es auch einen eigenen Bereich dafür, das Element *HostingEnvironment*. Mit dem Attribut *idleTimeout* wird die Wartezeit in Minuten angegeben, nach der die Anwendung heruntergefahren wird. Mit *ShutdownTimeOut* wird die maximale Zeit bis zum erfolgreichen Shutdown dieser Anwendung in Sekunden angegeben.

```
<hostingEnvironment
  idleTimeout="20"
  shadowCopyBinAssemblies="True"
  shutdownTimeout="30"
/>
```

**Listing 11.16** Auszug aus *machine.config*

Selten genutzte Anwendungen verschwenden somit keinen Arbeitsspeicher.

## MailSettings

Mit dem Bereich *MailSettings* wird das Senden von E-Mails gesteuert. Immer wenn mit der Klasse *MailClient* die Funktion *Send* ausgeführt wird, werden die Settings aus der *web.config* gelesen. Hier werden der Servername und der Serverport festgelegt. Auch die

Standardabsenderadresse kann mit dem *from*-Attribut angegeben werden. Diese wird nur verwendet, wenn im Code keine E-Mail Adresse definiert wurde. Wenn ein SMTP-Server eine Authentifizierung erfordert, können dazu Benutzername und Passwort übergeben werden. Dies wird im Bereich *network* vorgenommen. Zumindest der Mailserver (mit dem Attribut *host*) und der Port müssen spezifiziert sein.

```
<mailSettings>
  <smtp deliveryMethod = "Network" [Network | SpecifiedPickupDirectory | PickupDirectoryFromIis]>
  <network
    defaultCredentials = "False" [true|false]
    host = "" [String]
    password = "" [String]
    port = "25" [number]
    userName = "" [String]
    from = "" [String] />
  <specifiedPickupDirectory
    pickupDirectoryLocation = "" [String]/>
</smtp>
</mailSettings>
```

**Listing 11.17** Auszug aus *machine.config*

Für die Entwicklung von Anwendungen ist es meist nicht gewünscht, dass E-Mails sofort versendet werden. Vielleicht möchte man seine erzeugte E-Mail erst einmal ansehen. Dazu stoppt man am besten den Mailserver. Da aber die *MailClient*-Klasse versucht per SMTP-Protokoll die E-Mails zu senden, wird dann ein Fehler ausgelöst. Der Trick ist nun, die wenig bekannte *PickupDirectoryFromIIS*-Methode zu verwenden.

Wenn *PickUp* zum Einsatz kommt, werden die E-Mails einfach per Dateizugriff in das *PickUp*-Verzeichnis des SMTP-Servers gelegt. Der Mailserver muss dann allerdings auf der gleichen Maschine wie die Webanwendung liegen. Wenn der Speicherort auf einer Netzwerk-Freigabe liegen soll, kann die Anpassung daran mit dem Attribut *pickupDirectoryLocation* erfolgen.

Nach dem Test kann der Mailserver gestartet werden, und die Mails werden versandt. Wenn das nicht gewünscht ist, können diese einfach gelöscht werden.

## System.Data

Im neuen Bereich *System.Data* werden die zur Verfügung stehenden Datenbank-Provider definiert.

```
<system.data>
  <DbProviderFactories>
    <add name="Odbc Data Provider" invariant="System.Data.Odbc" support="BF"
      description=".Net Framework Data Provider for Odbc" type="System.Data.Odbc.OdbcFactory,
        System.Data, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <add name="OleDb Data Provider" invariant="System.Data.OleDb" support="BF"
      description=".Net Framework Data Provider for OleDb" type="System.Data.OleDb.OleDbFactory,
        System.Data, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <add name="OracleClient Data Provider" invariant="System.Data.OracleClient" support="1BF"
      description=".Net Framework Data Provider for Oracle"
      type="System.Data.OracleClient.OracleClientFactory, System.Data.OracleClient,
        Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <add name="SqlClient Data Provider" invariant="System.Data.SqlClient" support="1FF"
      description=".Net Framework Data Provider for SqlServer"
      type="System.Data.SqlClient.SqlClientFactory, System.Data, Version=2.0.3600.0,
        Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </DbProviderFactories>
</system.data>
```

**Listing 11.18** Auszug aus *machine.config*

## System.CodeDOM

Mit *CodeDOM* lassen sich zur Laufzeit Code-Fragmente nachkompilieren. Die Compiler-Einstellungen können Sie im Bereich *System.CodeDOM* vornehmen.

```
<system.codedom>
  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs" type="Microsoft.CSharp.CSharpCodeProvider, System,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" warningLevel="1" />
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
      type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.3600.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" />
    <compiler language="js;jscript;javascript" extension=".js"
      type="Microsoft.JScript.JScriptCodeProvider, Microsoft.JScript, Version=8.0.1100.0,
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <compiler language="vj#;vjs;vjsharp" extension=".jsl" type="Microsoft.VJSharp.VJSharpCodeProvider,
      VJSharpCodeProvider, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  </compilers>
</system.codedom>
```

**Listing 11.19** Auszug aus *machine.config*

Die weiteren Bereiche wurden bereits im Buch detailliert besprochen.





## Kapitel 12

# Allerlei Neues

### **In diesem Kapitel:**

E-Mail-Versand	298
Asynchrone Seitenausführung	301
Cross Page Posting	302
URLMappings	303
Benutzer-Steuerelement	305
SQL Server 2005 Express	305
QuickStart Tutorials	310
Migration	311
ATLAS	312
AccessMembership-Provider	313

Für eine Beschreibung des vollen Funktionsumfangs von ASP.NET 2.0 reicht der Platz in diesem Buch nicht aus. Es gibt aber viele kleinere und größere Änderungen und Neuerungen zur Vorläuferversion, die sehr nützlich sein können. Im Folgenden finden Sie eine Zusammenstellung meiner Favoriten.

## E-Mail-Versand

Das Versenden von E-Mails funktioniert mit einer speziellen Mail-Klasse. In früheren Versionen befand sich diese im Namespace *System.Web*. Da das Versenden von E-Mails eigentlich keine webspezifische Aufgabe ist, wurde diese Funktionalität in den Namespace *System.Net* verlegt. Die alte Klasse *SmtpMail* ist aber weiterhin vorhanden. Sollten Sie diese verwenden, unterweilt Visual Studio 2005 diese Codezeile grün.

Die Möglichkeiten der neuen Klasse sind jedoch erheblich erweitert und vereinfacht worden.

### SMTP-Methode

Durch die doppelte Verwendung der Bezeichnung *Mail* als Klassenname ergibt sich allerdings ein möglicher Namenskonflikt. Deshalb muss im Code unbedingt der Namensraum *System.Net* referenziert werden.

```
<%@ Import Namespace="System.Net" %>
```

Trotzdem muss der Namensraum *Net* im Code bei Verwendung der Mail-Klasse immer mit angegeben werden.

Um eine E-Mail zu versenden, benötigt man eine Instanz einer *SmtpClient*-Klasse. In ASP.NET 1.x war dies nicht nötig, da die betreffende Klasse als statisch deklariert war.

In diesem Objekt ist die Funktion *send* vorhanden, der als Parameter der Absender, der Empfänger, der Betreff und der eigentliche E-Mail-Text übergeben werden. Damit lässt sich mit zwei Zeilen Code eine E-Mail versenden.

```
Dim myMail As New Net.Mail.SmtpClient  
myMail.Send("asp@ppedv.de", "hannesp@ppedv.de", "Betreff", "Nachricht")
```

#### Listing 12.1 E-Mail versenden

Hier gibt es gleich mehrere potentielle Fehlerquellen. Wenn in der *web.config* der Hostname oder die IP-Adresse des Mailservers nicht gesetzt ist, muss dies im Code erledigt werden. Dazu wird die Eigenschaft *Host* verwendet. Dieser wird die IP-Adresse oder ein auflösbarer DNS-Name übergeben.

```
Dim myMail As New Net.Mail.SmtpClient  
myMail.Host = "localhost"  
myMail.Send("asp@ppedv.de", "hannesp@ppedv.de", "Betreff", "Nachricht")
```

#### Listing 12.2 E-Mail-Versand mit Angabe des Mailservernamens

Sie können dies auch in der *web.config* im Abschnitt *System.Net* setzen, wie in Kapitel 11 beschrieben.

Eine spezielle Methode zum Schließen der SMTP-Verbindung ist nicht vorhanden. Das SMTP QUIT-Kommando wird automatisch versandt.

## Pickup-Methode

Ein weiteres Problem tritt auf, wenn der Mailserver nicht erreichbar ist, weil z.B. der Mail-Service gestoppt ist. Aber auch das lässt sich mit dem Microsoft SMTP Server und der Mail-Klasse einfach lösen.

Üblicherweise werden E-Mails per Socket Verbindung auf Port 25 und SMTP Kommandos gesendet. Alternativ lässt sich eine E-Mail auch direkt in das Pickup-Verzeichnis des Microsoft SMTP-Servers schreiben. Dies geschieht dann per einfachem Dateizugriff, der zudem um ein Vielfaches schneller ist.

Wenn der Mail-Service wieder gestartet wird, verschiebt dieser die E-Mail-Dateien vom Pickup-Verzeichnis ins Queue-Verzeichnis, bis die Mail versandt ist.

Die Eigenschaft der *SMTPClient*-Klasse dafür lautet *DeliveryMethod*. Dieser können drei mögliche Werte zugewiesen werden.

DeliveryMethod	Bedeutung
<i>Network</i>	Es wird eine SMTP-Verbindung zum Versenden verwendet. Der Servername wird mit der Eigenschaft <i>Host</i> und der verwendete Port mit der Eigenschaft <i>Port</i> angegeben.
<i>PickupDirectoryFromIis</i>	Es wird direkt in das <i>Pickup</i> -Verzeichnis des Mailservers geschrieben. E-Mail-Versand ist somit auch dann möglich, wenn keine Netzwerkverbindung vorhanden ist.
<i>SpecifiedPickupDirectory</i>	Es kann ein spezielles Verzeichnis angegeben werden, in das die erzeugten E-Mails gelegt werden. Mit der Eigenschaft <i>PickupDirectoryLocation</i> wird dieses Verzeichnis spezifiziert. Es wird verwendet, um externe Prozesse mit E-Mail-Nachrichten zu speisen.

**Tabelle 12.1** Werte der Eigenschaft *DeliveryMethod*

## MailMessage-Objekt

Wenn Sie etwas mehr Kontrolle über den Inhalt der E-Mail haben wollen, kann auch ein *MailMessage*-Objekt erzeugt werden. So können mithilfe erweiterter Eigenschaften z.B. das Encoding der E-Mail gesetzt oder auch Dateianhänge versendet werden.

Zunächst wird eine Instanz des *MailMessage*-Objekts erzeugt. Dabei werden als Parameter jeweils die *From*- und die *To*-E-Mail-Adresse übergeben. Man kann die *From*-Adresse auch später noch setzen. Allerdings wird keine Zeichenkette zugewiesen, sondern ein *MailAddress*-Objekt.

Mit der Eigenschaft *BodyEncoding* wird das Encoding der E-Mail mitgeliefert. Die möglichen Werte finden sich in der *Encoding*-Auflistung aus dem Namensraum *System.Text*. UTF8 ist in der Regel eine passende Encodierung.

```
Dim mmsg As New Net.Mail.MailMessage("nochleer@fake.de", "redaktion@aspnet-professional.de")
mmsg.From = New Net.Mail.MailAddress("Hannesp@ppedv.de")
mmsg.Subject = "Mein Betreff"
mmsg.Body = "Mein Mail Text"
mmsg.BodyEncoding = Encoding.Unicode
myMail.Send(mmsg)
```

**Listing 12.3** Ein *MailMessage*-Objekt versenden

Die Klasse *MailAdress* verfügt noch über eine zweite nette Eigenschaft. Mit einem zweiten Parameter bei der Erzeugung der Instanz kann auch der Anzeigename angegeben werden. Dieser wird vom Mail Client verwendet, um den Absender anzuzeigen.

Durch zwei zusätzliche Zeilen Code lässt sich auch eine Datei mit der Mail versenden. Das Message-Objekt stellt dazu die *Attachments*-Auflistung bereit. Dieser können per Funktion *Add* eine oder mehrere Dateien hinzugefügt werden.

```
Dim Datei As New Net.Mail.Attachment("datei.txt")
mmsg.Attachments.Add(Datei)
myMail.Send(mmsg)
```

**Listing 12.4** Dateianhang per E-Mail versenden

Häufig werden E-Mails auch als HTML Mails, also mit HTML Inhalten versandt. Dies kann einfach mit der Eigenschaft *IsBodyHtml* definiert werden.

```
mmsg.IsBodyHtml = True
```

Etwas ungewöhnlich ist die Verwaltung der zusätzlichen Empfänger in CC (Carbon Copy) oder BCC (Blind Carbon Copy). Man muss das MailMessage-Objekt bemühen, um diese setzen zu können. Das es mehrere Empfänger sein können, ist in beiden Eigenschaften eine Auflistung vorhanden, die wie üblich per *Add* Methode um zusätzliche Einträge ergänzt werden kann.

```
mmsg.CC.Add("cc@ppedv.de")
mmsg.Bcc.Add("bcc@ppedv.de")
```

**Listing 12.5** Zusätzliche Empfänger definieren

Wie bereits erwähnt, ist das Versenden von E-Mails ein vergleichsweise langsamer Prozess. Es kann pro E-Mail durchaus zwei bis drei Sekunden dauern, bis eine Socket-Verbindung aufgebaut und selbst eine kleine E-Mail versandt ist. Je größer die E-Mail-Datei ist, desto länger bleibt die ASPX-Seite stehen.

Ganz neu hinzugekommen ist deshalb die Möglichkeit E-Mails asynchron zu versenden. Dies ist in der normalen Webseite eher weniger notwendig. Der Einsatzzweck liegt eher bei Winforms-, Command- oder Service-Programmen, die den Sende-Thread schneller wieder verfügbar haben wollen.

In diesem Beispiel wurde trotzdem ASP.NET verwendet, um die grundsätzliche Vorgehensweise zu erläutern. Über die Eigenschaft *SendCompleted* wird die Callback-Funktion definiert.

```
AddHandler myMail.SendCompleted, AddressOf fertig
myMail.SendAsync(mmsg, "Fertig Meldung")
```

**Listing 12.6** Per *Delegate* wird ein Event verknüpft

Wenn der SMTP-Vorgang abgeschlossen ist, wird die vorher definierte Funktion aufgerufen.

```
Public Sub fertig(ByVal sender As Object, ByVal e As System.ComponentModel.AsyncCompletedEventArgs)
    Response.Write("ferdich")
End Sub
```

**Listing 12.7** Funktion *Mail-Versand*

# Asynchrone Seitenausführung

Webseiten, die lang laufende Transaktionen anstoßen, können die Gesamtperformance einer Webanwendung deutlich verschlechtern. Der Grund liegt im Threading-Verhalten von ASP.NET bzw. der IIS. Speziell beim Einsatz von Web Services kann es zu längeren Laufzeiten kommen.

Bereits in ASP.NET 1.x war es möglich ASP.NET-Seiten asynchron abzuarbeiten.

Allerdings geht das nur durch Eingriffe in die Prozess-Pipeline – kurz gesagt ist das vergleichsweise kompliziert und damit nicht jedermanns Sache.

Das alles geht natürlich mit ASP.NET 2.0 viel einfacher. Die *Page*-Deklaration enthält ein neues *Async*-Attribut.

```
<%@ Page Language="VB" MasterPageFile="~/masterpage2.master" Title="Async Page" async="true"%>
```

Das sieht zwar schon beeindruckend aus, bewirkt aber so noch nichts. In der Methode *Page\_Load* müssen nämlich noch Funktionen, die den Start- und End-Handler ausführen, deklariert werden. Mit der Methode *AddOnPreRenderCompleteAsync* werden diese dann registriert.

```
<%@ Import Namespace="System.net" %>
<script runat="server">
Dim myRequest As WebRequest
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim los As BeginEventHandler = New BeginEventHandler(AddressOf BeginFunc)
    Dim ferdich As EndEventHandler = New EndEventHandler(AddressOf EndFunc)
    AddOnPreRenderCompleteAsync(los, ferdich)
    myRequest = WebRequest.Create("http://www.devtrain.de")
End Sub

Private Function BeginFunc(ByVal src As Object, ByVal args As EventArgs, ByVal cb As AsyncCallback,
    ByVal state As Object) As IAsyncResult
    Response.Write("begin" & System.Threading.Thread.CurrentThread.GetHashCode())
    Return myRequest.BeginGetResponse(cb, state)
End Function

Private Sub EndFunc(ByVal ar As IAsyncResult)
    Response.Write("end" & System.Threading.Thread.CurrentThread.GetHashCode())
End Sub
</script>
```

## Listing 12.8 Beispiel zur asynchronen Seite

Beachten Sie, dass der Thread, in dem die Page abgehandelt wird, dabei nicht blockiert wird. Es wird ein extra Thread für die Funktion *EndFunc* gestartet.

Für den Fall, dass die aufgerufene Funktion nie oder sehr spät fertig wird, ist es sinnvoll, ein Timeout zu setzen. Dazu wird das Attribut *AsyncTimeout* in der *Page*-Deklaration gesetzt.

```
<%@ Page Async="True" AsyncTimeout="10" %>
```

Dies ist nur eine der Möglichkeiten für den Einsatz asynchroner Seiten. Wenn mehrere asynchrone Tasks ausgeführt werden sollen, wird z.B. die Funktion *RegisterAsyncTask* verwendet.

## Cross Page Posting

Damit der Anwender von Seite zu Seite gelangt, gibt es verschiedene Möglichkeiten. Neben dem klassischen Hyperlink ist die häufigste Methode, per *Response.Redirect* vom Server aus den Client anzuweisen, eine andere Seite zu laden. Mit *Server.Transfer* wird auch am Server eine Umleitung auf eine andere Seite vorgenommen. In diesem Fall merkt der Client nichts davon. In der Adressleiste des Browser wird nur eine URL angezeigt. Eine weitere neue Möglichkeit ist das so genannte *Cross Page Posting*. Damit wird ein POST eines Web-Formulars auf eine andere Seite weitergeleitet und dort bearbeitet.

In Kapitel 1 wurde Cross Page Posting bereits kurz beschrieben.

Da die Zielseite in den Web Controls, z.B. Buttons, definiert wird, ist es auch möglich von einer Seite zu verschiedenen Seiten weiterzuleiten.

In der zweiten Seite ist dann in der Page-Klasse die Eigenschaft *PreviousPage* vorhanden. Damit kann erkannt werden, von welcher Seite der Post kommt. Mit der Funktion *FindControl* kann jedes Steuerelement auf der ersten Seite gefunden und dessen Werte abgerufen werden.

```
If Not Page.PreviousPage Is Nothing Then
    Dim txtBox As TextBox
    txtBox = CType(PreviousPage.FindControl("txtPage1"), TextBox)
    If Not txtBox Is Nothing Then
        lblPage1.Text = txtBox.Text
    End If
End If
```

**Listing 12.9** Inhalte von *Previous Page* auslesen

Schwieriger wird es im Zusammenspiel mit Masterseiten, da die Steuerelemente in ein *PlaceHolderControl* eingebettet sind. Dadurch muss ein doppeltes *FindControl* durchgeführt werden: Einmal für den *PlaceHolder-Container* und dann innerhalb dessen wieder für das eigentliche Control.

```
lblPage1.Text = CType(PreviousPage.Master.FindControl("ContentPlaceHolder1").FindControl("txtName"), _
    TextBox).Text()
```

**Listing 12.10** Cross Page Posting mit Master Pages

Wenn die zweite Seite typisiert wird, kann direkt auf Eigenschaften der ersten Seite zugegriffen werden. Dazu legen wir zuerst eine *Public Property* in *Page1.aspx* an.

```
<script runat="server">
    Public Property vonSeite1() As String
        Get
            Return txtName.Text
        End Get
        Set(ByVal value As String)
            End Set
    End Property
</script>
```

**Listing 12.11** Public Property anlegen

Mit einem Button wird dann auf *Page2.aspx* das Cross Posting durchgeführt.

```
</asp:TextBox><asp:Button ID="Button1" runat="server" Text="Page2" PostBackUrl="page2.aspx" />
```

In *Page2.aspx* muss dann die Seite mit der zusätzlichen Deklaration *PreviousPageType* streng typisiert werden .

```
<%@ Page Language="VB" MasterPageFile="~/all.master" Title="page 2" %>  
<%@ PreviousPageType VirtualPath="~/12/page1.aspx" %>
```

**Listing 12.12** Deklaration der *Page2.aspx*

Mit dem *PreviousPage*-Objekt kann dann direkt auf die Eigenschaft zugegriffen werden. Dabei bleiben selbst die Tücken der Masterseiten ohne Auswirkungen.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)  
    Response.Write(PreviousPage.vonSeite1)  
End Sub
```

**Listing 12.13** In *Page2.aspx* wird eine Property gelesen

---

**HINWEIS** Cross Page Posting ist kein Postback. Wenn Sie per *IsPostBack* eine Programmentscheidung treffen, erhalten Sie beim Cross Page Post den Wert *False*.

---

## URLMappings

Auch wenn man auf den ersten Eindruck glauben könnte, dass mit URLMappings ISAPI-Filter abgelöst werden, ist dem aber leider nicht so. Mit URLMappings können 1:1-Mappings zwischen einer angeforderten Datei und der angezeigten Datei definiert werden. Der Benutzer merkt nichts davon, weil er die aufgerufene URL angezeigt bekommt. Meist wird ein Query String in eine reale URL umgeleitet.

Die URLMappings werden im Bereich *urlMappings* verwaltet. Mit *Add* wird ein neues Mapping angelegt. Das Attribut *URL* gibt die vom Benutzer aufgerufene Adresse an. Im Attribut *mappedURL* steht die Adresse der Webseite, die dargestellt wird.

```
<urlMappings enabled="true">  
    <add url="~/artikel123.aspx" mappedUrl="~/artikel.aspx?id=123" />  
</urlMappings>
```

**Listing 12.14** URLMapping

Für ein schnelles Aktivieren oder Deaktivieren der Mappings besitzt das URLMappings-Element das Attribut *enabled*.

Es ist nicht möglich mit Regular Expressions die Mappings zu maskieren. Für eine dynamische Umleitung ist nach wie vor der Weg über HTTPHandler nötig.

---

**HINWEIS** Solche statischen Umleitungen verursachen oft Probleme mit eingebetteten Links oder Ressourcen wie Bildern.

---

## ASP.NET Expressions

Mit dem neuen Feature ASP.NET Expressions kann man auf Ressourcen- und Konfigurationsdaten zugreifen. Die Anwendung ist einfach und erweiterbar. Am häufigsten wird dies bei `ConnectionString`, Anwendungseinstellungen und `ResourceStrings` verwendet.

```
<%$ ConnectionStrings: Name %>
<%$ AppSettings: Name %>
<%$ Resources: Name %>
```

**Listing 12.15** Einsatzmöglichkeiten von ASP.NET Expressions

Dieser Code wird dabei in den HTML-Code eingebettet oder als Attributwert für das Web-Steuerelement verwendet.

### ConnectionsStrings

Der Speicherort für die Verbindungsdaten zur Datenbank wird in der `web.config` direkt unterhalb des Configuration-Elements abgelegt.

```
<connectionStrings>
  <add name="AppConnectionString1" connectionString="Server=localhost;User ID=sa;Database=Northwind;
    Persist Security Info=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

**Listing 12.16** `ConnectionString` in der `web.config` verwalten

Aus dem Code kann der `ConnectionString` per `WebConfigurationManager.ConnectionStrings` ausgelesen werden.

### appSettings

Diese Einstellungen werden ganz ähnlich wie die `ConnectionStrings`, allerdings im Bereich `appSettings` verwaltet.

```
<appSettings>
  <add key="hannes" value="Testwert" />
</appSettings>
```

**Listing 12.17** `appSettings` in `web.config`

### Resources

In Kapitel 2 wurden bereits die Konzepte für das Erstellen von mehrsprachigen Websites beschrieben. Auch dort kommen ASP.NET Expressions zum Einsatz. Die Werte sind dabei in den RESX-Dateien hinterlegt. Zusätzlich zum Schlüsselfeld (*Key Field*) wird noch der Name der Ressource als Parameter benötigt.

```
<asp:Literal ID="Label1" Runat="server" Text="<% $Resources:Hannes,Beschreibung%>"></asp:Label>
```

In diesem Fall findet sich der Wert in der Datei `Hannes.resx` im Feld `Beschreibung`.



## Benutzer-Steuerelement

Hier möchte ich kurz eine nette neue Möglichkeit der Verwendung von Benutzer-Steuerelementen aufgreifen. Benutzer-Steuerelemente sollten immer dort verwendet werden, wo Funktionalität sinnvoll zusammengefasst werden kann. Ein Menü oder ein Login-Dialog sind Beispiele dafür. Dateien für Benutzer-Steuerelemente enden mit ASCX und sind ASPX-Dateien sehr ähnlich. Um ein Benutzer-Steuerelement in einer ASPX-Seite zu verwenden, zieht man es einfach aus dem Projekt-Explorer auf das Web-Formular. Das muss allerdings für jede Seite extra gemacht werden.

Viel praktischer ist da nun die Arbeit mit dem Page-Element in der *web.config*. Dort kann man nicht nur komplette Namensräume registrieren, sondern auch Benutzer-Steuerelemente.

Die Benutzer-Steuerelemente müssen allerdings in einem eigenen Verzeichnis abgelegt werden. Wenn das Steuerelement im gleichen Verzeichnis wie die ASPX-Seite liegt, wird ein Laufzeitfehler ausgelöst.

Im Controls-Element können per *Add* die Benutzer-Steuerelemente hinzugefügt werden. Dabei werden genau wie bei der Verwendung direkt in der Seite die Attribute *tagPrefix*, *tagName* und *src* verwendet.

```
<pages>
<namespaces >
<add namespace="System.IO"/>
</namespaces>
<controls>
    <add tagPrefix="uc" tagName="meinControl" src="controls/testcontrol.ascx"/>
</controls>
</pages>
```

**Listing 12.18** Auszug aus *web.config*

Wenn das Steuerelement erfolgreich registriert wurde, schlägt im HTML-Source-Modus IntelliSense dieses auch in der Control-Liste vor.

## SQL Server 2005 Express

Microsoft setzt sehr stark auf die neue Datenbank SQL Server 2005 Express Edition. Bei Visual Web Developer 2005 Express Installation kann diese Datenbank automatisch mitinstalliert werden. Damit soll der Entwickler motiviert werden, SQL Express zu verwenden und damit die Schwächen von Access zu umgehen. Es handelt sich dabei um eine abgespeckte Version des SQL Server 2005. Es ist dringend anzuraten SQL Express gleich mit zu installieren, da einige der ASP.NET-Features darauf aufsetzen.

Wie auch MSDE ist SQL Express kostenlos und besitzt keine Administrationsoberfläche.

Allerdings besitzt SQL Express im Vergleich zu seinem Vorgänger einige Vorteile. Dies gilt natürlich erst recht für SQL Server 2005.

Der Umfang der TSQL Sprache ist gewachsen. Beispielhaft sei hier das Kommando ROW\_NUMBER() erwähnt. Damit kann man wesentlich effektiver Daten Paging in einem Business-Objekt implementieren.

Weiterhin ist es nun möglich statt TSQL auch managed Code im SQL Server einzusetzen.

Ein weiteres neues Feature ist das automatische Attachen der Datenbank. Damit kann durch einfaches Kopieren der Webanwendung die Datenbank gleich mitverteilt werden. Dies wird als XCOPY-Deployment bezeichnet. Die Datenbank liegt als Datei mit der Erweiterung MDF vor und wird von SQL Express beim Start der Webanwendung automatisch geladen. Die geschieht, wenn der ConnectionString einer speziellen Syntax folgt und die Datenbank mit dem Attribut *AttachDBFilename* referenziert.

```
data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true
```

Sie müssen lediglich sicherstellen, dass die MDF-Datei im Verzeichnis *APP\_DATA* liegt.

Diesen Vorgang nennt man automatisches *Attachen*. Genauso automatisch wird die Datenbank wieder *detached*, wenn die Anwendung beendet wird. Damit beendet sich auch die Benutzer-Instanz des SQL Express Servers.

Nach der Installation von SQL Express ist dieser auf integrierte Sicherheit eingestellt. Die datenbankeigene Authentifizierung mit dem beliebten sa-Account funktioniert damit nicht.

Wer in den *mixed mode* schalten möchte, kommt nicht umhin, die Registry direkt zu bearbeiten. Unter folgendem Schlüssel muss der *Key LoginMode* auf 2 gesetzt werden.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.1\MSSQLServer
```

Dann kann man auch Datenbankbenutzer anlegen und zur Authentifizierung in den Anwendungen verwenden.

**HINWEIS**

Wenn SQL Express ohne Visual Studio 2005 installiert werden soll, muss die Datei *sqlexpr.exe* ausgeführt werden. Dafür benötigt man allerdings Administratorrechte; außerdem muss vorher das passende .NET Framework installiert sein.

## Verwaltung

Obwohl es in der Grundausstattung ohne einen visuell bedienbaren Datenbank Manager auskommen muss, lässt sich SQL Express auf verschiedene Arten doch verwalten.

### SQL Server Configuration Manager

Der SQL Computer Manager ist zunächst die einzige vorhandene Menüoption. Damit lassen sich die SQL Dienste starten und beenden. Auch die Protokolleinstellungen lassen sich mit diesem auf der MMC 2.0 basierenden Werkzeug verwalten. Wenn Sie den SQL Server auch nach außen verwenden wollen, müssen Sie TCP/IP als Protokoll aktivieren.

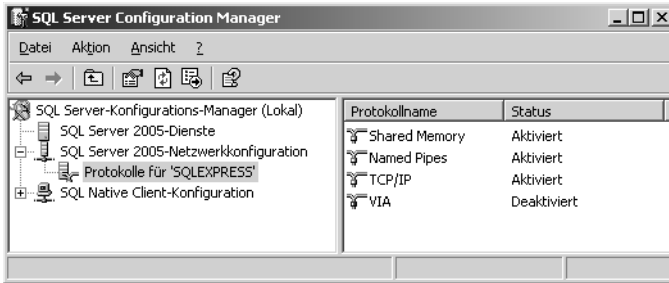


Abbildung 12.1 SQL Express Konfiguration

## Kommandozeile

Im Microsoft-Umfeld sind es die Entwickler meist nicht gewohnt, mit der Kommandozeile zu arbeiten. Trotzdem ist dies manchmal der schnellste Weg, um administrative Aufgaben zu erledigen. Zum Verwalten des SQL Express Servers muss das Werkzeug SQLCMD verwendet werden.

Dabei werden per Parameter die Startbedingungen definiert. Mit dem Attribut *S* wird der Name des SQL Express Servers festgelegt. Der Parameter *E* bewirkt ohne weitere Zusätze die Verwendung der integrierten Authentifizierung.

```
SQLCMD.exe -S (local)\SQLEXPRESS -E
```

Alles Weitere wird direkt per SQL-Kommandos in der Benutzerschnittstelle von SQLCMD durchgeführt. Mit den beiden gespeicherten Prozeduren *sp\_addlogin* und *sp\_addsrvrolemember* können dann bereits erste Benutzer angelegt und verwaltet werden.

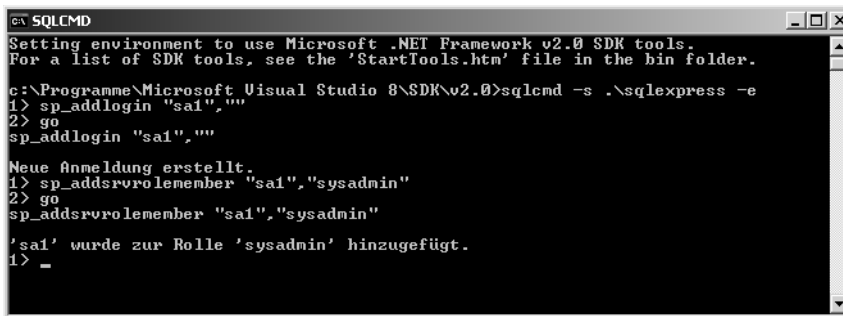


Abbildung 12.2 Kommandos mit SQLCMD

## SQL Server Management Studio Express

In den größeren Versionen des SQL Server 2005 ist eine Verwaltungsoberfläche mit dem Namen SQL Server Management Studio enthalten. Wie bereits erwähnt, fehlt diese in der Express Edition. Offenbar kommen viele Entwickler mit der Kommandozeile nicht zurecht und so bietet Microsoft eine Express Edition davon zum Download an. Zum Zeitpunkt der Drucklegung des Buches war diese nur in Englisch und als Beta erhältlich.

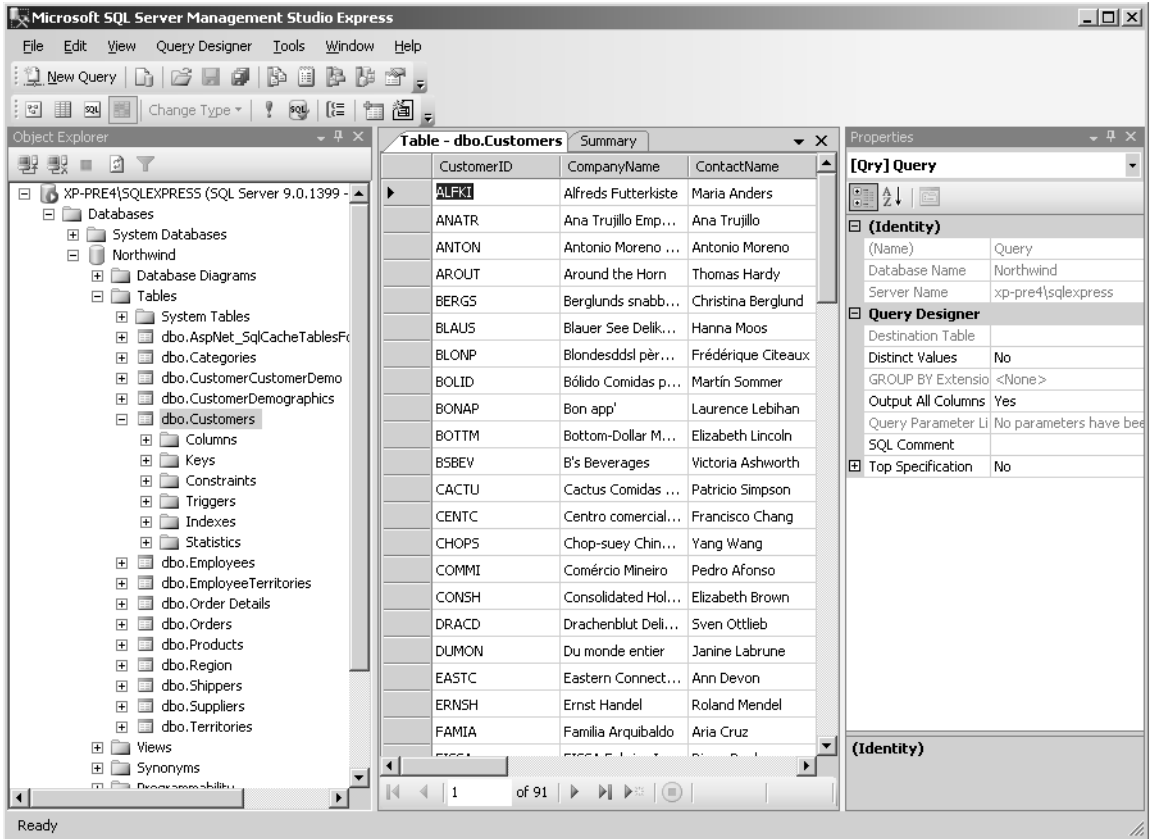


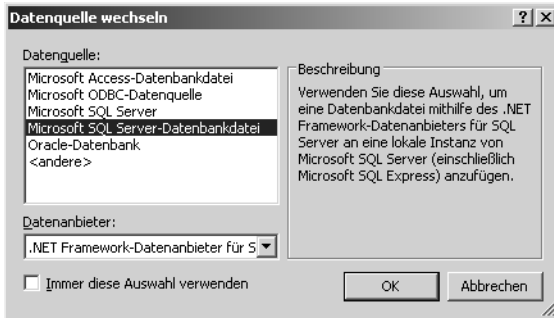
Abbildung 12.3 Management von SQL Server, wie man es sich wünscht

## SQL Express-Datenbanken im Projekt verwenden

Um einen schnellen Einstieg in die Verwendung von SQL Express zu vermitteln, werden im Folgenden die wichtigsten Schritte erläutert. Als Verwaltungswerkzeug für bestehende Datenbanken kann Visual Studio 2005 oder Visual Web Developer Express verwendet werden.

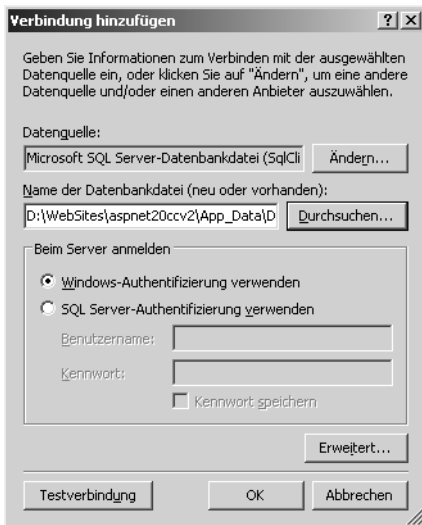
Wenn man im Projektmappen Explorer im Verzeichnis *app\_data* per Rechtsklick ein neues Element hinzufügt, kann man *Datenbank* auswählen. Damit wird eine MDF Datenbank Datei erzeugt.

Um eine z.B. in einer *SQLDataSource* eine Verbindung zu einer MDF-basierten Datenbank herzustellen, muss allerdings der spezielle *Microsoft SQL Server Datenbank Datei* Datenprovider verwendet werden. Für das folgende Beispiel fügen Sie eine Verbindung im Datenbank Explorer von Visual Web Developer Express hinzu (*mit Datenbank verbinden*).



**Abbildung 12.4** In Visual Studio Verbindung zu SQL Express-Datenbank herstellen.

Der folgende Dialog erlaubt die Auswahl einer MDF-Datei. Diese wird dann als Datenbank für diese Connection verwendet. Verwenden Sie im ersten Schritt die Windows-Authentifizierung.



**Abbildung 12.5** Neue Connection im Server Manager einrichten

Nach erfolgreichem Abschluss der Arbeiten muss im Datenbank Explorer von Visual Web Developer Express die Datenbank sichtbar sein. Dort können jetzt Änderungen an den Daten vorgenommen werden.

## ConnectionString

Um diese Datenbank im Code verwenden zu können, braucht man natürlich auch einen ConnectionString. Dieser hat im Attribut *AttachDbFilename* als wesentlichen Bestandteil die Referenz auf die MDF Datei.

```
Data Source=.\SQLEXPRESS;AttachDbFilename=D:\WebSites\ASPNET20CC\App_Data\ASPNETDB.mdf;Integrated Security=True;User Instance=True
```

**HINWEIS** Auch ältere Programme können auf die SQL Express-Datenbank zugreifen. Dazu wird der SQL Native Client verwendet. Die Verbindung erfolgt dann per ODBC oder OLE DB.

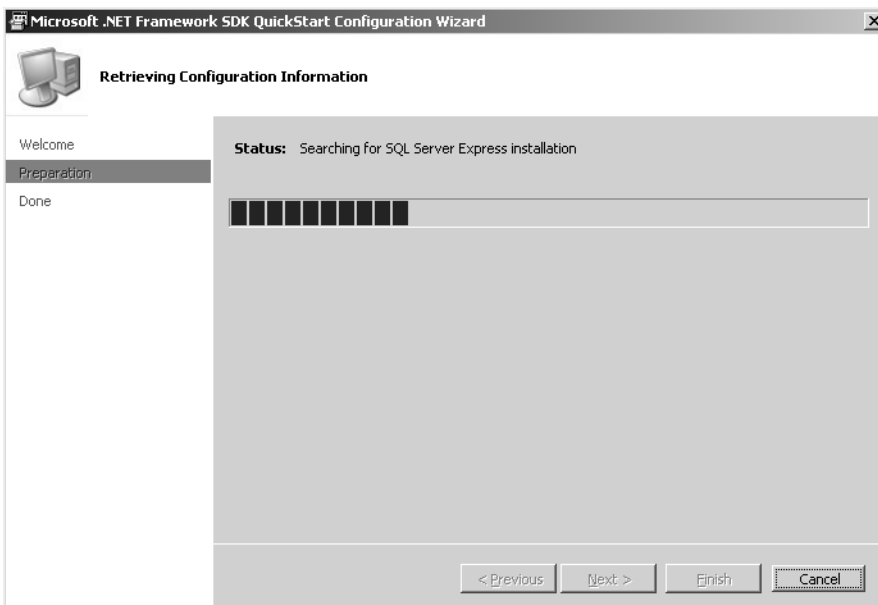
## QuickStart Tutorials

Die QuickStart Tutorials sind bereits seit Version 1.0 Bestandteil des .NET Framework SDK. Das SDK ist zwischen dem reinen .NET Framework und Visual Studio angesiedelt. In Visual Studio ist das SDK bereits beinhaltet. Alternativ steht das SDK auch zum kostenfreien Download zur Verfügung. Außerdem können diese Online genutzt werden unter <http://www.asp.net/QuickStart/>.

Mit den QuickStart Tutorials kann man sehr gut einzelne Anwendungsfälle erarbeiten. Es ist jeweils der Quellcode in VB.NET und C# vorhanden. Die Beispiele sind funktionsfähig und direkt per Link aufrufbar. Somit stellen die QuickStart Tutorials eine gute Quelle für den Schnellstart für ASP.NET 2.0-Entwickler dar.

Doch zunächst müssen ein paar vorbereitende Schritte unternommen werden: Im Startmenü unter dem Punkt *Alle Programme/Microsoft .NET Framework SDK v2.0- QuickStart Tutorials* erfolgt der erste Aufruf einer statischen HTML-Datei.

Dort wird die Installation und Vorbereitung durch Klick auf einen Hyperlink gestartet.



**Abbildung 12.6** SQL Express wird automatisch installiert

Dabei wird SQL Express installiert und in den IIS die entsprechenden Webs eingerichtet.

Nach erfolgreicher Installation finden sich dann unter anderem die ASP.NET 2.0 QuickStarts im Startmenü unter *Microsoft .NET Framework SDK v2.0*.

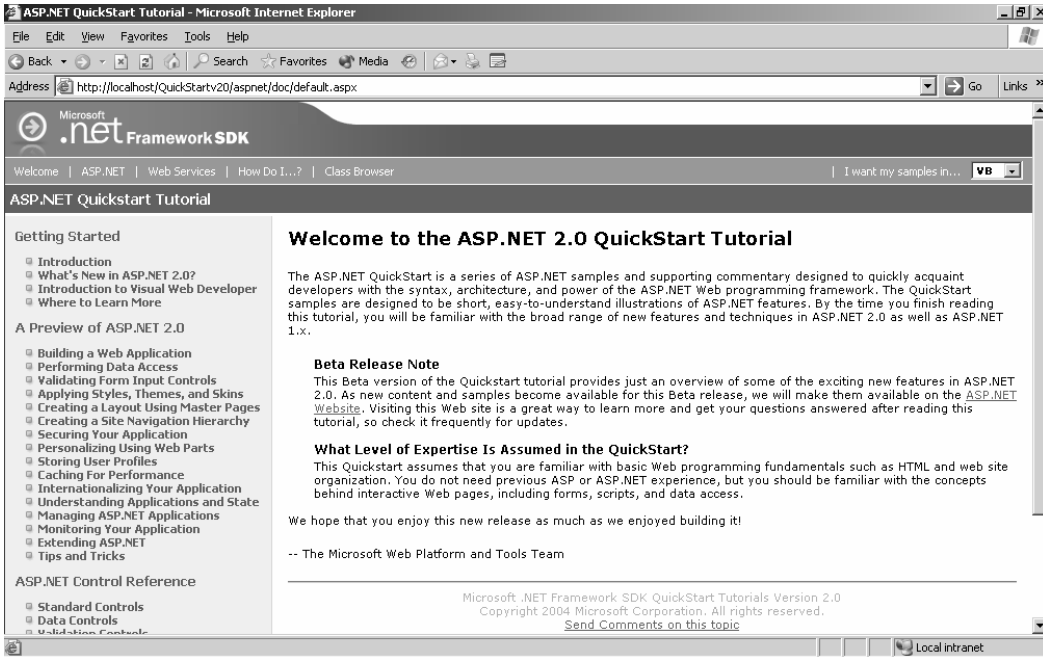


Abbildung 12.7 ASP.NET QuickStart Tutorial hilft beim Einstieg

## Migration

Wenn Sie ein Web Projekt mit Visual Studio .NET 2003 erstellt haben und dieses mit Visual Studio 2005 öffnen, wird es automatisch konvertiert.

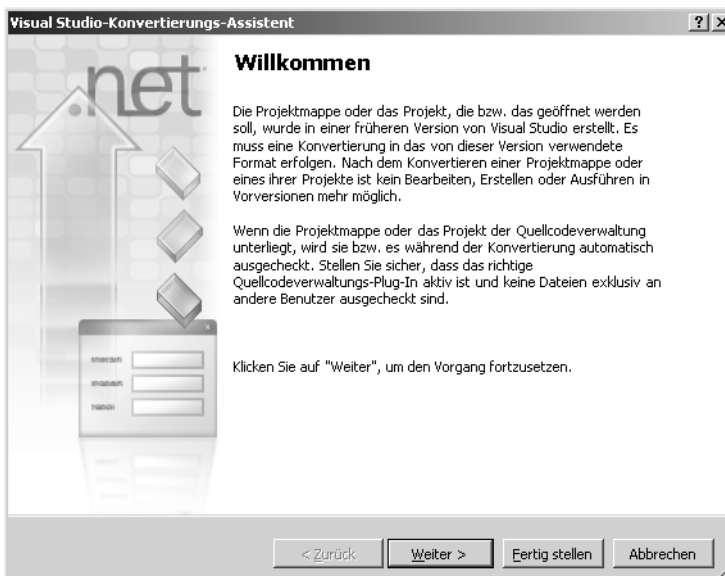


Abbildung 12.8 Ein Web-Projekt wird konvertiert

Im Grunde sollte dies auch ohne Probleme vonstatten gehen. Allerdings sind einige Probleme bekannt. So wird vor allem bei der Verwendung von Benutzer-Steuerelementen, deren interne Steuerelemente nach außen durch Public bekannt gemacht worden sind, der Konvertierungs-Assistent überfordert. Weiterhin führt es zu Problemen, wenn mehrere ASPX-Seiten dieselbe Codebehind-Datei verwenden.

Der Konvertierungs-Assistent erstellt in jedem Fall am Ende einen Report in dem alle Problempunkte aufgelistet sind. Sie finden die Datei unter dem Namen *conversionreport.txt*.

## ATLAS

Im Februar 2005 kam plötzlich der Begriff AJAX auf. Dies steht für asynchrone JavaScript Callbacks. Da Google & Co gleichzeitig eine Reihe von Anwendungen bereitgestellt hat, die eindrucksvoll beweisen, welche Möglichkeiten im Browser vorhanden sind, erfährt diese Technologie großen Zuspruch. Dabei ist es eigentlich eine Erfindung von Microsoft, die mit dem XMLHttpRequest-Objekt vor vielen Jahren den Grundstein dafür gelegt hat. Nun stand Microsoft mit seinem fast fertigen ASP.NET 2.0 und dem darin enthaltenen vergleichsweise bescheidenen Client Callback vor einem Problem. Kurzerhand wurde bereits zur PDC im Oktober ein Preview einer Microsoft Implementierung von AJAX mit dem Codenamen ATLAS vorgestellt.

So kann man, Stand Januar 2006, eine öffentlich verfügbare Beta Version von ATLAS downloaden, die eine zusätzliche Vorlage installiert.

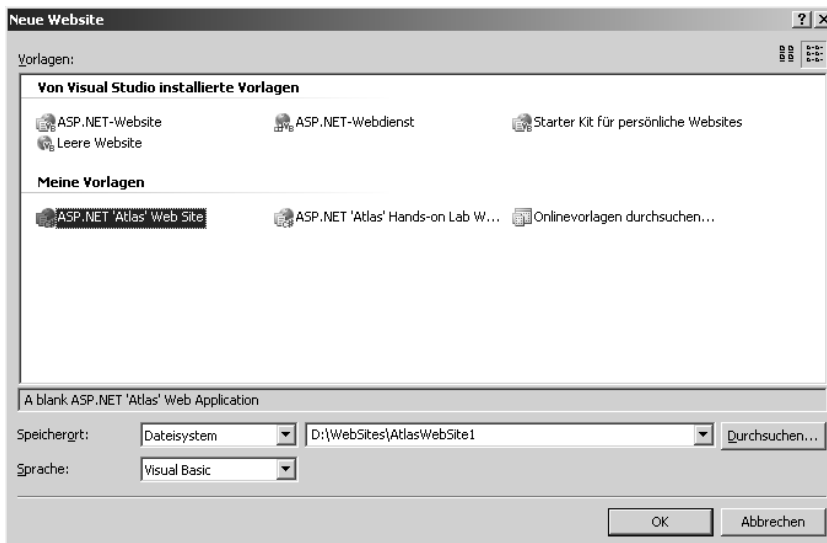


Abbildung 12.9 Neue Vorlage ATLAS

Grundsätzlich verfolgt Microsoft dabei zwei Ansätze. Mit Hilfe der umfangreichen Script-Bibliothek wird vom Client ein Web Service (ASMX) angesprochen oder direkt der Code aus den Ereignis-Behandlungsmethoden der ASPX-Seite ausgeführt.

Für letztere Methode steht ein Set von ATLAS Webserver-Steuerelementen zur Verfügung, die ergänzend zu den ASP.NET Webserver-Steuerelementen eingesetzt werden. Dabei wird zumindest ein Atlas:ScriptManager benötigt.



```
<atlas:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="True">
</atlas:ScriptManager>
<atlas:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:Button ID="Button2" runat="server" OnClick="Button2_Click" Text="Button">
  </ContentTemplate>
</atlas:UpdatePanel>
```

**Listing 12.19** ATLAS-Beispiel

Für eine derartige Seite wird kein Postback mehr ausgeführt.

## AccessMembership-Provider

Die Membership-Funktionen von ASP.NET erlauben es ohne Code die häufigsten Aufgaben in Zusammenhang mit Personalisierung von Webseiten durchzuführen. Realisiert wird dies mit einem Provider-Konzept, genauer dem SQLMembership-Provider. Dieser ist so vorkonfiguriert, dass er die SQLEXPRESS-Instanz verwenden möchte. Es lässt sich zwar mit dem Werkzeug `aspnet_regsql` auch ein normaler SQL Server als Datenbank einrichten, aber was ist, wenn dieser nicht vorhanden ist?

Ein Membership Provider ist relativ einfach selbst zu erstellen. Für die Datenbank Access liefert Microsoft einen solchen samt Quellcode mit. Dies geschieht allerdings ohne weiteren Support von Microsoft und ist auch nicht ganz problemlos.

Der Download dazu findet sich auf der Webseite <http://msdn.microsoft.com/asp.net/downloads/providers/>.

Achten Sie auf den Link zu *Sample Access Providers*. Der Link zeigt dann auf die Datei `SampleAccessProviders.vsi`, die lediglich ca. 125 KB groß ist.

Dateien mit der Endung VSI sind eine Erweiterung von Visual Studio, um in die Entwicklungsumgebung Erweiterungen zu implementieren. Demnach kann so eine VSI Datei einfach per Doppelklick installiert werden. Die Installation erfolgt dann in einem Starter Kits Ordner.

Nun folgt die Handarbeit.

Um das Projekt zu kompilieren, wird Visual C# 2005 Express benötigt. Wenn Sie Visual Studio 2005 in einer höheren Version besitzen, können Sie diese natürlich dafür benutzen.

Dazu wird ein neues C# Projekt erstellt mit dem Projekttyp Starter Kits. Darin findet sich die Vorlage ASP.NET Access Providers. Wählen Sie einen Namen für das Projekt wie z.B. `AccessMembershipProvider`. Der vorgeschlagene Name enthält leider Leerzeichen.

In dem so erstellten Projekt befindet sich ein Muster `web.config`, die Beispiel-Datenbank `aspnetdb.mdb` und die Source Codes. Als Erstes muss nun die Anwendung im Release Modus kompiliert werden. Dadurch wird eine DLL erzeugt mit dem Namen des Projektes. Darin enthalten sind unter anderem die Klassen für Membership, Rollen und Profile.

Diese DLL muss nun in das BIN Verzeichnis der Webanwendung kopiert werden. Wenn noch kein BIN Verzeichnis vorhanden ist muss dieses erstellt werden.

Die MDB Datei wird in das Verzeichnis `app_data` kopiert.

Je nachdem, welche Funktionen Ihre Webanwendung wirklich benötigt, müssen dann noch in der `web.config` Ergänzungen vorgenommen werden.

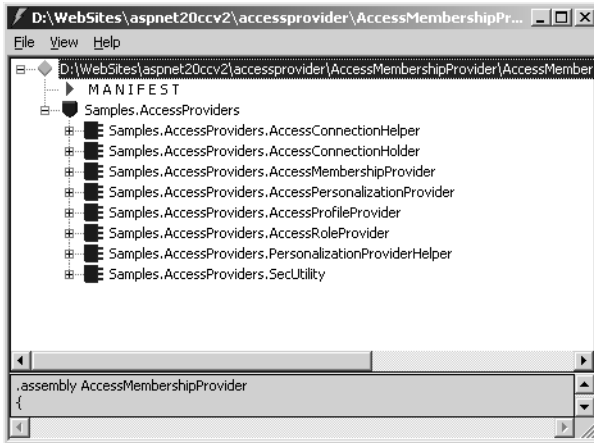


Abbildung 12.10 Die DLL mit ILDASM betrachtet

Zunächst muss der ConnectionString hinzugefügt werden, der die Lokation der *aspnetdb*-Datenbank definiert.

```
<add name="AccessFileName" connectionString="~/App_Data/ASPNetDB.mdb" providerName="System.Data.OleDb"/>
```

Wie man in Abbildung 12.10 erkennen kann, ist der Namensraum *Samples.AccessProviders*, der dann in der Konfiguration mit angegeben werden muss. Dieser wird benötigt, wenn im nächsten Schritt der Membership-Provider definiert wird. Es ist möglich mehrere Membership-Provider gleichzeitig einzusetzen. Wenn nur der Access Membership-Provider verwendet wird, sollte dieser als Default-Provider im Membership-Element deklariert werden. Dadurch erspart man sich die Definition der Provider z.B. bei den Logging-Steuerelementen.

```
<membership defaultProvider="AccessMembershipProvider">
  <providers>
    <add name="AccessMembershipProvider"
      type="AccessMembershipProvider "
      connectionStringName="AccessFileName"
      enablePasswordRetrieval="false"
      enablePasswordReset="false"
      requiresUniqueEmail="false"
      requiresQuestionAndAnswer="false"
      minRequiredPasswordLength="1"
      minRequiredNonalphanumericCharacters="0"
      applicationName=""
      hashAlgorithmType="SHA1"
      passwordFormat="Hashed"/>
  </providers>
</membership>
```

Listing 12.20 Konfiguration-Membership in *web.config*

Wenn der Default-Provider nicht gesetzt ist, erfolgt über das Attribut *MembershipProvider* im Steuerelement die Zuweisung.

```
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server"
  MembershipProvider="AccessMembershipProvider">
```

# Stichwortverzeichnis

# 268, 276  
.master 38  
.MDB 196

## A

AccessDataSource 202  
AccessDatasource 196  
Active Server Pages 16  
Add 300  
AddOnPreRenderCompleteAsync 301  
AddUserToRole 158  
AdRotator-Steuer-element 115  
AdvertisementFile 116  
AllowPaging 218  
als implizite Lokalisierung 59  
AlternateText 106  
AlternatingItemTemplate 227  
anonymousIdentification 175  
App\_code 28  
App\_data 28  
app\_Data 147  
App\_globalresources 28  
app\_GlobalResources 59  
App\_localResources 28  
app\_LocalResources 57  
App\_Themes 61  
App\_themes 28  
AppearanceEditorPart 72  
AppendDataBoundItems 112  
Application-Variablen 21  
AppSettings 304  
appSettings 304  
Architektur 92  
ASP .NET Configuration 146  
ASP.NET Caching 78  
ASP.Net debugging 255  
ASP.NET Expressions 59, 304  
ASP.NET Web Server Controls 24  
aspnet\_compiler 265  
aspnet\_regiis 96  
aspnet\_regsql 147  
ASPNETDB.MDF 147  
ASPX Seiten 27  
ASPX-Seite 315  
AssociatedControlID 106, 115  
Async 301  
AsyncTimeout 301

AttachDbFilename 309  
Attachments 300  
Authentication 142  
AutoCompleteType 107  
AutoGenerateDeleteButton 233

## B

Benutzer anlegen 148  
Benutzer anmelden 150  
Benutzer erstellen 152  
Benutzer Update 155  
Bin 28  
Bind 188, 212, 227  
Blättern 191  
BodyEncoding 299  
BodyFileName 169  
Bound Columns 220  
BulletedList Control 135  
BulletStyle 135

## C

Cache API 84–85  
Cache Variablen 22  
Cachduration 208  
CacheProfile 79  
cacheRolesInCookie 158  
Caching 78, 291  
caching 80  
Calendar Control 117  
Caption 104  
Catalog Zone 73  
CausesValidation 112  
ChangePassword Control 171  
ChangePasswordQuestionAndAnswer 156  
Chat 277  
CheckBox 114  
CheckBoxField 224  
CheckBoxList Control 133  
Client CallBack 231, 233  
ClientCallback 276  
ClientScript 268  
ClientTarget 286  
Codebehind 27  
codefile 30  
codeSubDirectories 32

CommandField 224  
 CommandName 239  
 CompareValidator 109  
 Configuration API 260  
 ConfigurationManager 200, 210  
 ConflictDetection 198  
 ConnectionStrings 304  
 Content Page 41  
 ControlParameter 199  
 Controls Element 305  
 CONTROLSTATE 100  
 Cookie 19  
 cookieless 143  
 Cookieless Forms Authentication 143  
 CookieParameter 199  
 CreateRole 160  
 CreateUser 152  
 CreateUserWizard 161  
 Cross Page Posting 33, 302  
 Cross Site Scripting 93  
 CType 98  
 Culture 55  
 CurrentUICultur 56  
 CustomValidator 109

## D

Data Caching 84, 191  
 Datafield 221  
 DataFile 202, 208  
 DataGrid 241  
 DataKeynames 203  
 Datalayer 204  
 DataList 242  
 DataNavigateUrlFormatString 226  
 DataSet 180  
 DataSourceID 182, 214  
 DataTextFormatString 226  
 DateFormatString 221  
 Datei Upload 288  
 Daten einfügen 234  
 Datenbindung 112  
 DbProviderFactories 197  
 Debug 254  
 Default Button 269  
 DefaultButton 108  
 DefaultFocus 108  
 DefaultProxy 290  
 DeleteParameter 193  
 DeleteUser 155  
 DeliveryMethod 299  
 DescriptionURL 106, 115  
 DetailsView 232  
 Direction 119

DiskCacheable 79  
 DisplayMode 65, 70, 136  
 Duration 79

## E

ECMA Script 268  
 EditItemTemplate 227  
 Editor Zone 72  
 EditorZone 72  
 Einbinden von Themes 62  
 Enable Paging 240  
 EnableCaching 191, 208  
 enableClientBasedCulture 286  
 enableExport 75  
 EnablePagingCallbacks 231, 233  
 EnablePagingCallbacks. 231, 233  
 EnablePasswordReset 168  
 EnablePasswortRetrieval 168  
 EnableTheming 62  
 Ereignisbehandlung 24  
 Erzeugen von Themes 61  
 Eval 210, 244  
 Event Logs 94  
 explizite Lokalisierung 59  
 ExportMode 74  
 ExportSensitiveDataWarning 75  
 ExportWebPart 75  
 ExpressionBuilders 292

## F

Field 18  
 Fieldset 119  
 FileUpload 122  
 FilterExpression 201  
 FilterParameters 201  
 FilterParamters 192  
 FindControl 302  
 FindUsersByEmail 153  
 FindUsersByName 154  
 Flow Layout 36  
 Focus 269  
 Focus setzen 268  
 Format Assistenten 51  
 FormatString 171  
 FormParameter 199  
 Forms Authentifizierung 143  
 FormView 239  
 Fragezeichen 142  
 Fragment Caching 81  
 Frames 36  
 FTP Requests 290

**G**

GenerateEmptyAlternateText 105  
Geschäftsobjekt 205  
GetAllProfile 176  
GetAllRows 159  
GetAllUsers 154  
GetHistory 133  
GetSectionGroup 262  
GetUser 151, 153  
GetUserNameByEmail 153  
Globalization 286  
GridView 182, 214  
GridView Events 228  
GridView-Steuerelement 176

**H**

Handles 25, 30  
HeaderText 65  
Health Monitor 265  
Health Monitoring 265  
HiddenField 19  
HiddenField Control 124  
Host Name 298  
HostingEnviroment 293  
HotSpots 124  
HTML Server Controls 121  
HTMLHead Control 137  
HTMLInputPassword 139  
HTMLInputSubmit 140  
HTMMLink 139  
HTMLSelect 122  
HTMLTitle 139  
HTTPCookies 293  
HttpHandlers 287  
HttpModules 287  
HTTPPost 19  
HTTPRuntime 288  
HyperLinkField 226

**I**

ICallbackEventHandler 277  
IDataSource 195  
Image Control 115  
ImageButton 114  
ImageMap Controls 124  
implizite Lokalisierung 59  
ImportCatalogPart 75  
ImportWebPart 76  
Inline Code 29  
InProc 99  
Internet Information Server 257  
IsClientScriptBlockRegistered 275  
IsClientScriptIncludeRegistered 275

IsOnSubmitStatementRegistered 275  
IsUserInRole 159  
ItemTemplate 227  
IWebPart 64

**J**

JScript 268

**K**

Komplierung 263  
Konfiguration 32  
Konfiguration API 261

**L**

Label Control 115  
LabelAttributes 114  
LayoutEditorPart 72  
LinkButton 114  
ListBox Enabled 113  
ListItem 133  
Localizable 58  
localize 58  
Location 79  
Log Dateien 93  
Login Control 164  
Login Web Controls 145  
LoginName Control 171  
LoginStatus Control 170  
LoginText 170  
LoginView Control 166  
LogOutImageUrl 170  
LogoutText 170

**M**

machine.config 32  
machine.config.comments 286  
Mail Encoding 299  
Mail versenden 298  
MailAddress 299  
MailMessage 299  
MailMessage Objekt 299  
MailSetting 293  
MailSettings 293  
Mappings 262  
Master & Detail 236  
Master Page dynamisch laden 43  
Master Pages 38  
MasterType 43  
maxRequestLength 124  
MDF Datei 306  
Mehrsprachige Websites 55

Membership 148  
 MemberShip API 150  
 Membership konfigurieren 150  
 MembershipCreateStatus 152  
 MembershipUserObjekt 153  
 Menü Control 49  
 MenuItemClick 52  
 MenuItemDatabound 53  
 Menüs 36  
 meta  
     localize 58  
 Meta Tag 41  
 Microsoft SQL Server Database File 308  
 MigrateAnonymous 177  
 mixed mode 306  
 mostRecent 251  
 Multiview Control 127

## N

Namensräume 31  
 Network Load Balancing 97  
 NextTextButtonText 130

## O

ObjectDataSource 196  
 OnActiveViewChanged 128  
 onClick 278  
 OnClientClick 113, 270  
 OpenWebConfiguration 261  
 Optimistic Concurrency 186  
 Orientation 52, 145  
 Output Caching 78  
 OutPutCacheProfiles 81  
 OutputCacheSettings 81

## P

Page Element 305  
 PageCount 220  
 PageIndex 220  
 PagerSettings 218  
 PagerTemplate 220, 240  
 Pages 289  
 PageSize 220  
 PageStatePersister 100  
 Paging 191, 218  
 Panel Control 118  
 parameterisierte SQL Kommandos 193  
 Partial 30  
 PartStyle 67  
 PartTitleStyle 67

passwordQuestion 152  
 Passwörter 155  
 Performance Monitor 94  
 Personalisierung 173  
 PersonalizationAdministration 70  
 Pickup Verzeichnis 299  
 PickupDirectoryFormIIS 294  
 PickupDirectoryFromIis 169  
 pickupDirectoryLocation 294  
 Portlet 63  
 PostBackURL 107  
 PostBackUrl 33  
 PostbackUrl 114  
 PreviosButtonText 130  
 PreviousPage 302  
 Previouspage 33  
 PreviousPageType 303  
 Profile Manager 176  
 ProfileManager 176  
 ProfileParameter 199  
 Profiles 174  
 protectedData 95  
 Public Property 302

## Q

Query Builder 185  
 Querybuilder 185  
 Querystring 19  
 QueryStringParameter 199

## R

RaiseCallbackEvent 278  
 RangeValidator 109  
 RedirectFromLoginPage 144  
 RegisterArrayDeclaration 274  
 RegisterAsyncTask 301  
 RegisterClientScriptBlock 272  
 RegisterClientScriptInclude 273  
 RegisterClientScriptResource 273  
 RegisterExpandoAttribute 275  
 RegisterHiddenField 273  
 RegisterOnSubmitStatement 274  
 RegisterStartupScript 272  
 RegularExpressionValidator 109  
 Remote Scripting 276  
 Remoting 97  
 RemoveUserFromRole 158  
 Repeat Layout 243  
 RepeatColumns 242  
 Repeater 244  
 requestLimit 251  
 RequiredFieldValidator 109

requireSSL 293  
Resource Manager 57  
Response.Redirect 302  
ResponseHeaderEncoding 286  
ReturnHandler 279  
Rich Client Anwendung 268  
Role Manager 156  
RoleExists 159  
roleManager 157  
Rollen zuweisen 158  
RowStyle 216  
RSS 208

## S

s 124  
sa Account 306  
Script Callbacks 276  
ScrollBars 118  
SecurityTrimmingEnabled 47  
securityTrimmingEnabled 161  
SelectedDataKey 229  
SelectedIndexChanged 228  
SelectedValue 228, 237  
SelectParameters 192  
SelectParamters 192, 199  
SendCompleted 300  
Server Objekte 21  
Server.HTMLEncode 93  
Server.Transfer 302  
Session 98  
Session Variablen 22, 98  
SessionParameter 199  
SessionState 99  
SetFocusOnError 112  
ShowSelectButton 224  
ShowStartingNode 52  
ShutDownTimeOut 293  
Sicherheit 92  
Sicherheitslücken 92  
Sitemap 161  
SiteMapDataSource 196  
SitemapDataSource 209  
SitemapDatasource 50  
SiteMapNode 46  
SiteMapPath Control 48  
SiteMapProvider 48  
SkinID 62  
Skins 61  
SmtpClient 298  
SmtpMail 298  
SOA 97  
sortieren 190  
sp\_addLogin 307  
sp\_addsrvrolemember 307  
Spracherkennung 59  
SQL Computer Manager 306

SQL Injection 92  
SQL Server 2005 Express 305  
SQLCMD 307  
SQLDataSource 181, 196  
SqlDataSource 196  
SqlDependency 79  
StateServer 99  
StaticDisplayLevel 52  
StaticDisplayLevels 51  
StepType 131  
SubmitDisabledControls 109, 121  
Substitution 82  
Substitution Control 83, 137  
Summary 105  
SupportsCallback 283  
SwitchViewByID 127  
SwitchViewByIndex 128  
System.CodeDOM 295  
System.Data 294  
Systemvoraussetzung 26

## T

Tabelle Caption 120  
Table Adapter Configuration Wizard 183  
Table Control 119  
TableAdapters 187  
TableFooterRow 105  
TableHeaderRow 105  
Template Editing 240  
Template-Spalten 227  
TextBox Control 115  
Textwriter 93  
Titel setzen 41  
Title 66  
Trace.axd 251  
Trace.Warn 250  
Trace.Write 250  
TraceFinished 254  
TransformFile 208  
TreeNodePopulate 54  
Treeview 53

## U

UI Culture 55  
UniqueID 270, 278  
UpdateParameter 193  
UpdateParameters 204  
URL Mappings 303  
UrlEncode 93  
UrlMappings 262  
urlMappingsSection 262  
UrlPathEncode 93  
UseAccessibilityHeader 106  
UseSubmitBehavior 107, 114

**V**

ValidateEmptyText 112  
Validation controls 31  
ValidationGroup 110, 121  
Validations-Steuer-elemente 109  
ValidationSummary 110  
ValueChanged 124  
VaryByControl 80  
VaryByCustom 80  
VaryByHeader 80  
VaryByParam 80  
VaryByParm 79  
Verschlüsseln 95  
Viewstate 98  
Viewstate Variablen 22  
Visual Studio 2005 26

**W**

Web Site Administrator Tool 258  
Web.Config 286  
Web.Config.Comments 286  
WebConfigurationManager 261, 304  
WebMatrix 27  
WebPartManager 64  
WebPartZone 65

WebRequestModules 290  
WebResource.axd 269  
wichtige Dateien 27  
Windows Authentifizierung 142  
Wizard Control 129  
WriteSubstitution 82–83  
writeToDiagnosticsTrace 253

**X**

XCopy 306  
XML Binden 241  
XML Control 121  
XMLDataSource 50, 208, 241  
XmlDataSource 196  
XPath 121, 208–209, 212, 241  
XPath Datenbindung 209  
XPathNavigator 121  
XP-Style 54  
XSLT 121

**Z**

Zonetemplate 66  
Zugriff auf Master Page 42