

ASP.NET- Programmierung mit VB.NET

PROGRAMMER'S

CHOICE

Die Wahl für professionelle Programmierer und Softwareentwickler. Anerkannte Experten wie z. B. Bjarne Stroustrup, der Erfinder von C++, liefern umfassendes Fachwissen zu allen wichtigen Programmiersprachen und den neuesten Technologien, aber auch Tipps aus der Praxis. Die Reihe von Profis für Profis!

Hier eine Auswahl:



Programmierung mit der .NET-Klassenbibliothek

Holger Schwichtenberg,
Frank Eller
984 S., 1 CD-ROM
EUR 59,95 [D]/EUR 61,70 [A]
ISBN 3-8273-1905-6

Licht im Dschungel der .NET-Klassenbibliothek! In diesem Buch finden Sie Objektdiagramme, Beispiele und Hintergrundinfos zu den wichtigsten Klassen z.B. für ADO.NET, XML, Serialisierung, IO, Dienste, Performance-Counter, Registry, Active Directory, WMI, Prozesse, Threading, Reflection, GUI, Kryptographie.



Webapplikationen mit Visual C#

Klaus Aschenbrenner
876 S., 1 CD-ROM
EUR 59,95 [D]/EUR 61,70 [A]
ISBN 3-8273-1966-8

Mit Erscheinen des .NET-Frameworks werden viele Dinge möglich gemacht, an die in der Vergangenheit nicht zu denken war. In diesem Buch erfahren Sie, wie Sie diese neuen Möglichkeiten effizient umsetzen können. Themen des Buches sind unter anderem Webanwendungen, Webservices, Message-Queueing, Transaktionen und .NET-Komponenten.



.NET verstehen

David Chapell
348 S.
EUR 39,95 [D]/EUR 41,10 [A]
ISBN 3-8273-2023-2

Wenn Sie eine neue Technologie anwenden wollen, müssen Sie sie zunächst verstehen. .NET verstehen liefert einen tiefen Einblick in das .NET-Framework von Microsoft. Klare Beschreibungen aus einem objektiven Blickwinkel erklären nicht nur, wie diese neue Technologie funktioniert, sondern auch, wie Sie sie effizient einsetzen können.

Jörg Krause, Uwe Bünning

ASP.NET- Programmierung mit VB.NET

Dynamische, datenbankgestützte Webseiten
mit .NET entwickeln



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <<http://dnb.ddb.de>> abrufbar.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

Bei der Zusammenstellung von Abbildungen und Texten wurde mit größter Sorgfalt vorgegangen.

Trotzdem können Fehler nicht vollständig ausgeschlossen werden.

Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.

Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig eingetragene Produktbezeichnungen oder sollten als solche betrachtet werden.

Umwelthinweis:

Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt.

5 4 3 2 1

05 04 03

ISBN 3-8273-1975-7

© 2003 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany

Alle Rechte vorbehalten

Einbandgestaltung: Christine Rechl, München

Titelbild: Campanula persicifolia, Glockenblume. © Karl Blossfeldt Archiv –
Ann und Jürgen Wilde, Zülpich/VG Bild-Kunst Bonn, 2003.

Lektorat: Sylvia Hasselbach, shasselbach@pearson.de

Korrektur: Haide Kraus-Fiebeler, Berlin

Herstellung: Monika Weiher, mweiher@pearson.de

Satz: reemers publishing services gmbh, Krefeld, www.reemers.de

Druck und Verarbeitung: Bercker Graph. Betrieb, Kevelaer

Printed in Germany

Inhalt

	Vorwort	13
	Über das Buch	17
A	Vorbereitung auf ASP.NET	27
I	Prinzip und Funktionsweise	29
1.1	Unterschiede zum alten ASP	29
1.1.1	Grundlegende Unterschiede	29
1.1.2	Vorteile für den ASP.NET-Entwickler	32
1.1.3	Eine neue Philosophie	33
1.2	Vorbereitung auf ASP.NET	35
1.2.1	Überblick über die System-Voraussetzungen	35
1.2.2	Grundlegende Komponenten eines Entwicklungssystems	36
1.2.3	Eine Website im IIS manuell einrichten	44
1.2.4	Ein erstes Beispiel ausprobieren	47
1.3	Grundlagen der Webserverprogrammierung	51
1.3.1	Das Protokoll HTTP	51
1.3.2	Wie dynamische Webseiten entstehen	56
1.4	Grundprinzip der Programmierung mit ASP.NET	59
1.4.1	Was verbirgt sich hinter dem Begriff .NET?	59
1.4.2	Überblick über das Framework	62
1.4.3	Die Welt der Objekte	64
1.4.4	Der Übersetzungsvorgang	67
1.4.5	Wie die Seite im Server verarbeitet wird	69
1.5	Programmierprinzipien	77
1.5.1	Basiselemente einer Applikation: Web Forms	77
1.5.2	Code Behind	79
1.6	Allgemeine Sicherheit im Framework	83
1.6.1	Sicherheitskonzepte	84
1.7	Hinweise zum Stil – Codekonventionen	85
1.7.1	Schreibweise von Namen im Code	85
1.7.2	Hinweise zur Benennung von Standardtypen	86
1.7.3	Hinweise für Web-Programmierer	89

2	Visual Studio .NET	91
2.1	Einführung	91
2.1.1	Ausführungen von Visual Studio .NET	91
2.1.2	Installation von Visual Studio .NET	93
2.2	Überblick über die IDE	95
2.2.1	Genereller Aufbau des Studios	95
2.2.2	Verwaltung von Projekten	97
2.3	Erste Schritte in der Programmierung	102
2.3.1	Kein <i>Hello World</i> im Studio	102
2.3.2	Ein erstes Eingabeformular	106
2.3.3	Ein erstes Datenbankprojekt	114
2.4	Fehlersuche und Debugging	128
2.4.1	Fehlerarten	129
2.4.2	Das Debuggen von Anwendungen aktivieren	131
2.4.3	Der Visual Studio-Debugger im Detail	134
2.5	Makros in Visual Studio .NET	140
2.5.1	Erste Schritte mit Makros	141
2.5.2	Der Makrorecorder	141
3	Einführung in Visual Basic .NET	143
3.1	Überblick für Umsteiger von VBScript	143
3.1.1	Variablen, Geltungsbereiche und Definitionen	143
3.1.2	Allgemeine Hinweise zur Notation	145
3.1.3	Umgang mit den Funktionen und Bibliotheken	147
3.2	Überblick über Neuerungen gegenüber VB 6	148
3.2.1	Variablen und Datentypen	148
3.2.2	Funktionen	149
3.3	Prinzip der ASP.NET-Abarbeitung	151
3.4	Spracheinführung	153
3.4.1	Schreibweise und Notation	153
3.4.2	Variablen und Datentypen	154
3.4.3	Konstanten	161
3.4.4	Kommentare	162
3.4.5	Operatoren	162
3.4.6	Verzweigungen	165
3.4.7	Schleifen	171
3.4.8	Namensräume und Klassen	176
3.4.9	Strukturen und Aufzählungen	204
3.4.10	Ausnahmebehandlung	208
4	Die Basisklassen des Frameworks	213
4.1	Schnellstart	213
4.1.1	Über dieses Kapitel	213
4.1.2	Wegweiser in die Referenz	215
4.2	Verarbeitung von Arrays	217
4.2.1	Verarbeiten von Arrays	217
4.3	Aufzählungen und Kollektionen	222
4.3.1	Einführung in die Welt der Kollektionen	223

4.3.2	Einfache Listen mit ArrayList	225
4.3.3	Schlüssel/Werte-Paare mit Hashtable speichern	228
4.3.4	Zugriff auf Aufzählungen	231
4.3.5	Wörterbücher	236
4.3.6	Spezialisierte Klassen für Wörterbücher	242
4.3.7	Spezielle Kollektionen	250
4.4	Mathematische Operationen	250
4.4.1	Felder und Methoden der Klasse <code>System.Math</code>	251
4.5	Zufallszahlen	252
4.5.1	Die Klasse <code>Random</code>	252
4.6	Zeichenketten	254
4.6.1	Verarbeitung von Zeichenketten	254
4.6.2	Zeichenketten als Ausgabeziel	261
4.7	Reguläre Ausdrücke	267
4.7.1	Überall reguläre Ausdrücke	268
4.7.2	Grundlagen regulärer Ausdrücke	274
4.7.3	Weitere Betrachtungen zu regulären Ausdrücken	278
4.8	Eigenschaften und Methoden der Datentypen	282
4.8.1	Eigenschaften der Datentypen	283
4.8.2	Methoden der Datentypen	283
4.9	Datum und Zeit	284
4.9.1	Ein Datum ermitteln	285
4.9.2	Datumsberechnungen	288
4.9.3	Datumswerte formatieren	291
4.10	Globalisierung und Mehrsprachigkeit	294
4.10.1	Grundlagen der Globalisierung	294
4.10.2	Mehrsprachige Seiten programmieren	296
4.10.3	Konfiguration in <code>web.config</code>	328
4.11	Zugriff auf das Dateisystem	329
4.11.1	Einführung	329
4.11.2	Zugriff auf Verzeichnisse und Dateien	330
4.11.3	Ausnahmebehandlung bei Dateioperationen	340
4.11.4	Dateien erzeugen und schreiben	341
5	Grundlagen der Datenspeicherung	345
5.1	Schnellstart	345
5.1.1	Über dieses Kapitel	345
5.2	Die Grundlagen	346
5.2.1	Auswahl des Datenbankmanagementsystems	346
5.2.2	Die Basistechnologien	347
5.3	SQL mit MS Access lernen	349
5.3.1	Was ist SQL?	349
5.3.2	Die Bestandteile einer Datenbank	351
5.3.3	Praktische Arbeit mit Datenbanken	358
5.3.4	Daten speichern mit <code>INSERT</code>	362
5.3.5	Daten löschen mit <code>DELETE</code>	364
5.3.6	Daten ändern mit <code>UPDATE</code>	365
5.3.7	Daten abfragen mit <code>SELECT</code>	365
5.4	Der Datenbankzugriff in Visual Studio .NET	373

5.4.1	Datenzugriff zur Entwurfszeit	373
5.4.2	Herstellen einer Datenbankverbindung	374
5.4.3	Zugriff auf die Daten in Visual Studio .NET	374
5.5	Transact-SQL – SQL mit dem SQL Server 2000 lernen	376
5.5.1	Datentypen in T-SQL	376
5.5.2	Tabellen in T-SQL erzeugen, ändern und löschen	381
5.5.3	Indizes	388
5.5.4	Funktionen in T-SQL	390
5.5.5	Arbeiten mit Datenbankzeigern	411
5.5.6	Erweiterter Umgang mit SELECT	415
5.5.7	Sichten (Views)	423
5.5.8	SQL-Programmierung mit Transact-SQL	426
5.5.9	Programmsteuerung in T-SQL-Programmen	429
5.5.10	Transaktionen	432
5.5.11	Gespeicherte Prozeduren und Funktionen	434
5.5.12	Trigger	438
5.6	Praktische Einführung in XML	441
5.6.1	Anwendung und Verarbeitung	441
5.6.2	XML-Grundlagen	443
5.6.3	XML im praktischen Einsatz	454
5.6.4	Verarbeitung einfacher XML-Daten	458
5.7	XML mit .NET verarbeiten	461
5.7.1	XML-Dokumente direkt lesen	462
5.7.2	XML selbst erzeugen	464
5.8	Daten suchen mit XPath	470
5.8.1	Eine Einführung in XPath	470
5.8.2	Mit .NET XPath-Ausdrücke verarbeiten	474
5.9	Daten transformieren mit XSLT	478
5.9.1	Eine kompakte Einführung in XSLT	479
5.9.2	XSLT-Klassen in .NET	485

B Praktische Programmierung in ASP.NET 499

6 Programmierung von Web Forms 501

6.1	Schnellstart	501
6.1.1	Über dieses Kapitel	501
6.1.2	Wegweiser in die Referenz	503
6.2	HTML Server-Steuerelemente (HTML Server Controls)	504
6.2.1	Einführung	505
6.2.2	Prinzipieller Umgang mit HTML Server-Steuerelementen	506
6.3	HTML Server-Steuerelemente im Detail	517
6.3.1	Gemeinsame Eigenschaften und Methoden	517
6.3.2	Basisoperationen mit Steuerelementen	518
6.3.3	Ereignisse verarbeiten	520
6.3.4	Gestalterische Elemente	527
6.3.5	Dateien per HTTP hochladen (Upload)	530
6.4	Web Server-Steuerelemente (Web Server Controls)	535
6.4.1	Übersicht über die Web Server-Steuerelemente	536

6.4.2	Einsatzprinzipien und Basiseigenschaften	539
6.4.3	Text auf der Seite steuern	544
6.4.4	Texteingabefelder erzeugen und auswerten	548
6.4.5	Schaltflächen erzeugen und verwenden	551
6.5	Listen-Steuerelemente	556
6.5.1	Listen erzeugen und verwalten	556
6.6	Vorlagengebundene Daten-Steuerelemente	566
6.6.1	Einführung in die Daten-Steuerelemente	566
6.6.2	Aufbau der Vorlagen in Daten-Steuerelementen	570
6.6.3	Vorbereiten einer individuellen Datenquelle	576
6.6.4	Automatische Listen mit DataList	586
6.6.5	DataGrid	605
6.7	Kontroll-Steuerelemente (Validation Controls)	616
6.7.1	Prinzipien der Feldkontrolle	617
6.7.2	Typische Kontroll-Steuerelemente	618
6.7.3	Weitere wichtige Kontroll-Steuerelemente	623
6.7.4	Selbstdefinierte Kontrollelemente (Custom Validation Controls)	629
6.8	Komplexe Steuerelemente (Rich Controls)	635
6.8.1	Kalender anzeigen mit Calendar	635
6.8.2	Weitere Steuerelemente	643
7	Erweiterte Web Form-Programmierung	645
7.1	Schnellstart	645
7.1.1	Über dieses Kapitel	645
7.1.2	Wegweiser in die Referenz	646
7.2	Modularer Code mit Benutzer-Steuerelementen	647
7.2.1	Grundlagen	647
7.2.2	Wie Benutzer-Steuerelemente entstehen	648
7.2.3	Spezielle Techniken der Benutzer-Steuerelemente	655
7.3	Browserorientiert programmieren	662
7.3.1	Umgang mit Browsern im Zusammenhang mit Clientskripten	663
7.3.2	Feststellen des Browsertyps und seiner -funktion	664
7.3.3	Interne Arbeitsweise der Browsererkennung	666
7.4	SmartNavigation	668
7.4.1	Einsatzmöglichkeiten der Eigenschaft SmartNavigation	669
7.4.2	Aktivierung und praktische Nutzung	670
7.5	Zusätzliche Steuerelemente: WebControls 1.0	676
7.5.1	Installation und Anwendungsbeispiel	677
7.5.2	TabStrip	683
7.5.3	MultiPage	685
7.5.4	TreeView	686
7.5.5	Toolbar	699
7.5.6	Weitere Funktionen der Applikation »Server Explorer«	703
7.6	Kundenspezifische Steuerelemente (Custom Controls)	706
7.6.1	Grundlagen	706
7.6.2	Zusammengesetzte kundenspezifische Steuerelemente	707
7.6.3	Erweiterte Entwicklung eigener Steuerelemente	723
7.6.4	Stile und Attribute für Steuerelemente	736
7.6.5	Die Integration ins Visual Studio	738

8	Protokollnahe und ablauforientierte Programmierung	745
8.1	Schnellstart	745
8.1.1	Über dieses Kapitel	745
8.1.2	Wegweiser in die Referenz	747
8.2	Die Welt der Standardobjekte	747
8.2.1	Die Standardobjekte	747
8.3	Daten senden und empfangen	750
8.3.1	Den Datenfluss steuern	750
8.3.2	Daten von Seiten zu Seite übertragen	755
8.3.3	HTTP-Kopfzeilen erzeugen und analysieren	759
8.3.4	Servervariablen	759
8.3.5	Den Inhaltstyp bestimmen	762
8.3.6	Anforderungen weiterleiten	767
8.3.7	Übergabe der Programmsteuerung	769
8.3.8	Kontext-Handler und Seitenreferenzierung	773
8.4	Sitzungen (Sessions)	778
8.4.1	Grundlagen	779
8.4.2	Konfiguration des Sitzungsmanagements	786
8.4.3	Sitzungsvariablen verwenden	790
8.5	Cookies	791
8.5.1	Cookies als Informationsspeicher	791
8.5.2	Cookies praktisch verwenden	797
8.6	Applikationsmanagement	802
8.6.1	Einführung in das Applikationereignismodell	802
8.6.2	Die Datei global.asax	808
8.7	Optimierung des Datenverkehrs	809
8.7.1	Caching von Seiten und Steuerelementen	809
8.7.2	Speicherung von Daten während der Laufzeit	816
8.7.3	Allgemeine Tipps zur Optimierung	820
9	Datenbanken und ADO.NET	823
9.1	Schnellstart	823
9.1.1	Über dieses Kapitel	823
9.1.2	Wegweiser in die Referenz	824
9.2	Grundlagen zu ADO.NET	825
9.2.1	Prinzip der Arbeit mit ADO.NET	825
9.2.2	Die Architektur von ADO.NET	829
9.2.3	Verbindung zu einer Datenbank aufbauen	831
9.2.4	SQL-Befehle an die Datenbank senden	842
9.2.5	Datensätze lesen	846
9.2.6	Detaillierte Informationen über eine Tabelle ermitteln	863
9.2.7	Gespeicherte Prozeduren verwenden	865
9.3	Datenverwaltung mit ADO.NET	871
9.3.1	Der Datenspeicher: <i>DataSet</i>	872
9.3.2	Die Struktur einer Tabelle im <i>DataSet</i> festlegen	884
9.3.3	Der Datenadapter (<i>DataAdapter</i>)	892
9.3.4	Datenansichten mit <i>DataView</i>	896
9.3.5	Aktualisieren einer Datenbank mit <i>CommandBuilder</i>	904

9.4	Umgang mit Datenbanken in Visual Studio .NET	921
9.4.1	Zugriff auf einen Datenbankserver	921
9.4.2	Daten in Visual Studio .NET manipulieren	923
9.4.3	Hilfsmittel zum Generieren komplexer Abfragen	923
9.5	Die Datenbindung an Steuerelemente	926
9.5.1	Die Bindung von Datenquellen	927
9.5.2	Hintergrund der Datenbindungssyntax	935
9.5.3	Ereignisbehandlung	936
9.6	Indexdienst programmieren	937
9.6.1	Motivation und Hintergründe	937
9.6.2	Der Entwurf eines Suchformulars	941
9.6.3	Ausführung der Suche	943

C Professionelle Techniken in ASP.NET

951

10 Konfiguration, Optimierung und Sicherheit

953

10.1	Schnellstart	953
10.1.1	Über dieses Kapitel	953
10.2	Konfiguration einzelner Seiten: Die Direktiven	954
10.2.1	Übersicht	954
10.2.2	Die Seitendirektive @Page	955
10.2.3	Die Direktive @Import	959
10.2.4	Die Direktive @Register	959
10.2.5	Die Direktive @OutputCache	960
10.2.6	Die Direktive @Control	961
10.3	Konfiguration von Applikationen: web.config	962
10.3.1	Prinzipieller Umgang mit Konfigurationsdateien	965
10.4	Konfiguration des gesamten Systems machine.config	966
10.4.1	Optionen des Compilers	966
10.4.2	Den ASP.NET-Worker Process konfigurieren	969
10.5	Sicherheit	973
10.5.1	Das Sicherheitskonzept des IIS	973
10.5.2	ASP.NET-Sicherheitsfunktionen im IIS	975
10.5.3	Forms-Authentifizierung	977
10.5.4	Personalisierung	984
10.5.5	Windows-Authentifizierung	985

11 Besondere .NET-Klassen

989

11.1	Schnellstart	989
11.1.1	Über dieses Kapitel	989
11.2	Bilder dynamisch erstellen	990
11.2.1	Prinzip der Bilderzeugung	990
11.2.2	Bilder direkt erzeugen	994
11.2.3	Eine kleine Grafikbibliothek	998
11.2.4	Hinweise zur Fehlersuche	1005
11.3	Versenden von E-Mail via SMTP	1006
11.3.1	Vorbereitung	1007

11.3.2	Eine E-Mail-Anwendung programmieren	1012
11.3.3	Spezielle E-Mail-Funktionen	1018
11.4	Zugriff auf andere Server	1019
11.4.1	Prinzip des Netzwerkzugriffs	1019

I 2 Servererweiterungen I 027

12.1	Schnellstart	1027
12.1.1	Über dieses Kapitel	1027
12.2	HTTP-Handler	1028
12.2.1	Vorhandene HTTP-Handler	1028
12.2.2	Erweiterung durch eigene Handler	1030
12.2.3	Programmierung eines eigenen Handlers	1031
12.2.4	HTTP-Handler ohne IIS-Konfiguration verwenden	1037
12.3	HTTP-Module	1037
12.3.1	Grundsätzliche Funktionsweise	1038
12.3.2	Anwendungsbeispiel	1039

I 3 Webservices I 043

13.1	Schnellstart	1043
13.1.1	Über dieses Kapitel	1043
13.2	Grundlagen	1044
13.2.1	Die Protokolle	1045
13.3	Webservices konsumieren	1054
13.3.1	Hinweise zu öffentlichen Diensten	1054
13.3.2	Ein einfaches Programm am Beispiel Google	1055
13.3.3	WSDL programmtechnisch auswerten	1064
13.4	Webservices anbieten	1076
13.4.1	Anregungen für den praktischen Einsatz	1076
13.4.2	Praktische Umsetzung eines Dienstes	1077

D Anhang I 085

I 4 Über die Autoren I 087

14.1	Die Autoren kurz vorgestellt	1087
14.2	Aktuelle Informationen finden Sie im Internet	1088
14.3	Informationen über professionelle Unterstützung	1089

Index I 091

Vorwort

Sie haben vermutlich bereits einiges über .NET gehört und gehören damit zu einem erlauchten Kreis. Wenn Sie dieses Buch gelesen haben, werden Sie ASP.NET – eine der derzeit fortschrittlichsten Technologien zur Webserverprogrammierung – kennen und damit zu einem noch erlauchteren Kreis gehören.

Die Domäne von Microsofts ASP-Technologie war bislang das Intranet. Mit der Zusammenführung der Webserver- und der Windows-Programmiertechnologie unter dem Dach .NET wird sich dies weiter etablieren. Will man unter Windows effektiv programmieren, ist nun das ideale Werkzeug geschaffen worden. ASP.NET ist in dieser Hinsicht ebenso konsequent wie leistungsfähig Bestandteil einer einheitlichen Programmierwelt geworden.

Inzwischen ist ASP.NET ebenso wie das gesamte Framework bereits ein Jahr verfügbar und die ersten großen und viele kleine Projekte wurden erfolgreich umgesetzt. Die Feuerprobe hat es definitiv bestanden. Die Vorteile konzentrieren sich zwar auf größere Projekte, aber auch für die kleine Applikation im Intranet ergeben sich deutlich kürzere Produktzyklen. In diesem Sinne dürfte die Zahl der Anwender rasch zunehmen.

Wer bislang mit Java und JSP gekämpft hat, dem dürfte der Einstieg in ASP.NET dagegen nicht nur leichter fallen, sondern auch eine Art Offenbarung darstellen. Denn hier ist vieles nicht nur einfacher, sondern auch schneller zu erledigen. Umsteiger von ASP werden mehr zu kämpfen haben, denn sie steigen nun in die Welt der professionellen Softwareentwickler ein. Hier sind Themen wie Objektorientierung, Debugging, Tracing, Designer usw. bestimmend. Dennoch ist mit VB.NET eine recht brauchbare Implementierung gelungen. Diese orientiert sich aber naturgemäß weit mehr an VB 6 als an VBScript. Der Anspruch ist klar: Mit .NET kann Software in besserer Qualität und dennoch in kürzerer Zeit entwickelt werden. Dem Entwickler, der lediglich das Problem

seines Kunden lösen muss, ist dabei ein einzigartiges Instrument in die Hand gegeben worden. Er muss nur darauf achten, dass niemand so etwas wie Plattformunabhängigkeit fordert, was in den meisten Fällen aber sowieso mehr eine nette Illusion ist.

.NET verlangt einen hohen Einarbeitungsaufwand. Wenn Sie heute starten, werden Sie vermutlich gut sechs Monate brauchen, bis Ihnen schnelle und elegante Lösungen aus der Hand fließen. Der Effekt: Sie werden deutlich produktiver als vorher sein. Dieses Buch bietet viel Informationen, verlangt aber auch viel. Sie müssen lernen, parallel mit der MSDN-Referenz zu arbeiten. Ohne den ständigen Zugriff auf die endlosen Listen von Namensräumen, Klassen und deren Mitgliedern ist die Chance, lauffähige Software zu erhalten, relativ gering.

Sie sollten sich auch einige Regeln zurecht legen, um effektiv lernen zu können. Hier ein paar Tipps:

- ▶ Nicht alles, was geht, muss auch verwendet werden
- ▶ Fast alles, was Sie benötigen werden, ist bereits in Form einer Klasse implementiert – erst suchen, dann selbst programmieren.
- ▶ Gute Lösungen sind klein; immer an das KISS-Prinzip denken (Keep It Small and Simple)
- ▶ Versuchen Sie die Zusammenhänge zu verstehen, auch wenn Sie nur einen Bruchteil der Prozesse durch ein Programm beeinflussen.

In diesem Sinne kann sich der Leser freuen. Das erworbene Wissen qualifiziert auch für andere Aufgaben. Der Fokus dieses Buches liegt auch auf der Vermittlung der Sprache VB.NET. Trotz der Ähnlichkeit im Namen sind die Unterschiede zu VBScript signifikant und zu VB 6 immer noch erheblich. Auch hier ist also eine Lernkurve zu bewältigen. Dafür kann man auch außerhalb von ASP.NET mit VB.NET professionelle Software schreiben. Insofern ist es mehr als nur ein einfaches Lernbuch, denn es sollte ein ganzheitlicher Ansatz vermittelt werden. Die begrenzte Seitenzahl eines Buches zwang dennoch zu einer ausschnittweisen Darstellung. Seitenlange Tabellen, die denen in der Referenz entsprechen, wurden vermieden. Ebenso sind Ihnen endlose Wiederholungen derselben Funktionen in immer neuen Varianten erspart geblieben. Vieles erschließt sich ohnehin nur, wenn man die Zusammenhänge verstanden hat. Die Vermittlung der Hinter-

gründe, egal ob es solche aus der .NET-Welt sind oder nicht, stand dann auch im Mittelpunkt der Bemühungen der Autoren.

Dieses Buch verfolgt überdies einen ganzheitlichen Ansatz auf hohem Niveau. Ein wenig Wissen über die Webserverprogrammierung müssen Sie mitbringen. Das betrifft vor allem HTML, aber auch ein wenig JavaScript ist hilfreich. Alle anderen Aspekte der Dynamik werden vermittelt, auch Randgebiete. So finden Sie im Sinne unseres Ansatzes ein Kapitel zu SQL und darin aufbauend Transact-SQL – der SQL Server 2000 steht im Mittelpunkt der Datenbank-Beschreibung. Gefolgt werden diese Grundlagen von einer Einführung in XML, in die Transformationssprache XSLT und die Syntax von XPath. Im Zusammenhang mit der Datenspeicherung ist auch von XSD (Schema) die Rede. Das alles natürlich immer mit Blick auf die Klassen des .NET-Frameworks und praktischer ASP.NET-Beispiele.

Ob die Methodik erfolgreich war, können Sie gern direkt den Autoren mitteilen. Wir freuen uns über jede Rückmeldung. Ebenso stehen wir gern zur Verfügung, wenn es um die Ausbildung von Mitarbeitern oder die Umsetzung von Projekten geht. Schreiben Sie an folgende E-Mail-Adresse: *dotnet@buenning-krause.de*.

Jörg Krause und Uwe Bünning

Berlin, im Februar 2003

Wie Sie dieses Buch verwenden sollten

Dieses Buch hat anspruchsvolle Aufgaben zu erfüllen. Es soll Sie in möglichst kurzer Zeit mit einer komplexen Materie bekannt machen. Es soll solides Basiswissen vermitteln, aber auch eine professionelle Hilfestellung für praktische Programmierprobleme bieten. Es soll ebenso ein dauerhafter Begleiter am Schreibtisch des Entwicklers sein. Die Verfasser hoffen, dass ihnen dieses Kunststück gelungen ist.

Die Teile

Das Buch ist in drei Teile und einen Anhang gegliedert. Die drei Teile trennen den Inhalt nach dem Niveau:

► **Teil A** – »Vorbereitung auf ASP.NET«

Hier werden die Grundlagen gelegt und elementare Informationen über .NET, Visual Studio .NET und die Errichtung eines Entwicklungssystems vermittelt. Ein weiteres Kapitel führt dann in die Programmiersprache VB.NET ein, gefolgt von den Basisklassen des Frameworks, die immer wieder im Buch Verwendung finden.

Da der Schwerpunkt des Buches auf der Datenbankprogrammierung liegt, finden Sie auch eine Einführung in Datenbanken und speziell in Transact-SQL. Wenn Sie damit bereits Erfahrung haben und sich in der Webserverprogrammierung auskennen, ist ein Start mit Teil B möglich.

► **Teil B** – »Praktische Programmierung in ASP.NET«

In diesem Teil des Buches geht es um die Programmierpraxis. ASP.NET wird von allen Seiten betrachtet und sehr viele Beispiele werden vorgestellt, die Formulare und andere Programme realisieren. Der Schwerpunkt liegt hier auf den Steuerelementen und den für Webserver typischen Techniken wie Sitzungen, Cookies und die Kontrolle des Ein- und Ausgabedatenstromes. Den Abschluss bildet ein umfassendes Kapitel zur Datenbankprogrammierung mit ADO.NET und den Daten-Steuerelementen.

► **Teil C** – »Professionelle Techniken in ASP.NET«

Mit dem Grundstein aus Teil A und B werden nun kompliziertere Techniken vorgestellt, die von ASP.NET exzellent unterstützt werden. Sie finden hier Informationen zur Konfigura-

tion und Sicherheit und zu speziellen .NET-Klassen. Darunter fällt beispielsweise die dynamische Bilderzeugung und das Senden von E-Mail. Den Abschluss bildet ein Ausflug in die Welt der Webservices.

Aufbau des Buches

Sie können dieses Buch von Anfang bis Ende durcharbeiten. Die Kapitel bauen weitgehend aufeinander auf. Das ist wegen der miteinander verwobenen Klassen des Frameworks nicht immer einfach. Es sind deshalb viele Verweise im Buch, denen Sie immer dann folgen sollten, wenn das angesprochene Thema für Sie neu ist. Es ist also durchaus möglich, dass Sie in Kapitel 3 auf Kapitel 11 verwiesen werden, um sich über eine spezielle Methodik zu informieren, die hilfsweise eingesetzt werden musste. Umgekehrt führen die hinteren Kapitel immer wieder zurück zum Anfang des Buches, damit Sie das frisch erworbene Wissen leicht festigen können.

**Arbeiten mit
Verweisen**

Die zweite Maßnahme zur Leitung des Lesers sind Einführungen in die Kapitel. Diese Abschnitte tragen immer den Titel »Über dieses Kapitel«. Darin ist nicht nur eine Vorschau auf den Inhalt enthalten, sondern auch eine sehr kompakte Zusammenfassung, ein grober Überblick über das Thema. Sie können von hier gezielt in die einzelnen Abschnitte springen, da zu allen Hauptüberschriften entsprechende Links eingebaut wurden.

Einführungen

Nicht zuletzt hat es sich in der Praxis als sehr wichtig herausgestellt, immer Zugriff auf die Online-Referenz oder die MSDN-Library zu haben. Es ist natürlich schwer, darin etwas zu finden, wenn man den Namen der Dinge nicht kennt, nach denen man sucht. Deshalb steht vor jedem Kapitel ein Abschnitt mit dem Titel »Wegweiser in die Referenz«, der die verwendeten Klassen, Methoden oder Kommandos in ihrem Kontext darstellt und kurze Hinweise zur Benennung gibt. Diese Darstellungen erheben keinen Anspruch auf Vollständigkeit, bieten aber durchaus beste Hilfe bei der Suche.

Referenzverweis

Beispiele

In der Wissensvermittlung war »Learning-by-Doing« schon immer einer der erfolgreichsten Wege. Damit Sie alles ausprobieren

können, werden alle vorgestellten Funktionen durch Beispiele unterlegt. Es mag sein, dass diese nicht immer von großem praktischen Nutzen sind; manchmal waren es lediglich didaktische Gesichtspunkte, die zur Darstellung führten. Egal ob konstruiert oder nicht, Sie sollten jedes Beispiel in Ihren Editor laden, verändern, anpassen und modifizieren und sorgfältig die Reaktion des Systems beobachten. Am Anfang ist es auch sinnvoll, den Code vollständig selbst einzutippen. Dies ist eine sehr sichere Methode, die Fehlermeldungen des Compilers kennenzulernen. Damit Sie bei größeren Beispielen die Lust nicht verlieren, liegt dem Buch eine CD bei, die alle Codes enthält.

Im Buch finden Sie immer wieder folgendermaßen gekennzeichneten Code:

```
' Dies ist Listingcode
Sub Prozedur
    '
End Sub
```

Listing 1: Hier steht die Listingunterschrift (Ausschnitt aus DateiName.aspx)

Solche mit einem Dateinamen gekennzeichneten Programme sind allein ablauffähig bzw. sind so gestaltet, dass sie die benötigten Zusatzdateien automatisch laden und verwenden. Es ist aber in den meisten Fällen so, dass aus Platzgründen nicht das gesamte Programm abgedruckt wurde. Oft verwenden alle Listings eines Abschnitts immer wieder dieselben Standardcodes am Anfang, beispielsweise den Import von Namensräumen. Dies wird nicht immer wieder abgedruckt. Sie müssen deshalb, wenn Sie Listings abtippen, an den Anfang des Kapitels blättern und dort nachschauen, was fehlt. Derartige »unvollständige« Codes erkennen Sie daran, dass der Dateinamen folgendermaßen geschrieben wird: (Ausschnitt aus DateiName.aspx). Steht nur (DateiName.aspx), ist das Programm vollständig.

ASP.NET erlaubt die Trennung von Code und Design. In fast allen Fällen, wo diese Technik verwendet wird, stimmen die Namen von Design- und Code-Datei überein. Die folgenden beiden Listings gehören zusammen:

```
<html>
</html>
```

Listing 2: Eine sinnlose Datei (SinnLos.aspx)


```
Imports SinnLos  
Sub Page_Load ()  
    ' nichts  
End Sub
```

Listing 3: Sinnloser Code dazu (SinnLos.aspx.vb)

Für besonders interessierte Leser: Der Exkurs

Wenn Sie mehr wissen wollen, als für die Arbeit mit ASP.NET unbedingt notwendig ist, etwas über die Geschichte eines Standards, Hintergrundinformationen oder Theoretisches, dann sind die gelegentlich eingestreuten Exkurse hilfreich.

Was ist eigentlich ein Exkurs?

Exkurse gehen über den Kontext des Themas deutlich hinaus. Sie können, wenn Sie es eilig haben, diese Teile problemlos überspringen. Sie sind zum unmittelbaren Verständnis der gezeigten Abläufe nicht zwingend erforderlich.

Wenn Sie Profi werden wollen und ein solides Basiswissen benötigen, helfen Ihnen die Exkurse dabei, einen Blick über den Tellerrand zu werfen.

Hinweise zur Schreibweise und Symbolik

Dieser Abschnitt erklärt die verwendeten Schreibweisen und die Symbole in der Marginalspalte.

Ein Hinweis zur Sprache

Ein direkter und persönlicher Hinweis sei gleich am Anfang gestattet. Im Gegensatz zu vielen anderen Fachbüchern sind in diesem Buch – wann immer es sinnvoll und sprachlich vernünftig erschien – deutsche Begriffe verwendet worden. Sie finden bei der Einführung eines neuen Begriffs jedoch immer die englische Entsprechung. Es ist aber unangebracht, den Text wild aus deutschen und englischen Wörtern zu mischen, wenn eine saubere Übersetzung gelingt. Sicher wird auch hier der Browser nicht zum Blättern

**Deutsche Begriffe
für ein deutsch-
sprachiges
Fachbuch**

rer. Aber in vielen anderen Fällen werden Übersetzungen konsequent genutzt. Wenn Sie den ersten Schock nach reichhaltigem Studium englischer Webseiten und zweifelhafter so genannter »Tutorials« überwunden haben, werden Sie schneller und flüssiger lesen können.

Schreibweise

Im Buch wird zur Hervorhebung spezieller Wörter eine einheitliche Schreibweise verwendet:

- ▶ Schlüsselwörter sind folgendermaßen bezeichnet: *Imports*, *Dim*.
- ▶ Variablen, Klassen, Methoden und andere Namen aus dem Framework oder eigenen Programmen – im weitesten Sinne also Code – wird vergleichbar geschrieben: `System.Math`, `HtmlControl`.
- ▶ Wenn es sich um Bezeichner handelt, die nicht direkt im Code auftauchen, sondern lediglich Namen darstellen, finden Sie die folgende Schreibweise: Tabelle *Adressen*, *x* und *y*.
- ▶ Ebenso sind Dateinamen, Pfade, URLs usw. leicht zu finden: `c:\winnt\system32`.
- ▶ Wenn Sie Befehle in Applikationen ausführen sollen, werden diese meist über Menüs, Dialoge usw. erreicht. Geschrieben werden Sie in Kapitälchen: DATEI | SPEICHERN.
- ▶ Eine besondere Schreibweise weist auf Tasten hin. Diese Schrift wurde auch für Schaltflächen verwendet, die Sie anklicken können: `Enter`, `Strg-S`, `Abbrechen`.

Im Text selbst finden Sie außerdem Abdrucke von Code. Diese sind immer so geschrieben wie einzelne Codes im Text auch:

```
Dies ist kein Code,  
aber er sieht so aus und  
erstreckt sich über mehrere Zeilen
```

Übrigens: Nur das, was im Text als »Listing« bezeichnet wird und mit einer Listingunterschrift versehen ist, finden Sie auf der CD und im Web und können es meist ohne weiteres ausführen. Alle anderen Codes dienen nur der Erläuterung oder Ergänzung und sind nicht allein ausführbar. Beachten Sie aber immer die Hinweise im Text, manche Listing bedürfen der Anpassung an die Ver-

hältnisse auf dem eingesetzten Entwicklungssystem, um lauffähig zu werden.

Symbole

Damit Sie zügig mit dem Buch arbeiten können, werden in aller Kürze die Techniken gezeigt, die Sie vorfinden. Wichtige Elemente sind durch Symbole gekennzeichnet:

Tipp – Dieses Symbol kennzeichnet Passagen, die aus dem Kontext herausragen, Zusatzinformationen liefern oder einfach interessant sind.



Hinweis – Besonderheiten und abweichendes Verhalten in bestimmten Situationen wird mit dem Hinweis-Symbol gekennzeichnet.



Warnung – Hier werden typische Fallen und Fehlerquellen angesprochen oder auf Nebenauswirkungen aufmerksam gemacht.



Über das Buch

Dieses Buch entstand vollständig in XML und wurde dann in einem professionellen Satzstudio für den Druck umgesetzt. Durch diese Technik konnte die auf CD befindliche HTML-Version, die sich nur wenig von der vorliegenden Druckfassung unterscheidet, mit wenig Aufwand realisiert werden.

Unterstützung für Leser

Die Autoren nehmen die Unterstützung des Lesers bei seiner Arbeit mit dem Buch sehr ernst. Aufgrund der unglaublich hohen Anzahl von Anfragen kann jedoch nicht garantiert werden, dass auf jede E-Mail sofort eine ausführliche Antwort folgt. Deshalb nutzen Sie unbedingt die Website zum Buch und die Daten von der CD, die abgedruckte Codes sinnvoll ergänzen.

Die Buch-CD

Die CD zum Buch enthält folgende Daten:

- Das Framework als Redistributable-Package in der Anfang 2003 vertriebenen Version
- Das .NET-Framework Service Pack 1
- Sämtliche Beispiele in Form einer Visual Studio .NET-Projektmappe mit dem Namen »ASPNETVBNET«.



Wenn Sie Visual Studio .NET nicht verwenden, kopieren Sie die einzelnen aspx- und aspx.vb-Dateien von der CD und verwenden Sie diese direkt. Beachten Sie die stellenweise benötigten Spieldaten in diversen Unterverzeichnissen.

Die Projekte umfassen jeweils ein oder mehrere Kapitel. Alle enthaltenen Programme sind exakt so benannt, wie im Buch in den jeweiligen Listing-Unterschriften. Die Zuordnung zwischen den Projektnamen und den Kapiteln des Buches können Sie der folgenden Tabelle entnehmen:

Projektname	Enthält Daten der folgenden Kapitel:
vbnetbasis	Kapitel 3, 4 und 8
vbnetdbase	Kapitel 5 und 9
vbnetwebform	Kapitel 6 und 7 (außer Abschnitt 7.6)
vbnetcontrolproject	Abschnitt 7.6 in Kapitel 7
vbnetusercontrol	
vbnetsecurity	Kapitel 10
vbnetadvanced	Kapitel 11 bis 13

Tabelle 1: Namen der Projekte und der Kapitel, aus denen sie Programme enthalten

Installation

Bevor Sie die Programme von der CD installieren, müssen Sie über ein lauffähiges Framework und ASP.NET verfügen. Visual Studio .NET ist in vielen Fällen empfehlenswert, aber nicht zwingend erforderlich. Die Programme zu diesem Buch liegen in Form eines Installationsprogramms vor. Um sie verwenden zu können, gehen Sie folgendermaßen vor:

1. Öffnen Sie den Ordner *Listings* auf der CD.
2. Starten Sie das Installationsprogramm mit *Setup.exe*.
3. Folgen Sie den Anweisungen.

Das Installationsprogramm erzeugt nun die zum Ablauf der Programme notwendigen virtuellen Verzeichnisse im Webserver, kopiert die Quellcodes und Assemblies und die zum Start erforderliche HTML-Datei. Der Standardname des Stammverzeichnisses ist *AspnetVBNet*. Die darunter liegenden Verzeichnisse enthalten jeweils ein Projekt. Eine Übersicht finden Sie in Tabelle 1. Nach der Installation starten Sie mit folgender Adresse:

<http://localhost/AspnetVBNet/index.htm>

Wenn Sie Ihren Webserver nicht lokal betreiben, ersetzen Sie »localhost« durch den entsprechenden Namen oder die IP-Adresse. Wichtig ist, dass Sie die URL verwenden und nicht den Aufruf über das Dateisystem. Der Steuerung der Listen der Programme erfolgt über ein kleines ASP.NET-Programm, das nur ausgeführt wird, wenn der IIS die Seite verarbeitet.

Nach der Installation können Sie aus den gelieferten Dateien die Projekte in Visual Studio .NET rekonstruieren. Der hier beschriebene Weg sichert ab, dass die Programme alle mit lokalen Pfaden angesprochen werden, so als ob Sie diese selbst erstellt hätten. Gehen Sie dazu folgendermaßen vor:

Projekte in Visual Studio .NET wiederherstellen

1. Legen Sie eine neue Projektmappe an. Speichern Sie diese in einem lokalen Ordner.
2. Fügen Sie der Projektmappe nun Projekte hinzu. Verwenden Sie dazu die Option PROJEKT VOM WEB ÖFFNEN. Suchen Sie die Projekte in dem Ordner, den Sie bei der Installation angegeben haben.
3. Stellen Sie die Konfiguration Ihrer Projekte auf DEBUG ein.
4. Erstellen Sie die gesamte Projektmappe erneut – rechte Maustaste auf die Projektmappe zum Aufruf des Kontextmenüs und dann PROJEKTMAPPE ERSTELLEN wählen.
5. Jetzt können Sie einzelne Programme mit F5 oder Strg-F5 starten. Beachten Sie dabei, dass diese Tasten – mit oder ohne Debugger – immer das so genannte Startprojekt und darin die Startdatei starten. Beides lässt sich über die Kontextmenüs der Projekte bzw. Dateien einrichten. Alternativ können Sie die Programme weiter über die Startdatei der Applikation, *index.htm*, erreichen. Vergessen Sie nicht, die Programme nach Änderungen neu zu erstellen.

Probleme? Informieren Sie die Autoren bitte, wenn Sie glauben Programmfehler gefunden zu haben oder es Schwierigkeiten mit der Installation gibt. Auf der Webseite zum Buch finden Sie einige Zeit nach Erscheinen des Buches Updates der Installationsprogramme.

Die Website zum Buch

Die Website zum Buch erreichen Sie über folgende Portalseite: <http://www.dotnet.comzept.de>. Klicken Sie dort auf das Cover des von Ihnen erworbenen Buches und Sie finden Informationen, verbesserte und aktualisierte Programme und Kontakt zu anderen Lesern.

Sie finden dort – frühestens vier Wochen nach Erscheinen des Buches – eine aktualisierte Fassung der gesamten Projektmappe vor.

Hostingservice

Wenn Sie einen Provider suchen, der Ihre Applikation aufnimmt, ohne dass Sie gleich einen eigenen Server aufsetzen, können wir Ihnen die Firma QualityHosting empfehlen. Mehr Informationen dazu finden Sie im Web unter <http://www.qualityhosting.de>.

QualityHosting lässt seine Server vom externen, unabhängigen Anbieter »1stWarning.com« überwachen. Bei Überwachung aller Server ergaben sich Messwerte im Bereich von 99,89% bis 99,98% Uptime. Im unabhängigen Geschwindigkeitstest der Plattform *WebHostlist.de* wurden die Server wiederholt mit der Note »Sehr Gut« ausgezeichnet.

50 Mb Webspace gibt es bereits ab € 10,90 pro Monat (unverbindliche Angabe von Ende 2002).

A Vorbereitung auf ASP.NET

I Prinzip und Funktionsweise

Dieses Kapitel führt Sie schnell und direkt in die Programmierung mit ASP.NET ein. Es ist vor allem ein Schnellstart, der Anfängern und Umsteigern einen Überblick über Möglichkeiten und Hintergründe gibt.

I.1 Unterschiede zum alten ASP

Wenn Sie bereits mit ASP 3.0 programmiert haben, hilft dieser Abschnitt, Unterschiede und Ähnlichkeiten schnell zu erfassen. Haben Sie mit ASP.NET das erste Mal mit der Webserverprogrammierung zu tun, können Sie diesen Abschnitt überspringen. Fahren Sie dann mit Abschnitt 1.5, »Programmierprinzipien« ab Seite 77 fort.

I.1.1 Grundlegende Unterschiede

Die Unterschiede zum Vorläufer ASP sind durchaus beachtlich, denn ASP.NET ist keine weitere Version, sondern startet mit einem völlig neuen Konzept wieder bei Version 1.0.

Die Programmiersprachen

Erste Unterschiede werden bereits mit der Wahl der Programmiersprache offensichtlich. ASP wird überwiegend mit VBScript programmiert, alternativ wird auch JScript verwendet. Das offene Konzept des Scripting Host erlaubt dabei auch andere Sprachen, wie beispielsweise Perl und Python. Diese konnten sich in der ASP-Welt bislang kaum durchsetzen, weil die eigenen Bibliotheken zur Webserverprogrammierung umfangreicher sind als die mit ASP gelieferten. ASP-Programme können von der Erweiter-

barkeit über COM bzw. COM+ profitieren – wenn Sie denn in der Lage sind, mit C++ oder Visual Basic nativ zu programmieren.

Fünf Programmiersprachen verfügbar

Mit .NET ist diese Unterscheidung nicht mehr gegeben. Sie erstellen Programme in VB.NET, C#, JScript.NET, J# oder C++.NET und nicht mehr in einer Skriptsprache. C++.NET rückt dabei etwas in den Hintergrund, denn die Übersetzung in die IL (Intermediate Language) erlaubt nur noch wenige sprachtypische Vorteile, sodass es nur selten lohnt, aus Laufzeitgründen die Sprache zu wechseln. Verfügbar ist auch ein Java-Dialekt, aus lizenzrechtlichen Gründen J# genannt. Wer vorher Java programmiert hat, wird sich hier zu Hause fühlen. Allerdings erstellt der J#-Compiler keinen Code, der von einer Java Virtual Machine ausgeführt werden kann, sondern wie alle .NET-Sprachen IL-Code.

ASP.NET-Programmierer haben also die Wahl, in fünf standardmäßig verfügbaren Sprachen zu programmieren. Weitere Sprachen können von anderen Anbietern geliefert werden. Allerdings ist zu beobachten, dass die extra für .NET entworfene Sprache C# sich sehr schnell großer Beliebtheit erfreut hat. Ebenso hat sich die große Gemeinde der VB-Programmierer zügig mit VB.NET angefreundet. Den Schwerpunkt stellen also eindeutig die Sprachen C# und VB.NET dar. Aus Sicht des VBScript-Programmierers mag VB.NET leichter erscheinen. Allerdings ist der Lernaufwand insgesamt erheblich, sodass die Wahl der Sprache eher nebensächlich ist.

Warum C#?

Ein paar Tipps aus der praktischen Erfahrung mit allen Sprachen: C# ist eleganter und flüssiger zu schreiben. Wenn Sie schon immer C lernen wollten, haben Sie nun die Gelegenheit, sich in die Denkweise und Syntax einzuarbeiten. Der Sprung nach C++ ist dann nicht mehr ganz so groß (sollte aber trotzdem nicht unterschätzt werden). Wenn Sie bereits in Visual Basic 6 oder VBScript erfolgreich programmiert haben, ist VB.NET die erste Wahl. Wenn Sie ASP 3.0 mit JScript programmiert haben, ist JScript.NET die beste Wahl. Allerdings ist der Umstieg auf C# wegen der ähnlichen Syntax unproblematisch. JScript fehlt überdies die Unterstützung durch Visual Studio .NET, Sie müssten immer mit dem Kommandozeilecompiler arbeiten, was auf Dauer nicht wirklich Spaß macht.

In diesem Buch wird VB.NET verwendet. Wenn Sie sich für C# interessieren, finden Sie eine Version dieses Buches mit C# unter der ISBN 3-8273-1974-9.

Abarbeitung

Ein wesentlicher technologischer Unterschied zwischen ASP und ASP.NET ist die Art der internen Code-Verarbeitung. Während ASP ein reiner Interpreter ist, kommt mit ASP.NET-Programmen immer der Compiler der jeweiligen Sprache zum Einsatz. Glücklicherweise ist ASP.NET so programmiert, dass die Übersetzung im Hintergrund stattfindet. Erst durch eine Fehlermeldung bei falschem Code werden Sie mit den Ausgaben des Compilers konfrontiert. Die Übersetzung findet in die schon erwähnte Zwischensprache IL statt, erst bei der Ausführung konvertiert ein Just-In-Time-Compiler (JIT-Compiler) diesen Code in die Maschinsprache des Prozessors. Diesen Vorgang werden Sie nie bemerken – außer durch den erheblichen Gewinn an Leistung. ASP.NET ist vor allem unglaublich viel schneller als ASP.

**Interpreter vs.
Compiler**

Dateierweiterungen

Wie bei ASP wird eine Webseite über den IIS ausgeliefert und die Verarbeitung erfolgt aufgrund einer Verknüpfung zwischen einer bestimmten Dateierweiterung und der ASP.NET-Komponente. Aus *.asp* ist dabei, unter anderem, *.aspx* geworden. »Unter anderem« deshalb, weil es noch andere Dateierweiterungen gibt, die bestimmte Sonderdienste in ASP.NET ansprechen.

.aspx anstelle .asp

Modularisierung

ASP kennt selbst keine Modularisierungskonzepte. Lediglich die SSI (Server Side Includes), die über den IIS verfügbar sind, können eingesetzt werden, um Dateien mit Codeteilen wiederzuverwenden. SSI sind mit ASP.NET übrigens nicht verschwunden, solche Zeilen wie `<!-- #include file="name.htm" -->` funktionieren nach wie vor. Der Unterschied besteht darin, dass Sie eine Fülle von eleganten und leistungsfähigeren Modularisierungsfunktionen haben, die SSI schnell in Vergessenheit geraten lassen.

So können Sie HTML-Fragmente als so genannte Benutzer-Steuer-elemente definieren und so verwenden, als wären es eingebaute Steuer-elemente. Damit stehen die Stückchen Code nicht nur als eigenständige Dateien bereit, sondern auch als Objekt im Code der Seite. Die Trennung von Design und Code bleibt erhalten.

**Benutzer-Steuer-
elemente**

Kunden-Steuer- elemente	Noch weiter gehen die Kunden-Steuerelemente. Hier wird eine eigene Designvorschrift entworfen und Code dazu geschrieben. Dieser wird dann einzeln in eine Assembly übersetzt. Das Verfahren ist nicht nur erstaunlich einfach, sondern auch sehr leistungsfähig. Sie können Code nämlich so gut schützen und geben nur die DLL weiter, die erzeugt wurde.
Eigene Namens- räume	Nicht zuletzt können Sie natürlich auch eigene Namensräume definieren, die enthaltenen Klassen übersetzen und auch so geschützten Code weitergeben. Um die Modularisierung auf die Spitze zu treiben, steht jedem Modul die Wahl der Sprache frei. Dank der IL schreiben Sie Ihre Module vielleicht in VB.NET, Ihr Kollege in C# und der spätere Anwender bleibt bei JScript.NET auf der Seite.

1.1.2 Vorteile für den ASP.NET-Entwickler

ASP.NET ist gegenüber ASP keine Evolution, sondern ein völlig neues Produkt. Fast alles, was an ASP kritisiert wird, ist in ASP.NET nicht mehr zu finden.

Script Limits	ASP hat weit reichende Einschränkungen, die durch die Verwendung einer reinen Skriptsprache bedingt sind. Entwickler müssen hier oft auf Komponenten ausweichen, die in VB oder C++ geschrieben wurden – in richtigen Programmiersprachen also. Das ist nun nicht mehr notwendig. VB.NET ist eine »richtige« Programmiersprache und Sie müssen auf keine Funktion verzichten. Alles, was vorher in Komponenten geliefert wurde, kann nun direkt programmiert werden. Selbstverständlich lassen sich fertige Programme kompilieren und anderen zur Verfügung stellen. Damit einher geht natürlich auch ein Verlust der Einfachheit – statt typloser Programmierung muss das sehr strenge Typsystem CTS verwendet werden (siehe Abschnitt »Einführung in das Common Type System« ab Seite 154).
Verteilung	Die Installation fertiger ASP-Anwendungen ist ein durchaus komplizierter Vorgang, wenn es sich um ASP-Code handelt, der COM-Komponenten enthält. ASP.NET-Anwendungen lassen sich sehr leicht auf einem Entwicklungssystem erstellen und dann auf einen Produktionsserver bringen. Es genügt, die fertigen Programme zu kopieren. Weder Struktur noch Dateien müssen geändert werden. Legen Sie auf dem Produktionsserver ein virtuelles Verzeichnis an, kopieren Sie alle Daten dort hinein – fertig. Es ist

nicht notwendig, in die Metabasis des IIS einzugreifen oder Konfigurationen vorzunehmen. Alles, was konfiguriert werden muss, passiert in gut lesbaren XML-Dateien, die in den jeweils zu konfigurierenden Verzeichnissen liegen. ASP.NET liest diese Dateien und greift während der Laufzeit nur auf den Speicher zu. Es registriert aber sofort, wenn sich Änderungen an Code-Dateien ergeben, sodass Sie weder den Server noch den Webserver oder einen anderen Dienst neu starten müssen.

1.1.3 Eine neue Philosophie

ASP.NET ändert die Art und Weise, wie Webseiten programmiert werden, völlig. Dabei ist dies nicht nur im Vergleich zu ASP revolutionär, sondern auch gegenüber anderen Systemen, besonders den klassischen Skriptsprachen wie PHP und Perl.

Die Integration von HTML-Code auf der einen Seite und Skriptsprache auf der anderen ist bei großen Projekten ein massives Hemmnis für eine professionelle Softwareentwicklung. Der Code zerfasert durch die Interaktion von HTML – was notwendigerweise erzeugt werden muss – und den Programmmodulen, die elementare Aufgaben wie die Abfrage von Datenbanken erledigen. Im Ergebnis sind solche Seiten mit herkömmlichen HTML-Editoren nicht mehr zu bearbeiten und der Code selbst ist oft derart unleserlich, dass eine Wiederverwendung unmöglich erscheint. Im Endeffekt sollte diese Art Programmierung nur für sehr kleine Aufgaben genutzt werden; die Entwicklung wird sonst früher oder später sehr teuer. In viele Fällen wird sich mit Templatesystemen beholfen. Das sind Programme, die reine HTML-Vorlagen mit eingebetteten Variablen verwenden und den Code konsequent auslagern. Nachteilig ist die mangelnde Leistung, weil sie bislang immer in der Skriptsprache programmiert wurden, für die der Umgang erleichtert werden soll.

Klassisch: HTML und Code gemischt

Die wichtigste Erkenntnis beim Umgang mit ASP.NET ist: »ASP.NET ist ein Templatesystem«. Vielleicht haben Sie erwartet, dass der Compiler, C# oder das Framework die wichtigsten Neuerungen sind. Das sind sicher alles bedeutende Bestandteile der Programmierungsumgebung. Sie helfen aber nicht, ASP.NET so zu verstehen, dass es optimal verwendet werden kann. Die Idee hinter ASP.NET ist die Trennung von Code und Design. Der Nachteil von Templatesystemen, die mangelnde Leistung, tritt hier nicht

ASP.NET ist ein Templatesystem

auf. Die genannten Komponenten, im Mittelpunkt der Compiler, sorgen für die nötige Leistung.

ASP.NET-Seiten- verarbeitung

Damit das funktioniert, wird mit ASP.NET ein komplexer und sehr strenger Seitenverarbeitungszyklus eingeführt. Bevor die eigentliche Verarbeitung stattfindet, analysiert ASP.NET die Vorlage und erstellt aus allen entsprechend gekennzeichneten Elementen Objekte. Diese Objekte geben dem Programmierer einen Zugriff auf Inhalt, Attribute und verbundene Daten im eigentlichen Programm. Da in der Welt von .NET alles Objekte sind, steht auch die Seite selbst als solches bereit. Bestimmte Zustände dieser Objekte können Ereignisse auslösen. ASP.NET ermöglicht bei der Programmierung von Webserverapplikationen einen Stil, der dem der Windows-Programmierung sehr nahe kommt. Aber nicht nur das – alle Klassen des Frameworks, auch die grafischen Module zum Erstellen grafischer Oberflächen – stehen zur Verfügung. In Abschnitt 11.2, »Bilder dynamisch erstellen« ab Seite 990 finden Sie eine praktische Anwendung bei der dynamischen Erstellung von Bildern. Die Ereignissteuerung ist raffiniert gekapselt. Wenn Sie eine Schaltfläche mit einer Methode so verbinden, dass die Methode ausgeführt wird, wenn die Schaltfläche angeklickt wurde, ist dies Ereignissteuerung. Das natürlich ein Formular abgesendet und die per POST gesendeten Daten ausgewertet werden, bleibt weitgehend verborgen.

Vermeiden Sie den klassischen Weg unter ASP.NET!

Bei den ersten Schritten mag es einfacher erscheinen, klassisch zu programmieren. Denn es stehen die von ASP bekannten Klassen Request, Response usw. auch in ASP.NET zur Verfügung. Und nicht nur das, sie sind auch kräftig gewachsen und haben viele neue und leistungsfähige Methoden hinzubekommen. Nichtsdestotrotz ist der Einsatz relativ selten nötig, denn die neuartige Formularverarbeitung ist sehr leistungsfähig und spart viel Code, der sonst mühevoll programmiert werden müsste. Das gilt auch für clientseitigen JavaScript-Code. Hier liefert ASP.NET einige sehr gute Bibliotheken mit, um Reaktionen zu realisieren, die mit HTML alleine nicht möglich sind. Diese Bibliotheken laufen übrigens auch mit anderen Browsern als dem Internet Explorer. Tatsächlich spielen sich alle .NET-spezifischen Dinge auf dem Server ab. Im Browser kommt nichts als HTML, CSS und eventuell JavaScript an. Allerdings setzt Microsoft voraus, dass HTML 4- und CSS 2-konforme Endgeräte verfügbar sind.

I.2 Vorbereitung auf ASP.NET

Vor den ersten Schritten mit ASP.NET sind ein paar Vorbereitungen notwendig. Dies betrifft einerseits einige Grundlagen, andererseits auch die technische Basis. Beides wird in den folgenden Kapiteln des Buches systematisch vertieft und ausgebaut. Dieser Abschnitt hat die Aufgabe, Ihnen den ersten Schrecken zu nehmen und zu einem schnellen Erfolgserlebnis zu verhelfen.

I.2.1 Überblick über die System-Voraussetzungen

ASP.NET ist keine eigene Programmiersprache. Es ist eine Erweiterung des Webserver Internet Information Server (IIS) 5 bzw. IIS 5.1 um die Fähigkeit, mit Webseiten verknüpften Code auszuführen und das Ergebnis der Berechnungen an den Browser zu senden. Wie das erfolgt, wird im nächsten Abschnitt noch genauer gezeigt. Der eigentliche Verarbeitungsprozess passiert also in Form von Programmen, die in der Website oder damit verknüpft vorliegen und ausgeführt werden, wenn ein Benutzer eine Seite über das Internet anfordert.

An erster Stelle der Voraussetzungen zur Entwicklung von Websites steht der Webserver. Derzeit unterstützt nur der Internet Information Server (ab Version 5) von Microsoft ASP.NET. Damit kommen nur die folgenden Systeme in Frage:

Webserver

- ▶ Windows XP Professional
- ▶ Windows 2000 Professional (ab Service Pack 2)
- ▶ Windows .NET Server-Familie
- ▶ Windows 2000 Server-Familie (ab Service Pack 2)

Ein Serversystem direkt als Entwicklungsplattform einzusetzen, ist in der Praxis sicherlich eher unüblich. Denkbar ist natürlich auch der Fall, dass Sie auf einer Arbeitsstation entwickeln und die fertigen Seiten beziehungsweise Code-Dateien auf den Webserver laden. Für ein professionelles Entwickeln ist allerdings die Konfiguration Visual Studio mit einem lokal installiertem IIS unbedingt zu empfehlen.

Sie können auch ohne den IIS entwickeln. Die freie Entwicklungsumgebung Web Matrix bringt einen eigenen Webserver mit. Damit lässt sich eine Entwicklungsumgebung beispielsweise unter Windows XP Home aufbauen, obwohl diese Version keinen IIS mitbringt.



- .NET Framework** Das .NET Framework selbst bietet Microsoft kostenlos an. Im nachfolgenden Abschnitt finden Sie weiterführende Informationen zur Beschaffung und Installation. Wenn Sie das Visual Studio einsetzen, brauchen Sie sich übrigens darum weniger Gedanken machen – hier ist das Framework neben anderen Tools bereits im Lieferumfang enthalten.
- MDAC** Wenn Sie Applikationen entwickeln wollen, die auf Datenbanken zugreifen, werden Sie die Microsoft Data Access Components (MDAC) benötigen. Diese sind wie das Framework frei erhältlich und momentan in der Version 2.7 aktuell. Auf den SQL-Server (Version 7 oder 2000) können Sie direkt zugreifen und benötigen MDAC hierfür nicht.
- Editor** Zum Eintippen der Codes brauchen Sie einen Editor. Im Grunde reicht bereits der bei Windows standardmäßig enthaltene Notepad aus. Allerdings werden Sie damit schwerlich produktiv arbeiten oder gar größere Projekte bewältigen können. Deutlich besser geeignet sind professionelle Entwicklungswerkzeuge. Das wichtigste Werkzeug für die .NET-Programmierung, welches allerdings einiges kostet, ist derzeit das Visual Studio .NET. Dieses wird in Kapitel 3 näher behandelt.

1.2.2 Grundlegende Komponenten eines Entwicklungssystems

Reihenfolge der Installation beachten In diesem Abschnitt geht es darum, wie Sie ein Entwicklungssystem für ASP.NET einrichten. Dabei ist die Reihenfolge der Installationsschritte unter Umständen entscheidend. Für die Einrichtung eines Windows 2000- und Windows XP-Professional-Systems gehen Sie wie folgt vor:

1. Installation des Internet Information Servers
2. Installation des .NET Frameworks

Beide Installationsschritte werden in den nachfolgenden Abschnitten näher behandelt. Nach diesen beiden Schritten können Sie mit einem Texteditor bereits mit dem Entwickeln loslegen. Gegebenenfalls müssen Sie für die Datenbankprogrammierung noch das MDAC installieren.

Ausnahme: Visual Studio .NET Haben Sie vor, das Visual Studio .NET zu installieren, brauchen Sie sich um diese vorbereitenden Schritte keine Gedanken zu machen. Das Setup-Programm des Visual Studios.NET wird Sie auf even-

tuell fehlende Komponenten hinweisen. Das .NET Framework bringt dieses ohnehin mit.

Eine weitere Ausnahme bildet *Web Matrix*. Dieses Werkzeug bringt wie bereits erwähnt seinen eigenen Webserver mit. Nach der Installation des .NET Frameworks können Sie mit der Installation von Web Matrix direkt fortfahren. Mehr zu WebMatrix finden Sie unter <http://www.asp.net>.

Ausnahme:
Web Matrix

IIS installieren

Zur Installation des IIS 5 auf einem der unterstützten Windows-Betriebssysteme gehen Sie wie nachfolgend beschrieben vor. Die einzelnen Schritte sind bei den Windows Client-Systemen (2000 und XP Professional) sowie den Serversystemen (2000 und .NET Server) prinzipiell gleich. Gezeigt wird hier das Vorgehen bei einer Windows XP Professional Arbeitsstation.

Öffnen Sie zuerst über das Startmenü die Systemsteuerung und starten Sie dort das Programm SOFTWARE. Klicken Sie im dann folgenden Dialogfenster auf WINDOWS-KOMPONENTEN HINZUFÜGEN/ENTFERNEN. Es startet der ASSISTENT FÜR WINDOWS-KOMPONENTEN. In der Liste installierter Software suchen Sie den Eintrag INTERNET-INFORMATIONSDIENSTE (IIS). Wenn das Kontrollkästchen auf der linken Seite aktiviert ist, wurde der IIS bereits installiert.

Installations-
schritte

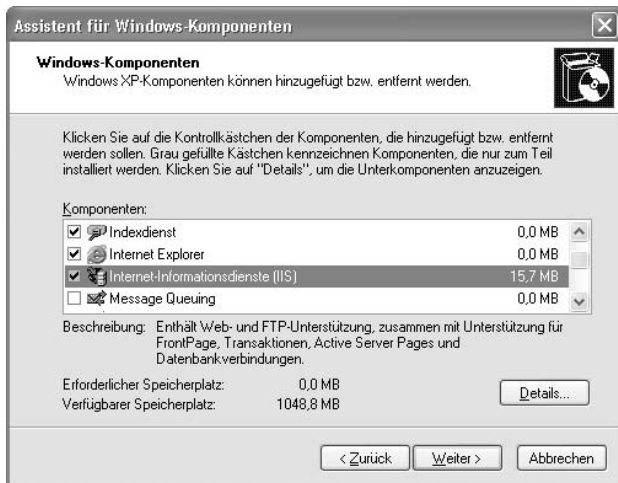


Abbildung 1.1: Installation des IIS unter Windows XP Professional

Für den IIS können einige Installations-Optionen bestimmt werden, die Sie über die Schaltfläche DETAILS erreichen. Für den Einsatz mit ASP.NET empfiehlt es sich, alle Optionen zu aktivieren. Ausgenommen sind davon die Optionen FTP-DIENST und SMTP-DIENST. Diese Komponenten sollten Sie nur dann installieren, wenn Sie entsprechende Serverfunktionen für diese Protokolle auf Ihrem System benötigen.

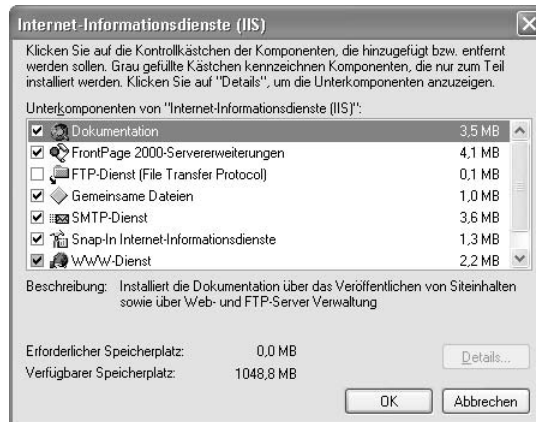


Abbildung 1.2: Auswahl der Installationsoptionen

Mit einem Klick auf OK wird der Installationsprozess gestartet. Möglicherweise werden Sie aufgefordert, die Installations-CD einzulegen. Der IIS startet – als Dienst WWW-PUBLISHINGDIENST – sofort nach der Installation.



Auch wenn Sie das System nach der Installation nicht zu einem Neustart auffordert: Starten Sie dennoch Ihr System neu, bevor Sie mit weiteren Installationsschritten fortfahren. Einige Komponenten des IIS arbeiten sonst unter Umständen noch nicht oder nicht richtig.

Wie der IIS arbeitet

Für eine erste Orientierung sollten Sie sich das IIS-Arbeitsverzeichnis anschauen. Der IIS legt ein Verzeichnis `\inetpub` an, das folgende wichtige Unterverzeichnisse enthält:

► `\iissamples`

Hier sind einige Beispiele im alten ASP 3.0 zu finden.

► `\scripts`

Dieses Verzeichnis wurde zum Ablegen von ASP-Programmen genutzt. Es spielt unter ASP.NET keine Rolle.

► `\wwwroot`

Das ist das Stammverzeichnis des Webservers; hier liegen die Dateien, die ohne weitere Pfadangabe erreicht werden können.

► `\webpub`

Dieses Verzeichnis dient der Ablage von Dateien, die per WebDAV hochgeladen wurden. WebDAV ist eine HTTP-Erweiterung, die den Zugriff auf das Dateisystem eines Webservers per HTTP erlaubt. Das Verzeichnis ist nach der Installation leer.

► `\ftproot` und `\mailroot`

Dies sind die Stammverzeichnisse des FTP- und SMTP-Servers. Wenn Sie die Installation auf einem Windows 2000- oder .NET Serversystem ausgeführt haben, finden Sie außerdem noch `\nnftproot`, das Stammverzeichnis des Newsservers.

Der IIS enthält einige Dateien, die für einen ersten Test benutzt werden können. Starten Sie dazu den Browser und geben dann folgende Adresse ein: `http://localhost`.

IIS testen

Sie sollten dann das folgende Bild sehen. Sie können nun mit der Installation des Frameworks fortsetzen.

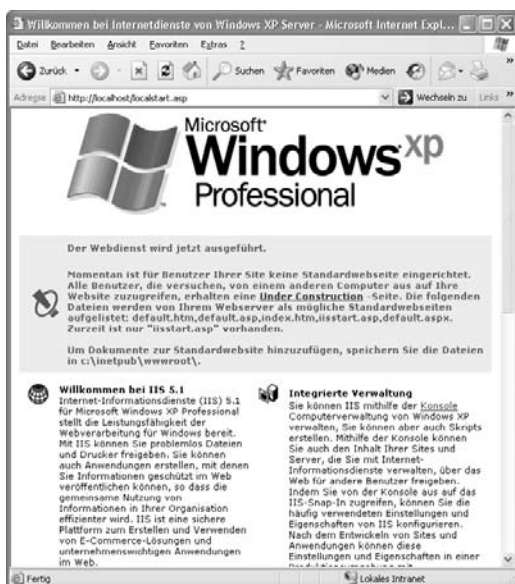


Abbildung 1.3: Der IIS wurde erfolgreich installiert.

Was tun bei Fehlern?

Mit dem funktionierenden IIS können Sie die weiteren Schritte ausführen. Zeigt der Browser einen Fehler an, sollten Sie erst fortfahren, wenn die Ursache gefunden ist. Gehen Sie dazu folgendermaßen vor:

- ▶ Prüfen Sie den Dienst WWW-PUBLISHINGDIENST. Er muss gestartet sein, damit der IIS arbeiten kann.
- ▶ Wenn das nicht die Ursache sein sollte, probieren Sie, statt »localhost« den Namen des Computers anzugeben.
- ▶ Möglicherweise liegt ein Konfigurationsfehler in der Netzwerkkonfiguration vor. Der Rechner sollte neben seiner IP-Adresse auch über 127.0.0.1 erreichbar sein.

.NET Framework installieren

Das .NET Framework wird momentan in zwei Varianten von Microsoft angeboten:

▶ *.NET Framework Redistributable*

Diese Variante ist ca. 21 MByte groß und beinhaltet nur die notwendigsten Bestandteile. Das sind die gesamten Laufzeitprogramme des Frameworks sowie ASP.NET. Diese Version wird üblicherweise auf Produktionsservern (Windows 2000 Server oder Windows .NET Server) eingesetzt.

▶ *.NET Framework SDK*

Neben den Bestandteilen der kleinen Fassung finden Sie in diesem Paket umfangreiche Dokumentationen, Kommandozeilen-Tools, Compiler, Debugger, MSDE sowie einige Beispiele. Diese Variante sollten Sie sich beschaffen, wenn Sie professionell entwickeln wollen und das Visual Studio .NET *nicht* besitzen. Hier ist das Framework SDK bereits enthalten.

SDK: auf Buch-CD

Das .NET Framework SDK finden Sie auf der dem Buch beiliegenden CD. So können Sie sich den Download sparen, der bei einer Paketgröße von ca. 135 MB doch einige Zeit benötigen kann.

Download-Adresse

Sie erhalten das .NET Framework in beiden Varianten auch kostenlos auf der folgenden Website von Microsoft: <http://www.asp.NET/download.aspx>. Hier finden Sie auch weiterführende Informationen und gegebenenfalls Download-Links für Service Packs. Momentan (Anfang 2003) ist bereits das Service Pack 2 für das

.NET Framework erhältlich und eine Beta-Version 1.1 des Frameworks verfügbar.

Für die ersten Schritte mit ASP.NET reicht das kleine Paket aus. Die Übersetzung der Programme läuft bei ASP.NET im Hintergrund beim Aufruf der Seite ab – zusätzliche Compiler werden nicht unbedingt benötigt. Wenn Sie auf die Online-Hilfe zugreifen möchten, besuchen Sie die Website <http://www.asp.NET>.

Das Framework wird mit einem Installationsprogramm geliefert. Zur Installation des ».NET-Framework SDK« starten Sie das Programm *Setup.exe*. Die weiteren Installationsschritte erklären sich von selbst. Während des Installationsvorgangs wird der WWW-PUBLISHINGDIENST vorübergehend angehalten. Nach Beendigung der Installation steht ASP.NET bereits zur Verfügung.

Nach der Installation des Frameworks empfiehlt es sich, die Einstellungen im IIS zu überprüfen. Ob die Verbindung von Framework und IIS erfolgreich hergestellt werden konnte, erkennen Sie im Dialogfenster ANWENDUNGSKONFIGURATION. Starten Sie dazu die MMC Internet Informationsdienste und erweitern Sie dort den Zweig WEBSITES. Öffnen Sie das Eigenschaften-Dialogfenster zur STANDARDWEBSITE und klicken Sie auf die Registerkarte BASISVERZEICHNIS. Über die Schaltfläche KONFIGURATION erhalten Sie das angesprochene Dialogfenster.

IIS-Einstellungen prüfen

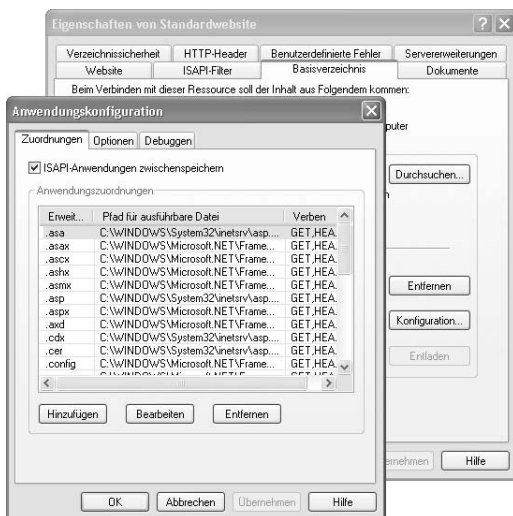


Abbildung 1.4: Prüfen der Anwendungskonfiguration im IIS

Neben den für den IIS üblichen Dateierweiterungen und den entsprechenden Zuordnungen müssen nun auch die neuen .NET-Erweiterungen wie beispielsweise .ASPX erscheinen. Ist das nicht der Fall, ist die Ursache meist eine falsche Installationsreihenfolge. Sie haben dann wahrscheinlich den IIS erst nach dem Framework installiert. Dies kann dann passieren, wenn Sie beispielsweise mit Web Matrix beginnen zu entwickeln (das den IIS nicht braucht) und sich erst später für ein anderes Werkzeug wie etwa Visual Studio .NET entscheiden.

Framework-Installation reparieren

Framework im IIS registrieren

Stellen Sie fest, dass die Zuordnungen im IIS nicht existieren, können Sie das Framework mit dem Aufruf des Dienstprogramms *aspnet_regiis.exe* neu im IIS registrieren:

```
aspnet_regiis.exe /r
```

Dieses Tool finden Sie in folgendem Verzeichnis:

```
%Systemroot%\Microsoft.NET\Framework\<version>
```

Der Platzhalter *<version>* steht für das konkrete Unterverzeichnis, welches durch die jeweilige Versionsnummer des Frameworks gekennzeichnet ist, beispielsweise *v1.0.3705*.

Die folgende Tabelle enthält wichtige Kommandozeilen-Parameter von *aspnet_regiis.exe*:

Parameter	Beschreibung
r	Reinstalliert die momentane ASP.NET-Version in alle Anwendungszuordnungen der IIS-Metabasis
c	Reinstalliert die clientseitigen JavaScript-Bibliotheken für Webserver-Steuerelemente für alle Websites
e	Deinstalliert die clientseitigen JavaScript-Bibliotheken für Webserver-Steuerelemente für alle Websites

Tabelle 1.1: Wichtige Parameter des Programms *aspnet_regiis.exe*

Framework-Reparaturmodus

Sind schwerwiegender Probleme zu verzeichnen, hilft eventuell die Neuinstallation des Frameworks. Ein Doppelklick auf *dotnetfx.exe* hilft da leider nicht weiter, da sich das Framework-Setup weigert, sich noch einmal zu installieren. Es gibt allerdings einen Reparaturmodus, den Sie mit dem folgenden Aufruf des Programms starten:

```
dotnetfx.exe /t:c:\temp /c:"msiexec.exe /fvecms c:\temp\netfx.msi"
```

Geben Sie hinter dem Schalter `/t` ein temporäres Verzeichnis Ihrer Wahl an, in den das Paket entpackt werden soll. Dieses Verzeichnis müssen Sie dann für die Angabe des Speicherortes von *netfx.msi* ebenfalls verwenden. Prüfen Sie nach erfolgter Installation wiederum die Einstellungen im IIS.

Komponenten für den Datenbankzugriff

Für den Zugriff auf Datenbanken müssen Sie unter Umständen noch zusätzliche Komponenten installieren. Dies ist allerdings abhängig von der Windows-Betriebssystem-Version und dem Ziel-Datenbanksystem. Die zentrale Komponente für die Datenbankprogrammierung unter .NET ist ADO.NET. ADO ist die Abkürzung für *ActiveX Data Objects* und für die Datenbankprogrammierung unter Windows die vor .NET hauptsächlich genutzte Schnittstellen-Technologie. ADO.NET löst ADO ab. Nachfolgende Abbildung soll die Zusammenhänge dabei verdeutlichen:

ADO.NET

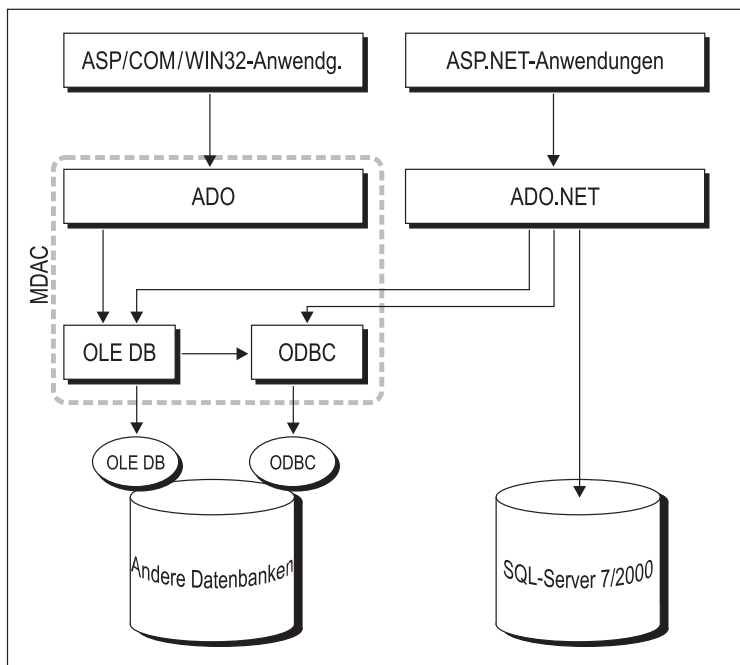


Abbildung 1.5: ADO und ADO.NET

Direkter Zugriff auf MS SQL-Server	Für den Zugriff auf den SQL-Server 7 oder 2000 aus ASP.NET-Anwendungen brauchen Sie auf Ihrem System keine weiteren Komponenten installieren. Lediglich das SQL-Serversystem selbst muss lokal oder gegebenenfalls auf dem externen Entwicklungs-Webserver installiert sein. Der Zugriff erfolgt über den so genannten <i>SQL Data Provider</i> , einer integralen Komponente von ADO.NET.
ODBC	Benutzen Sie andere Datenbanksysteme, brauchen Sie zusätzliche Schnittstellen. Eine international standardisierte Schnittstelle ist dafür ODBC (<i>Open Database Connectivity</i>). Über diese können unter Windows verschiedene relationale Datenbanken angesprochen werden. Voraussetzung ist dabei, dass auf die betreffende Datenbank über eine standardisierte SQL-Syntax zugegriffen werden kann.
OleDb	Die Schnittstelle OleDb wurde von Microsoft entwickelt, um leistungsfähige Applikationen für verteilte Daten erstellen zu können. Im Gegensatz zu ODBC spielt dabei der Typ der Datenbank (wie relational oder hierarchisch) oder die zu verwendende Abfragesprachsyntax keine Rolle. Wichtig ist nur, dass zum Datenaustausch mit der Anwendung eine Tabellenform verwendet werden kann. ADO, ODBC und OleDb sind Bestandteil des <i>Microsoft Data Access Components</i> (MDAC). In diesem Paket sind des Weiteren noch die <i>Remote Data Services</i> (RDS), die allerdings kaum noch eine Rolle spielen, sowie diverse ODBC-Treiber und OleDb-Datenanbieter. Aktuell ist heute (Mitte 2002) die Version 2.7. Sie erhalten MDAC kostenlos über diese Website: http://www.microsoft.com/data/download.htm . Das Paket ist ca. 5 MByte groß und sollte in der deutschen Version geladen werden, wenn Sie ein deutsches Windows verwenden.
MDAC in XP Pro enthalten	Die Installation ist nicht notwendig, wenn Sie Windows XP Professional verwenden. Hier ist MDAC 2.7 im Lieferumfang bereits enthalten. Im Rahmen des automatischen Updates von Windows XP wird auch diese Komponenten stets auf dem aktuellen Stand gehalten.

1.2.3 Eine Website im IIS manuell einrichten

Die nachfolgenden Einrichtungsschritte brauchen Sie nur dann vorzunehmen, wenn Sie *nicht* mit dem Visual Studio arbeiten. Dieses wird in Kapitel 2 vorgestellt. In diesem Abschnitt wird ge-

zeigt, wie Sie im IIS eine neue Website anlegen und wie Sie diese konfigurieren sollten, wenn Sie die ersten Schritte in ASP.NET mit einem einfachen Editor vornehmen wollen.

Ausgenommen hiervon ist wiederum auch das freie Entwicklungssystem *Web Matrix*. Hier erfolgt ebenfalls eine weitgehend automatisch ablaufende Übergabe des programmierten Codes an den Webserver. Mehr zu Web Matrix finden Sie unter folgender Adresse:

<http://www.asp.net/webmatrix/default.aspx>

Webmatrix wird nicht weiter betrachtet, weil wesentliche Funktionen für den professionellen Softwareentwickler fehlen, unter anderem Intellisense und Debugging.

Alle Einrichtungs- und Konfigurationsschritte am IIS und den Websites nehmen Sie über das Managementkonsolen-Snap-In *Internet-Informationdienste* vor. Sie finden eine entsprechend vorkonfigurierte Managementkonsole, wenn Sie die SYSTEMSTEUERUNG starten (in Windows XP in die klassische Ansicht umschalten) und dort das Fenster VERWALTUNG öffnen.

Webmatrix

**MMC Internet-
Informations-
dienste**

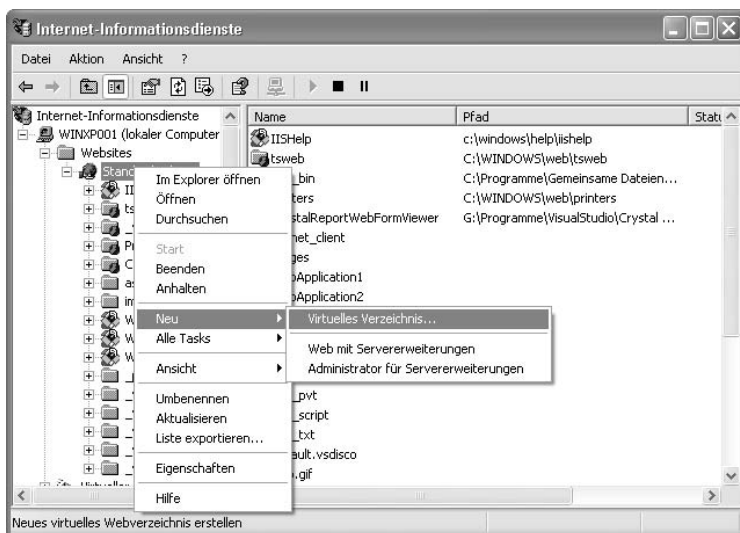


Abbildung 1.6: Anlegen eines neuen virtuellen Verzeichnisses

Neue Website einrichten

Gehen Sie wie folgt vor, um ein Verzeichnis für eine neue Website einzurichten:

1. Legen Sie über den Windows Explorer unterhalb von *%Systemdrive%\Inetpub\WWWRoot* einen neuen Ordner an, beispielsweise *\dotnet*.
2. Wählen Sie in der MMC INTERNET-INFORMATIONSDIENSTE den Eintrag STANDARDWEBSITE aus und wählen Sie aus dem Kontextmenü NEU und dann VIRTUELLES VERZEICHNIS.
3. Es startet ein Assistent, der folgende Angaben verlangt:

1. ALIAS

Ein Name, der nach außen sichtbar ist, also für den Benutzer am Browser. Die Beispiele im Buch sind unter dem Alias *aspdotnet* abgelegt.

2. VERZEICHNIS

Hier geben Sie den Pfad zu einem bereits bestehenden Ordner an. Sie können über einen Klick auf DURCHSUCHEN und dann auf NEUEN ORDNER ERSTELLEN auch an dieser Stelle einen Ordner im Dateisystem anlegen. Empfehlenswert ist die Anlage eines Ordners unterhalb von *%Systemdrive%\Inetpub\WWWRoot*.

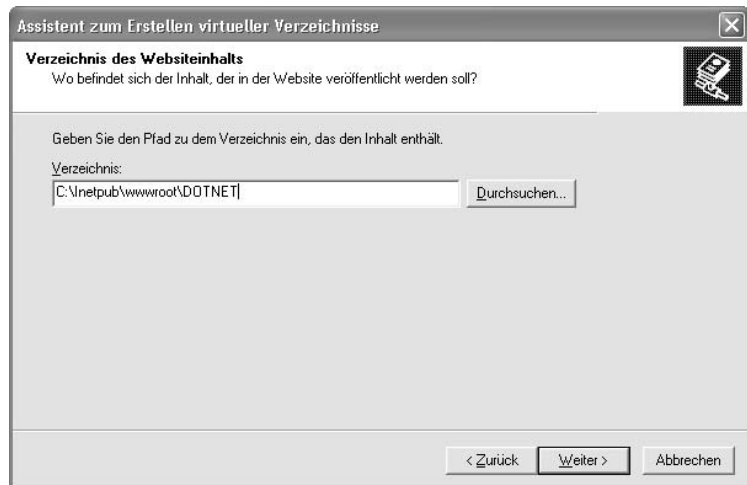


Abbildung 1.7: Pfad zum Verzeichnis angeben

3. ZUGRIFFSBERECHTIGUNGEN

In diesem Schritt geben Sie die Berechtigungen zum Zugriff an. Da ASP.NET ein Programm ist, müssen Sie neben **LESEN** und **SKRIPT** auch **AUSFÜHREN** aktivieren. Für viele Beispiele im Buch werden auch Schreibberechtigungen (**SCHREIBEN**) erwartet. Wenn Sie **DURCHSUCHEN** aktivieren, können Sie die fertigen Programme leichter auswählen. Für ein Entwicklungssystem werden deshalb sinnvollerweise alle Optionen aktiviert.

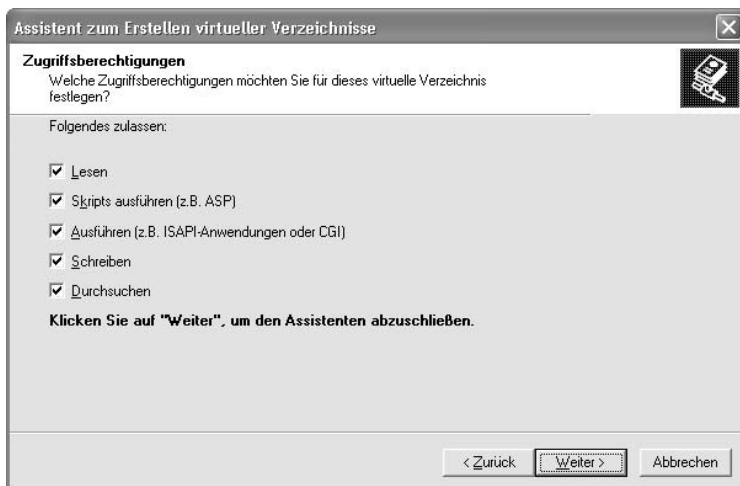


Abbildung 1.8: Zugriffsberechtigungen für ein Entwicklungssystem

Für einen Produktionsserver gilt dies natürlich nicht unbedingt. Es folgt deshalb noch eine Warnung, genau die beschriebenen Einstellungen nicht zu verwenden. Dies können Sie aber für ein Entwicklungssystem ruhig ignorieren.

1.2.4 Ein erstes Beispiel ausprobieren

Für einen ersten Test von ASP.NET brauchen Sie noch keine ausgefeilte Entwicklungsumgebung. Ein einfacher Texteditor wie der in Windows integrierte *Notepad* reicht bereits aus, um das erste Beispiel zu erfassen und auszuführen.

```

<script language="vb" runat="server">
Sub Page_Load()
    datum.Text = DateTime.Now.Day.ToString() + "." + 
        + DateTime.Now.Month.ToString() + "."
End Sub
</script>
<html>
    <head><title>Erster Test</title></head>
    <body>
    <h1>Willkommen</h1>
    Nein, diesmal nicht "Hello World".<br/>
    Eine Datumsausgabe:
    Heute ist der <asp:label id="datum" runat="server"/>
    </body>
</html>

```

Listing 1.1: Das erste ASP.NET-Programm (FirstTest.aspx)

Dateierweiterung aspx

Damit der IIS den Code abarbeiten kann, sollten Sie diesen im vorbereiteten Verzeichnis ablegen. Wichtigste Voraussetzung zur Nutzung von ASP.NET ist dabei, dass diese Datei die Erweiterung *aspx* aufweist. Geben Sie dann in der Adresszeile des Browsers den gewählten Aliasnamen an:

http://localhost/aspdotnet/firsttest.aspx

Sie sollten dann das folgende Bild im Browser sehen:

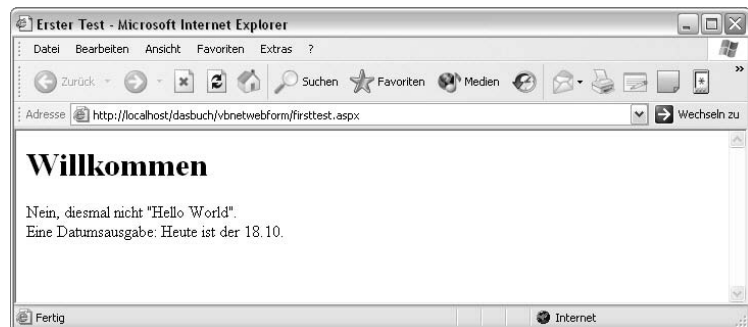


Abbildung 1.9: Ausgabe, wenn die Testdatei erfolgreich ausgeführt wurde

Fehler aufspüren

Möglicherweise haben Sie sich vertippt. Dann reagiert der Compiler mit einer ausführlichen Fehlerbeschreibung. Wenn nicht, provozieren Sie ruhig einmal einen Fehler im Codeteil (umgeben von dem Tag `<script>`), um ein Gefühl für die Reaktion von ASP.NET zu bekommen.

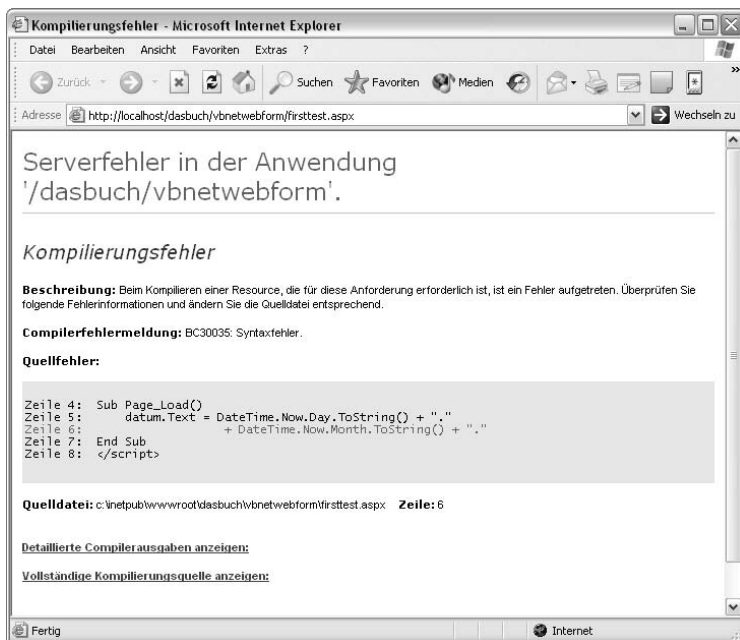


Abbildung 1.10: Fehler im Code der Seite

Hinter KOMPILIERUNGSFEHLER erhalten Sie im gezeigten Beispiel eine recht genaue Fehlerbeschreibung: Die zweite Zeile wurde ohne das Zeilenverbindungszeichen von VB.NET geschrieben. Die angezeigte Meldung gibt dies zwar nicht wirklich genau wieder – aber wenn Sie das alte ASP kennen, werden Sie feststellen, dass Sie mit diesen Compilerfehlermeldungen deutlich mehr anfangen können.

An dieser Stelle werden Sie sich vielleicht über das erste laufende Programm freuen, aber sicher ist es interessanter zu wissen, wie es funktioniert. Der Ablauf unterscheidet sich prinzipiell nur wenig von dem des Vorgängers ASP. Sie haben im Browser die Adresse Ihres Programms eingetippt. Der Browser sendet die Anforderung an den Webserver. Im IIS besteht eine Verknüpfung zwischen der Dateierweiterung *.aspx* und der ASP.NET-Komponente. Diese besteht im Wesentlichen aus einer DLL: *aspnet_isapi.dll*. Diese Komponente ist für die weitere Verarbeitung der Seite verantwortlich.

Eine Analyse des Codes zeigt das Prinzip der Verarbeitung. Zuerst wird ein `<script>`-Tag eingesetzt, das dem Server zwei Dinge

Wie es funktioniert

mitteilt: Die zu verwendende Sprache ist Visual Basic und die Verarbeitung soll auf dem Server erfolgen (`runat="server"`).

```
<script language="vb" runat="server">
```

Nun folgt der Code der Seite. *Sub* ist ein Schlüsselwort in VB.NET, das eine Methode deklariert, die nichts zurückgibt. Der Name der Prozedur ist festgelegt. `Page_Load` wird immer dann ausgeführt, wenn die Seite geladen wurde:

```
Sub Page_Load()
```

Dann wird einer Eigenschaft etwas zugewiesen. An dieser Stelle tangieren Sie das erste Mal die Welt der Objekte. In VB.NET, im .NET-Framework und damit in ASP.NET gibt es praktisch nur noch Objekte. Alles ist objektorientiert. Sie finden in diesem Buch viele Informationen über das Prinzip und die Verwendung. Zuerst aber zu der Technik dieses Codes. *datum* ist ein Objekt, dass in der nachfolgenden HTML-Seite definiert wurde. Es ist aus dem Tag `<asp:label id="datum" runat="server">` entstanden. Dieses Objekt besitzt eine Eigenschaft `Text`, der eine Zeichenkette zugewiesen werden kann. Diese entsteht durch Aufrufe von Methoden des .NET-Frameworks. Aus der Klasse `DateTime.Now`, die das aktuelle Datum enthält, wird der Tag (`Day`) und der Monat (`Month`) entnommen. Weil diese Methoden keine Zeichenketten sondern Datumswerte zurückgeben, wird noch eine Konvertierung mit `ToString` durchgeführt:

```
datum.Text = DateTime.Now.Day.ToString() + "." +  
             + DateTime.Now.Month.ToString() + "."
```

Damit ist das eigentliche Programm auch schon fertig.

```
End Sub
```

Da es sich um eine HTML-Seite handelt, die letztlich an den Browser gesendet werden soll, folgt nun deren Definition:

```
<head><title>Erster Test</title></head>  
<body>  
<h1>Willkommen</h1>  
    Nein, diesmal nicht "Hello World".<br/>  
    Eine Datumsausgabe:
```

Die einzige Besonderheit ist das spezielle ASP-Tag, dass den Text ausgibt. `<asp:label>` kann nicht mehr, als die Zeichenkette, die die Eigenschaft `Text` enthält, auszugeben:

```
Heute ist der <asp:label id="datum" runat="server"/>  
</body>  
</html>
```

ASP.NET erscheint auf den ersten Blick recht einfach. In der Praxis kommt dann doch etwas mehr auf Sie zu. Das liegt aber weniger an der Notwendigkeit, alles zu verwenden, damit überhaupt etwas läuft, sondern mehr an den Möglichkeiten, die die gelieferte Technik bietet:

- ▶ Der gesamte Sprachumfang der eingesetzten Programmiersprache steht zur Verfügung
- ▶ Sämtliche Klassen des .NET-Frameworks stehen bereit
- ▶ Viele Erweiterungen für die HTML-Programmierung sind speziell in den ASP.NET-Komponenten vorhanden

Es ist also erforderlich und sinnvoll, systematisch diese neuen Dinge zu erkunden. Wenn Sie bereits ASP programmiert haben, werden Sie schnell Gefallen daran finden, denn vieles ist einfacher und eleganter geworden. Wenn Sie noch keine Erfahrung mit ASP haben, werden Sie sich sehr schnell darin zurechtfinden, da die Struktur klar und aufgabenorientiert ist.

1.3 Grundlagen der Webserverprogrammierung

Um erfolgreich mit ASP.NET programmieren zu können sind einige wenige elementare Grundlagen notwendig. Der folgende Abschnitt enthält nur so viel Theorie, wie für die ersten Schritte im Umgang mit Webservern gebraucht wird. Weitere Informationen sollten Sie entsprechender Spezialliteratur und Informationsquellen im Internet entnehmen.

1.3.1 Das Protokoll HTTP

In diesem Abschnitt erfahren Sie das Wichtigste über HTTP (HyperText Transfer Protocol), das in der Webserver-Programmierung eine herausragende Rolle spielt.

Einführung

HTTP dient der Kommunikation mit Webservern. Es gibt zwei Versionen, 1.0 und 1.1. Auf Seiten der Browser dominiert inzwischen HTTP 1.1, denn alle Browser ab Version 4 beherrschen dieses Protokoll. Der Internet Information Server ab Version 5.0 beherrscht die Version 1.1 vollständig.

RFC 1945 Verbindungsloses Protokoll

HTTP 1.0 wurde im Mai 1996 in der RFC 1945 veröffentlicht, schon im August desselben Jahres folgte HTTP 1.1. Bei HTTP handelt es sich um ein verbindungs- oder statusloses Protokoll. Server und Client nehmen also nie einen besonderen Zustand ein, sondern beenden nach jedem Kommando den Prozess vollständig, entweder mit Erfolg oder mit einer Fehlermeldung. Es obliegt dem Kommunikationspartner, darauf in angemessener Weise zu reagieren.

Protokollaufbau, Header, Body

HTTP-Kommandos werden als ASCII-Text übertragen und können aus mehreren Zeilen bestehen. Die erste Zeile ist immer die Kommandozeile. Daran angehängt kann ein so genannter Message-Header (Kopf der Nachricht) folgen. Der Nachrichtenkopf enthält weitere Parameter, die das Kommando näher beschreiben. So kann ein *Content-Length*-Feld enthalten sein. Steht dort ein Wert größer als 0, folgen dem Nachrichtenkopf Daten. Die Daten werden also gleich zusammen mit dem Kommando gesendet, man spricht dann vom Body (Nachrichtenkörper) der Nachricht. HTTP versteht im Gegensatz zu anderen Protokollen den Umgang mit 8-Bit-Werten. Binärdaten, wie Bilder oder Sounds, müssen nicht konvertiert werden. Folgen dem HTTP-Kommando und den Nachrichtenkopf-Zeilen zwei Leerzeilen (Zeilenwechsel), so gilt das Kommando als beendet. Kommandos mit Nachrichtenkörper haben kein spezielles Ende-Zeichen, das *Content-Length*-Feld bestimmt, wie viele Bytes als Inhalt der Nachricht betrachtet werden.

Kommandoaufbau

Aufbau eines HTTP-Kommandos

Ein HTTP-Kommando hat immer folgenden Aufbau:

METHODE ID VERSION

Als **METHODE** wird das Kommando selbst bezeichnet. Die folgende Tabelle zeigt die HTTP-Kommandos auf einen Blick:

Kommando	Bedeutung
DELETE	Ressource löschen
GET	Ressource anfordern
HEAD	Header der Ressource anfordern
LINK	Verknüpfung zweier Ressourcen beantragen
OPTIONS	Optionen des Webserver erfragen
POST	Formulardaten an einen Serverprozess senden
PUT	Ressource auf dem Webserver ablegen
TRACE	Kommando zurückschicken lassen
UNLINK	Verknüpfung zwischen Ressourcen löschen

Tabelle 1.2: HTTP-Kommandos

Beachten Sie, dass die Kommandos unbedingt in Großbuchstaben geschrieben werden müssen, exakt wie in der Tabelle 1.2 gezeigt. Als Ressource werden solche Objekte bezeichnet, die übertragen werden können – in erster Linie also HTML-Dateien und Bilder.

Die ID einer Ressource kann beispielsweise eine Adresse oder ein Dateiname sein:

GET index.htm HTTP/1.0

Dieses Kommando fordert die Datei *INDEX.HTM* an.

HTTP-Statuscodes

Die Antwort auf ein Kommando besteht im Senden der Daten – wenn dies gefordert wurde – und einem Statuscode. Dem Statuscode folgen optionale Felder und, bei der Übertragung von Ressourcen, die Daten. Die Statuszeile hat folgenden Aufbau:

Statuscodes

VERSION STATUSCODE STATUSTEXT

Der Statuscode ist eine dreistellige Zahl, von der die erste Ziffer (Hunderterstelle) die Zuordnung zu einer bestimmten Gruppe anzeigt.

HTTP-Code	Bedeutung
200	Kommando erfolgreich (nach GET/POST)
201	Ressource wurde erstellt (nach PUT)
202	Authentifizierung akzeptiert (nach GET)
204	Kein Inhalt oder nicht angefordert (GET)
301	Ressource am anderen Ort
302	Ressource nicht verfügbar (temporärer Zustand)
304	Ressource wurde nicht verändert (steuert Proxy)
400	Syntaxfehler (alle Kommandos)
401	Keine Autorisierung
403	Nicht öffentlicher Bereich
404	Nicht gefunden (GET)
500	Serverfehler, Fehlfunktion
502	Kommando nicht implementiert

Tabelle 1.3: HTTP-Statuscodes

Sie werden den Fehler 404 sicher kennen. Kennen lernen werden Sie auch den Fehler Nummer 500, der erzeugt wird, wenn ein Programm nicht funktioniert, das Sie in ASP.NET geschrieben haben.

HTTP-Message-Header

An ein Kommando oder an die Statuszeile können weitere Felder angehängt werden. Der Aufbau lehnt an den MIME-Standard an:

Feldname Wert; Wert



MIME steht für Multipurpose Internet Mail Standard und definiert, wie bestimmte Dateiformate über Internet übertragen werden können. MIME wird nicht nur mit E-Mail, sondern unter anderem auch mit HTTP eingesetzt. Mehr dazu finden Sie im Exkurs »Was ist eigentlich MIME?« auf Seite 764.

Die Nachrichtenkopfheader können in drei Hauptgruppen aufgeteilt werden:

► **F**

Frage-Felder (Request-Header-Fields), die nur in Kommandos erlaubt sind

► **A**

Antwort-Felder (Response-Header-Fields), die Statusnachrichten vorbehalten sind

► I

Informationsfelder (General-Header-Fields), dienen der Übertragung aller anderen Nachrichten in die eine oder andere Richtung

Eine typische Anwendung, die bei der ASP.NET-Programmierung auftreten kann, ist die Übergabe eines Nachrichtenkopfes, der einen besonderen Dateityp angibt:

```
Content-type: application/pdf; name=aspnet.pdf
```

Freilich bietet dafür ASP.NET eine Methode innerhalb der entsprechenden Klasse an. Wenn diese Möglichkeiten aber nicht ausreichen, sind die Kenntnisse der Protokolle wichtig. Ebenso ist es hilfreich, die Begriffe zu kennen, um in großen Zahl von Klassen die passende zu finden.

Im Gegensatz zu anderen Protokollen ist die Länge eines Datenblocks im `Content-length` festgelegt, irgendwelche Begrenzungszeichen gibt es nicht. Wichtig ist auch, dass der Server nach dem Verbindungsaufbau keine Antwort sendet. Erst das erste eintreffende Kommando löst eine Reaktion aus. Darin ist die Ursache zu sehen, wenn der Browser nach der Anforderung eines unerreichbaren Server lange Zeit nicht reagiert. Als »Totsignal« wird einfach eine vorgegebene Zeitspanne gewartet, in welcher der Server auf das erste Kommando reagieren sollte.

Eine einfache HTTP-Verbindung könnte also folgendermaßen aussehen:

**Verbindungs-
ablauf**

```
Client: (Verbindungsaufbau des Browsers durch Nameserver, ➡
        TCP/IP)
Server: (keine Antwort)
Client: GET /default.aspx HTTP/1.0
Server: HTTP/1.0 200 Document follows
        Date: Mon, 04 Mar 2002 12:23:55 GMT+100 Server: ➡
        IIS 5.1, Microsoft Corporation
        Content-Type: text/html
        Last-Modified: Mon, 04 Mar 2002 12:23:55
        Content-Length: 1465
        JDGGF/(&$=$(?ED`?I`... Daten entsprechend der
        Längenangabe
```

Listing 1.2: Ablauf einer einfachen HTTP-Verbindung

Der Ablauf ist also recht simpel. Praktisch wird in diesem Beispiel eine Datei mit dem Namen *default.aspx* angefordert. ASP.NET

startet dann, führt den Code in der Seite aus, produziert den Inhalt der Seite und gibt ihn zusammen mit den richtigen Headern an den Webserver. Dieser setzt den Code 200 – Alles OK – davor und sendet alles an den Browser. Das der Benutzer dann mit den Daten etwas anfangen kann, dafür sind Sie verantwortlich. Den Rest können Sie vorerst ASP.NET und dem IIS überlassen. Profis wissen natürlich, dass sich hier trickreich eingreifen lässt. Im Normalfall ist das aber nicht notwendig.

1.3.2 Wie dynamische Webseiten entstehen

Grundlagenwissen

Unter dynamischen Webseiten werden Seiten verstanden, deren endgültige, an den Server gesendete Form erst im Augenblick des Abrufes entsteht. So können Daten interaktiv in die Seiten eingebaut werden. Der Vorteil besteht vor allem in der Möglichkeit, auf Nutzereingaben reagieren zu können. Formulare lassen sich sofort auswerten und schon die nächste Seite kann den Inhalt wiedergeben oder Reaktionen darauf zeigen. Die Anwendungsmöglichkeiten sind fast unbegrenzt. Ob und in welchem Umfang außerdem Datenbanken zum Einsatz kommen, hängt von der Zielstellung ab. Dynamische Webseiten an sich benötigen keine Datenbank. Sie sollten sich vor allem als Anfänger nicht dem Zwang unterziehen, gleich jedes Problem mit der Hilfe einer Datenbank zu lösen, auch wenn Profis dies bevorzugen würden. Im Buch werden viele Beispiele gezeigt, die mit einfachsten Mitteln beeindruckende Effekte erzielen – ganz ohne Datenbank. Die Entstehung einer dynamischen Website wird in Abbildung 1.11 erläutert. Dieser Ablauf sollte unbedingt verstanden werden, denn alle anderen, teilweise komplexeren Vorgänge in ASP.NET bauen darauf auf.

Wenn der Benutzer eine Adresse im Browser eintippt, läuft ein recht komplexer Vorgang ab:

1. Der Browser sucht einen Nameserver, um die IP-Adresse zum URL zu ermitteln.
2. Der Nameserver konsultiert gegebenenfalls weitere Server, um die IP-Adresse zu beschaffen.
3. Der Browser erhält eine IP-Adresse des Servers. Wenn das Protokoll HTTP verwendet wird, ist damit auch die Portadresse festgelegt (Port 80). IP-Adresse und Port bilden eine so genannte Socket.

4. Der Browser hat eine IP-Adresse vom Provider erhalten und einen Port für die Verbindung gebildet. Damit steht auch eine Socket zur Verfügung. Zwischen beiden Endpunkten kann nun IP-Verkehr stattfinden.
5. Der Browser sendet über diese Verbindung die Anforderung der Seite. Die erfolgt mit dem Protokoll HTTP, der entsprechende Befehl lautet GET, der Vorgang wird »Request« genannt.
6. Der Server empfängt die Anforderung und sucht die Datei. Wird sie gefunden liefert er sie aus. Dieser Vorgang wird »Response« genannt. Wird die Datei nicht gefunden, erzeugt der Server einen Fehler. Für nicht vorhandene Dateien definiert HTTP die Fehlernummer 404.
7. Der Browser empfängt Daten oder eine Fehlermeldung und zeigt diese an.

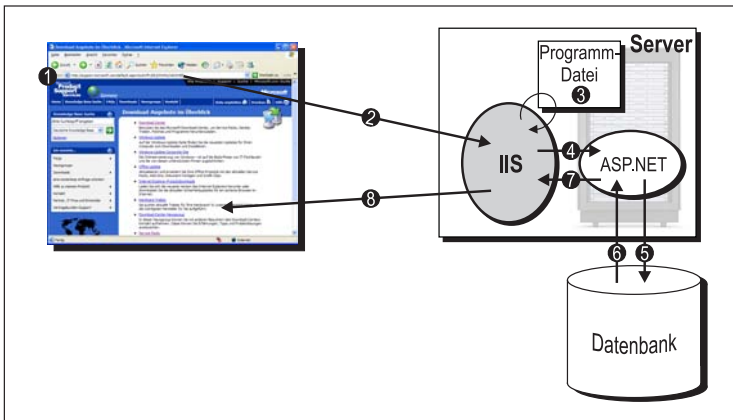


Abbildung 1.11: Ablauf der Generierung einer dynamischen Website (vereinfacht)

Zuerst fordert also der Nutzer mit seinem Browser ein Programm an. Der gesamte Vorgang ist letztlich benutzergesteuert, was die Art der Programmierung wesentlich von der Windows-Programmierung unterscheidet, auch wenn ASP.NET einige Tricks einsetzt, um den Unterschied kleiner werden zu lassen. Der Webserver leitet diese Anfrage aufgrund der Dateierweiterung an ein bestimmtes Programm weiter, bei ASP.NET an das ASP.NET-Modul *asp_isapi.dll*. Dort wird die Seite durchsucht und darin enthaltene Codes werden ausgeführt. Daraus entsteht wiederum HTML-Code, einschließlich der Daten aus Datenbankabfragen

oder früheren Nutzereingaben. Die fertige Seite wird dem Webserver zurückgegeben, der sie dann an den Browser sendet. Damit ist der Vorgang beendet. Beide Seiten »vergessen« alles, was beim Ablauf verwendet wurde. Mit der Anforderung des nächsten Objekts wird der gesamte Ablauf wiederholt. Die Vorgänge der Namensauflösung und Adressenbeschaffung laufen völlig transparent ab und sind auch bei der Programmierung kaum zu berücksichtigen. Der eigentliche Zusammenbau der Seiten ist der interessante Teil. Dies passiert in der Darstellung der Schrittfolge zwischen Schritt 4 und 7. Diesen Punkt gilt es also genauer zu untersuchen.

Für alle Probleme liefert ASP.NET interessante und hilfreiche Lösungen. Die Programmierung ist deshalb vergleichsweise einfach. Das ändert aber nichts am Prinzip oder der zugrunde liegenden Technik. Ohne das Ping-Pong-Spiel zwischen Browser und Webserver funktioniert nichts. Manchmal ist es nicht immer sichtbar, dass dieser Prozess tatsächlich abläuft, aber er wird dennoch ausnahmslos ausgeführt. Klar sollte auch sein, dass in der Webseite, wenn sie an den Browser gesendet wurde, nur noch HTML-Code steht, kein VB.NET. Auch das spezielle Element `<asp:label/>`, das im Testprogramm eingesetzt wurde, ist durch eine HTML-konforme Variante ersetzt worden. Schauen Sie sich den Quelltext der Seite nun im Browser an:

```
<html>
  <head><title>Erster Test</title>
</head>
<body>
  <h1>Willkommen</h1>
  Nein, diesmal nicht "Hello World".<br/>
  Eine Datumsausgabe: 🍷
  Heute ist der <span id="datum">8.3</span>
</body>
</html>
```

Listing 1.3: Quelltext des Testprogramms im Browser

Für das spezielle ASP.NET-Tag wurde das HTML-Tag `<span>` eingesetzt, der berechnete Text steht als fester Wert drin. Um eine Änderung zu erreichen, muss die Seite erneut beim Server angefordert werden. Welche Wege es dafür gibt, wird noch ausführlich diskutiert werden.

I.4 Grundprinzip der Programmierung mit ASP.NET

Wie der Name »ASP.NET« verrät, basiert die Form der Webserverprogrammierung auf dem .NET-Framework. Kenntnisse über das Framework selbst, die Klassen und Bibliotheken, aber auch die Produktphilosophie dahinter, sind enorm wichtig, wenn Sie damit erfolgreich programmieren möchten. Dieser Abschnitt zeigt die wichtigsten Hintergründe.

I.4.1 Was verbirgt sich hinter dem Begriff .NET?

.NET (sprich dott nett) besteht neben dem, was Sie als Softwareentwickler direkt tangiert, aus insgesamt drei Komponenten:

Die Vision

► **Die .NET-Vision**

Hier hat Microsoft seine Version der nahen Zukunft der Nutzung des Internet manifestiert. Dabei geht es darum, dass alle elektronischen Geräte über ein weltumspannendes und überall verfügbares Breitbandnetzwerk verbunden sind. Damit einher geht eine andere Art der Software- und Dienstleistungsverteilung, in deren Mittelpunkt die Webservices stehen.

► **Das .NET-Framework**

Um Entwickler wie Anwender mit den Technologien zu versorgen, wird ein großer Teil der Dienste als Framework geliefert. Sie haben es bereits erfolgreich installiert, es ist also nicht mehr und nicht weniger als ein großes Stück Software.

► **Die .NET-Server**

Um in lokalen oder globalen Netzwerken selbst Dienstleistungen anbieten zu können, mit denen die Vision in Erfüllung gehen kann, werden Enterprise Server eingesetzt. Dazu gehören der SQL Server 2000 und der BizTalk Server 2000. Beide basieren, ebenso wie die zahlreichen anderen Server, noch nicht auf dem .NET-Framework, nutzen aber die Technologie. Dies gilt auch für die Windows Server 2003, die Nachfolger der Windows 2000 Server.

Das .NET-Framework

Das Framework ist die Kerntechnologie des Marketingbegriffs .NET. Es zerfällt bei näherer Betrachtung in folgende Teile:

► Die .NET-Framework Klassenbibliothek

Hier finden die bereits erwähnten Klassen, die von der Netzwerkprogrammierung über Dateisystemzugriff bis zu dynamischen Bildern alles bieten, was bei der Programmierung notwendig sein könnte.

► Die .NET-Sprachen

Dazu gehören im Lieferumfang C#, VB, J#, JScript und C++. Weitere können von Drittanbietern bereitgestellt werden.

► Die Common Language Runtime (CLR)

Dies ist die Laufzeitbibliothek, die für die Ausführung der übersetzten Programme sorgt. Dazu wird der MSIL-Code Just-In-Time, also zur Laufzeit des Programms beim ersten Start, in Maschinensprache übersetzt.

► Die Microsoft Intermediate Language (MSIL)

Dies ist eine Zwischensprache, in die alle in den verschiedenen .NET-Sprachen übersetzt werden und die von der CLR ausgeführt werden kann.

► ASP.NET

Die Erweiterung für den IIS zur Erstellung dynamischer Webseiten.

Compiler und Interpreter

Zwischen dem Code, den Sie in Ihren Editor eintippen und dem vom Server ausgeführten liegt ein langer Weg. Bislang gab es nur zwei Arten von derartigen Übersetzern von vom Menschen geschaffenen Code in maschinenlesbaren: Compiler und Interpreter. Compiler übersetzen den Code in einem explizit gestarteten Lauf, benötigen dazu ein besonderes Programm und sind vergleichsweise aufwändig zu bedienen. Das Testen kompilierter Codes ist nicht einfach, weil jede Änderung erst nach der erneuten Übersetzung wirksam wird. Sie benötigen außerdem einen Debugger zur Fehlersuche, der Zustände während der Laufzeit des Programms abfragt und anzeigt. Auch solche Programme sind nicht trivial und erfordern einiges Verständnis für die inneren Zusammenhänge. Auf der anderen Seite gibt es Interpreter, die erst im Augenblick des Abrufes eines Programms die Übersetzung Zeile für Zeile vornehmen. Änderungen wirken sich sofort aus und Programme zur Übersetzung sind nicht notwendig. Es liegt in der Natur der Sache, dass Anfänger mit Interpretern besser und

schneller zurecht kommen und sich derartige Programme einer gewissen Beliebtheit erfreuen. Das alte ASP basiert auf einem Interpreter, ebenso wie die bekannten Skriptsprachen PHP und Perl. Windows-Programme in C++, C-Programme unter Unix und Java werden dagegen kompiliert – vor allem eine Domäne professioneller Entwickler.

Mit .NET ändert sich an dieser Stelle etwas. Zum einen werden ASP.NET-Programme, ebenso wie jede andere Anwendung, grundsätzlich kompiliert. Das haben Sie bereits getan und es vermutlich nicht einmal bemerkt. Tatsächlich dauert der erste Aufruf einer ASP.NET-Seite etwas länger als alle folgenden. Die ASP.NET-Komponente erkennt, dass die Seite noch nicht übersetzt wurde und führt die Übersetzung sofort und ohne jede Interaktion aus. Sie können das zwar durch diverse Einträge in den Code steuern, müssen es aber nicht. Dem Compiler ist damit der Schrecken genommen. Fehlermeldungen werden in einer gut lesbaren Form an den Browser gesendet. Die Übersetzung erfolgt jedoch nicht direkt in nativen Maschinencode für die CPU, sondern in eine Zwischensprache – die bereits erwähnte MSIL. Dies ist ein maschinennaher Code, der sehr viel schneller abgearbeitet werden kann, als es ein Interpreter mit dem Quellcode könnte. Dieser Code wird von der Common Language Runtime (CLR) ausgeführt, letztlich eine Art spezieller Compiler. Dieser Compiler ist ein so genannter Just-In-Time-Compiler (JIT-Compiler). Er übersetzt ein Stück Code beim Abruf in Maschinensprache und speichert ihn dann, sodass nur bei Änderungen eine erneute Übersetzung notwendig wird.

Darin liegt das Geheimnis der Sprachunabhängigkeit. Der MSIL-Code, den C# erzeugt, ist identisch mit dem von VB.NET oder JScript.NET und umgekehrt. Die MSIL kann man zwar anschauen, aber sie ist nicht dafür entworfen worden, von Menschen gelesen zu werden. Wenn Sie Assembler kennen, werden Ihnen Code-teile sicher bekannt vorkommen.

Sprachunabhängigkeit dank MSIL-Code

Die Ausführung der Übersetzung in MSIL und die Ausführung mit der CLR wird beim Abruf von ASP.NET-Programmen automatisch erfolgen. Man muss sich aber dieses Prinzip vor Augen halten, um das Laufzeitverhalten zu verstehen und auch den Zeitpunkt, an dem Fehlermeldungen ausgegeben werden. Es gibt Fehler, die treten während der ersten Übersetzungsphase auf und an-

dere erst beim Auftreten bestimmter Daten. Sie werden beides im Laufe Ihrer Arbeit mit ASP.NET mit Sicherheit kennen lernen.

Common Type System

In der .NET-Welt ist weiterhin immer wieder von den Klassen des Frameworks die Rede, genannt Basisklassen. Diese Klassen liefern alles, was im Programmieralltag benötigt wird. Vor allem aber – und dies ist ein Unterschied zu anderen Klassensystemen – liefern sie auch ein einheitliches Typsystem. Bislang kannte jede Programmiersprache eigene Datentypen; Ganze Zahlen (Integer), Zeichenketten oder komplexe Typen wie Arrays. Wenn nun ein Teil in C# und ein anderer in VB.NET geschrieben wird, beide aber reibungslos zusammenarbeiten müssen, funktioniert das nur, wenn sich auch die Datentypen angleichen. Dies würde jedoch zu Kompromissen in allen eingesetzten Sprachen führen. Deshalb sind diese Spracheigenschaften in das Common Type System (CTS) ausgelagert.

Die Wahl der Datentypen und deren Präsentation ist von großer Bedeutung bei der Programmierung. Wenn Sie bereits JScript oder VBScript programmiert haben, werden Sie den Begriff »Datentyp« nur am Rande registriert haben. Skriptsprachen arbeiten typlos oder mit sehr losen Typen, die zur Laufzeit vom System selbst vergeben werden. .NET basiert auf einem sehr strengen Typkonzept – dies gilt für alle Sprachen gleichermaßen. Sie müssen sich also stets Gedanken darüber machen, welchen Typ eine Variable besitzen soll, das heißt, welche Datenart darin gehalten wird. Typbezeichner stehen natürlich weiterhin zur Verfügung. Intern gibt es aber eine Verknüpfung zwischen dem vom Framework gelieferten und dem in der Sprache definierten Typ.

Ein Datentyp des Frameworks ist beispielsweise `System.Int32`. In C# nutzen Sie für ganze Zahlen `int`, in VB.NET `Integer`. Beides wird bei der Übersetzung in `System.Int32` umgewandelt. Das Framework kennt mehr Datentypen als die Sprachen, die jeweils mit einem Basissatz ausgestattet sind. Sie können immer direkt die Datentypen des Frameworks deklarieren. Wie das erfolgt, wird in der Spracheinführung zu VB.NET erläutert.

1.4.2 Überblick über das Framework

Die nachfolgende Abbildung gibt einen Überblick über die grundlegenden Bestandteile des Frameworks und deren Zusammenspiel:

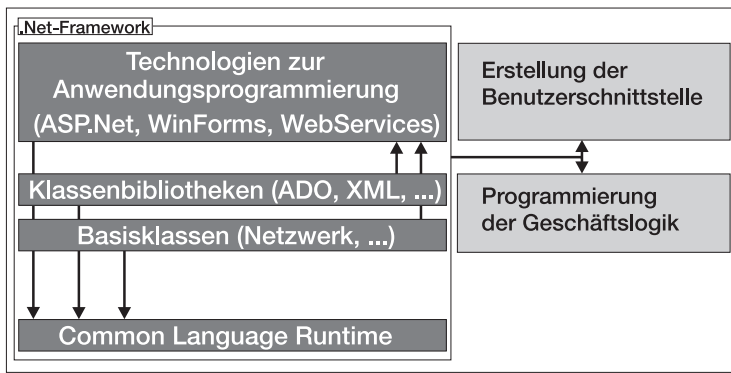


Abbildung 1.12: Grundlegender Aufbau des Frameworks

Allen Bestandteilen liegt die Common Language Runtime zugrunde. Darauf setzen die Basisklassen auf. Basisklassen sind solche für einfache Datenmodelle (`System.Collections`), Multithreading (`System.Threading`) oder IO (`System.IO`) für den Zugriff auf das Dateisystem. Von den Basisklassen abgeleitet und ergänzt folgen die Klassenbibliotheken. Dazu gehören die Bibliotheken für den Datenbankzugriff ADO.NET, die XML-Bibliotheken oder solche für reguläre Ausdrücke. Noch komplexere Aufgaben erledigen die Klassen, die für den Anwendungsprogrammierer interessant sind: ASP.NET, WinForms und Webservices sind die wichtigsten Vertreter. In diesem Buch wird ASP.NET behandelt. Verwechseln Sie das nicht mit *Webservices*, die zwar auch auf Webservern ausgeführt werden, aber nicht einen Browser als Client erwarten, sondern einen anderen Server.

Grundlage: CLR

Der Fokus ASP.NET bedeutet aber nicht, dass die anderen Stufen außer acht gelassen werden können. In vielen Abschnitten werden immer wieder Klassen aus allen drei Stufen der Bibliotheken verwendet werden. Dabei muss ein Zugriff auf eine Basisklasse keinesfalls komplizierter sein, als der auf eine Anwendungs-klasse.

Die Kunst beim Umgang mit dem Framework besteht im Wesentlichen darin, bei Bedarf den passenden Namensraum und darin die richtige Klasse zu finden. Die Online-Referenz ist dabei ein sehr wichtiges Arbeitsmittel, im Web unter folgender Adresse zu finden:

Referenz

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

Für dieses Buch ist ein Ausschnitt aus verschiedenen Namensräumen verwendet worden. Am Anfang der Einführungen finden Sie teilweise Ausschnitte aus der Klassenhierarchie. Diese Begriffe sind ein guter Startpunkt bei der Suche in der Referenz.

1.4.3 Die Welt der Objekte

Die Datentypen des CTS wurden bereits erwähnt. Zum Verständnis der Schreibweise müssen Sie sich zwangsläufig mit Objekten beschäftigen. Außerdem unterscheidet sich das Programmierprinzip des Frameworks etwas von der klassischen Programmierung. Dies wird in diesem Abschnitt erläutert.

Alles sind Objekte

In .NET sind alle Dinge, mit denen Sie zu tun haben, Objekte. Auch wenn es nicht danach aussieht oder Sie dies nicht erwarten – es sind und bleiben Objekte. Objekte bieten für die moderne Programmierung Vorteile. Wer mit programmieren anfängt oder nur VBA oder VBScript kennt, wird sich damit schwer tun. Diesen Schritt müssen Sie am Anfang gehen, denn sonst werden Sie auch die einfachsten Beispiele nicht lesen können.

Wenn es schwer ist, warum legt Microsoft dann solchen Wert darauf, die Welt der Objekte zur alleinigen Herrschaft in der Programmierwelt zu führen? In einfachen Applikationen, die nur wenige Zeilen Code enthalten, gibt es tatsächlich kaum Vorteile. Wenn jedoch größeren Anwendungen geschrieben werden, wofür auch ASP.NET bestens geeignet ist, dann wird es schwer, die Übersicht zu behalten. Der erste Schritt besteht darin, Techniken zu schaffen, die Codes wiederverwendbar machen. Dadurch werden Programme kleiner und die Fehlerquote sinkt, weil man auf geprüfte und ausgereifte Module zurückgreifen konnte. Im alten ASP war die einzige Möglichkeit der Modularisierung die Verwendung der SSI-Anweisung `INCLUDE`. Das ist völlig unzureichend, wenn man die Möglichkeiten der objektorientierten Programmierung daneben stellt.

Was ist ein Objekt?

Ein Objekt in Software – hier immer vereinfacht als Objekt bezeichnet – ist eine Sammlung von Code, der etwas aus der realen Welt konkret beschreibt. Das können Dinge wie Tiere oder Autos sein, aber auch Zustände oder Pläne wie Kalender oder Aufgaben. Man kann alles in der realen Welt als Objekt betrachten und deshalb ist die Abbildung in einer ähnlich Form in Software eigentlich genial. Objekte können in weitere, kleinere Objekte zerfallen.

Stellen Sie sich ein Auto vor. Es zerfällt in Bauteile wie Räder, Türen, den Motor usw. Jedes Teil teilt sich wiederum in weitere Bauteile. Treibt man diese Überlegung sehr weit, endet man bei den Elementarteilchen. Irgendwo zwischen Elektronen und Quarks gibt es ein Basisobjekt.

Vergleichen Sie das mit einem komplexen Programm, beispielsweise Windows. Dort gibt es ganz elementare Objekte, die die Mausbewegung erfassen oder Punkte und Linien zeichnen. Aus diesen entsteht – über viele Stufen – ein bewegliches Fenster. So funktionieren Softwareobjekte.

Der Vorgang von der Abbildung realer Dinge in Software wird als Abstraktion bezeichnet. Als Programmierer muss man deshalb nicht nur logisch sondern auch abstrakt denken können. Sie versuchen dabei das, was Sie aus der Realität abbilden möchten, in ein abstraktes Modell zu packen. Je besser Ihnen das gelingt, desto einfacher und stringenter ist die Lösung. Guter objektorientierter Code hat einige Eigenschaften:

- ▶ Er repräsentiert die menschliche Sicht auf die abgebildeten Dinge.
- ▶ Er hilft, Code effizienter zu schreiben.
- ▶ Er hilft, Code einfacher zu schreiben.
- ▶ Der Code ist leicht wieder verwendbar und leicht anpassbar.
- ▶ Guter Code bildet Dinge nicht kryptisch, sondern direkt ab.

Das klingt alles sehr gut, ist aber in der Praxis nur mit einiger Erfahrung zu erreichen. .NET hilft Ihnen aber dabei, denn wenn alles ein Objekt ist und VB.NET nun eine echte objektorientierte Programmiersprache, dann kann das Schreiben entsprechenden Codes nicht schwer sein. Objekt haben Eigenschaften. Das ist notwendig, um Dinge zu beschreiben. Die Wahl der passenden Eigenschaften ist wichtig. Sie sollten immer im Kontext der Verwendung betrachtet werden. Die Eigenschaften eines Autos sind beispielsweise Farbe, Länge oder Marke. Vielleicht denken Sie, dass die Leistung auch eine gute Eigenschaft wäre. Das ist zu wenig abstrahiert betrachtet. Denn das Objekt Auto enthält ein Objekt Motor, für das die Eigenschaft Leistung viel besser geeignet wäre. Ein Auto kann nämlich verschiedene Motoren haben und damit auch jeweils eine andere Leistung. Wird das Auto-Objekt aus dem Motor-Objekt abgeleitet, erbt es quasi dessen Eigenschaften mit. Sowohl Motor als auch Auto sind dann wieder verwend-

**Objekte haben
Eigenschaften**

Objekte haben Methoden

bar – wie in der realen Welt. Dinge haben nicht nur Eigenschaften, sie können auch Aktionen ausführen. Ein Auto kann fahren oder Türen öffnen und schließen. Steigt man in der Objekthierarchie wieder noch unten – zur Wurzel hin, werden andere Methoden interessant. So wäre für das Getriebe eine Methode »Gang höher« oder »Gang niedriger« interessant. Eine Methode »3. Gang« ist dagegen nicht sinnvoll, weil es nicht immer möglich und sinnvoll sein kann, direkt in einen Gang zu schalten. Dagegen ist eine Eigenschaft »Gang« besser geeignet, da dort der aktuelle Zustand abgefragt werden kann. Eben diese Überlegungen sollten auch bei Software angestellt werden, was nicht so schwer ist, weil es nur um die Abbildung realer Dinge geht – theoretisch jedenfalls.

Die bisherige Beschreibung sagt noch nichts über die Verwendung aus. Es ist sinnvoll, Objekte so zu gestalten, dass man sie einsetzt, ohne das Innenleben zu kennen. Der Inhalt ist gekapselt. Sie können Auto fahren, ohne genau zu wissen, wie jedes Bauteil aufgebaut ist und funktioniert. Ebenso verhält es sich bei Objekten. Sie sollten so konstruiert sein, dass sie einfach zu benutzen sind.

Baupläne für Objekte

Objekte entstehen nicht einfach so, als Einzelstücke, sondern auf der Basis eines Bauplanes. Schließlich will man nicht nur ein Auto herstellen, sondern möglichst viele nach demselben Muster. Ein solcher Bauplan wird in objektorientierter Software eine Klasse genannt. Die Klassen des .NET-Frameworks sind also eine gigantische Sammlung von Bauplänen für Ihre Software. Wenn aus einer Klasse ein Objekt abgeleitet wird, dann spricht man von Instanziierung – das Objekt ist eine Instanz der Klasse.

Eine genaue Herleitung der Schlüsselwörter, die für Klassen und Objekte in VB.NET genutzt werden, finden Sie im Abschnitt 3.4.8, »Namensräume und Klassen« ab Seite 176 bei der Beschreibung der Sprachelemente.

Namensräume

Im Zusammenhang mit der Programmierung in .NET stößt man schnell auf den Begriff Namensraum (engl. Namespace). Namensräume haben zwei grundlegende Aufgaben:

- ▶ Sie teilen zusammengehörende Systemtypen in logische Gruppen ein.
- ▶ Sie verhindern Namenskonflikte zwischen Dateien.

Systemtypen sind in .NET alles, was die Basis für Objekte ist. Dies sind die schon erwähnten Klassen, aber auch komplexe Strukturi-

ren, Datenfelder, Aufzählungen und Schnittstellen – letztere sind quasi spezialisierte Baupläne. Jeder derartige Systemtyp liegt in irgendeinem Namensraum vor. Da damit Namenskonflikte vermieden werden, können innerhalb eines Bauplanes für Eigenschaften und Methoden Namen verwendet werden, die auch außerhalb des Bauplanes auftreten. Dies ist eine wichtige Eigenschaft objektorientierter Programmierung, denn so können Sie ihre Baupläne anderen Programmierern zur Verfügung stellen oder solche von anderen benutzen, ohne Angst zu haben, dass sich Konflikte ergeben. In VBScript war dies ein großes Problem, weil es nur einen einzigen Namensraum gab. In .NET gibt es viele – unendlich viele – durch immer neue Definitionen.

Namensräume existieren nicht physisch, sie werden also nicht dadurch gebildet, dass der Code in DLLs oder ausführbaren Dateien oder Modulen untergebracht wird. Wenn Sie sich Namensräume vorstellen wollen, denken Sie an so etwas ähnliches wie Schubladen oder Ordner. Zwei Schubladen können zwei gleiche Objekte enthalten und diese kann man trotzdem anhand der Lage in dem einen oder anderen Schubfach unterscheiden.

Die Verwendung eines Namensraumes muss in VB.NET erklärt werden, dazu dient das Schlüsselwort *Imports*. Sie werden das in vielen Programmbeispielen aber nicht sehen, weil ASP.NET die wichtigsten bereits automatisch importiert. Auch dies wird natürlich noch genauer erörtert.

Übrigens wird oft auch keine richtige Klasse geschrieben und ein Objekt daraus abgeleitet. Viele ASP.NET-Beispiele sehen wie konventioneller prozeduraler Code aus. Tatsächlich nutzt aber ASP.NET Klassen des Frameworks, um aus Ihrem Code komplette objektorientierte Quellen zu erzeugen und diese dann zu übersetzen. Dies ist ein Trick, der den Einstieg in ASP.NET vereinfacht und von dem Windows-Programmierer nicht profitieren können. Mehr Vorteile für ASP.NET-Entwickler werden im nächsten Abschnitt gezeigt.

1.4.4 Der Übersetzungsvorgang

Das ASP.NET kein Interpreter ist, sondern ein Compiler, wurde bereits angesprochen. Der Vorgang des Übersetzens und der Nutzung der Codes ist weitgehend transparent, sodass Sie dafür nichts tun müssen. Dennoch läuft dies nicht im geheimen Inneren

von Windows ab, sondern ist sowohl sichtbar als auch kontrollierbar. ASP.NET erzeugt in einem ersten Lauf aus der Seite, die sowohl HTML als auch Code in einer der .NET-Sprachen enthält, eine so genannte »Page«. Der Page-Compiler erstellt daraus eine Page-Klasse, wozu auf verschiedene Basisklassen zurückgegriffen wird (System.CodeDOM und System.DLL). Diese Klasse wird übersetzt und als Assembly abgelegt.

Assemblies Assemblies sind die Nachfolger der DLLs, wobei die Dateierweiterung *dll* erhalten geblieben ist. Assemblies enthalten viele Zusatz- und Konfigurationsinformationen, sodass die Verwaltung einfacher und sicherer ist, als bei DLLs. Wenn Sie das erste Beispiel ausgeführt haben, können Sie die DLL bereits sehen. Sie ist in folgendem Pfad zu finden:

```
%Systemroot%  
  \Microsoft.NET  
    \Framework  
      \<version>  
        \Temporary ASP.NET Files
```

Haben Sie ein virtuelles Verzeichnis mit dem Namen *aspdotnet* verwendet, folgt dieser Name als weiterer Ordner. Darunter sind möglicherweise mehrere Verzeichnisse zu finden, die Versionsnummern enthalten. So können mehrere Versionen koexistieren. Darunter wiederum liegen die DLLs – die Assemblies. Sie müssen das nicht weiter beachten; wenn Ihnen der Pfad nicht gefällt, können Sie ihn auch ändern. ASP.NET ist sehr offen, was die Einflussnahme angeht und auch in der Darstellung dessen, was während der Laufzeit passiert.

ASP.NET in Windows Sie arbeiten, trotz .NET-Framework und ASP.NET, natürlich immer noch im klassischen Windows 2000 oder XP. Auch die neuen .NET-Server dienen lediglich als Host für .NET, sind aber nicht selbst in .NET programmiert. Für ASP.NET ist deshalb interessant, wo die Schnittstelle zwischen dem alten und neuen System ist – es ist der Webserver IIS. ASP.NET ist eine herkömmliche ISAPI-Anwendung.

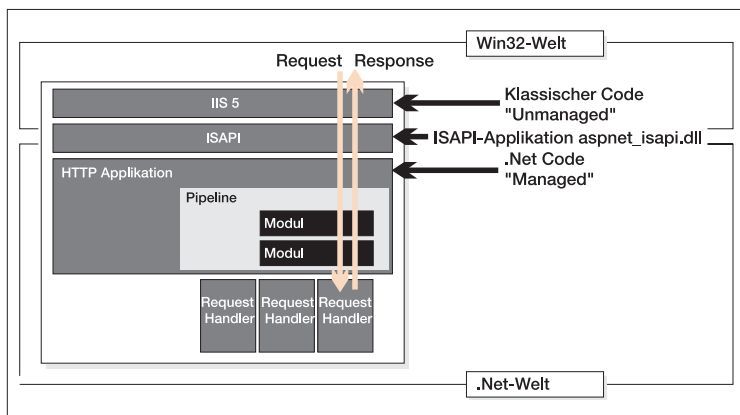


Abbildung 1.13: Struktur der Anforderungsverarbeitung in ASP.NET

Im Kern basiert ASP.NET auf einer HTTP-Applikation, der HTTP-Laufzeitumgebung. Diese Komponente empfängt von der ISAPI-Anwendung die Anforderung (so genannter *Request* vom Browser) und leitet sie über Module – die sie programmieren können – an den *Request Handler* weiter. Dieser Teil ist für die Ausführung des angeforderten Codes verantwortlich. Er kann aber auch nur einfache statische Seiten ausliefern, wenn diese zwar wegen der Erweiterung *.aspx* den Prozess in Gang setzten, aber keine Programmierung enthielten. Was im Wesentlichen hier erledigt wird, ist die korrekte Nutzung des Protokolls HTTP. Darüber müssen Sie sich keine Gedanken machen. Es ist aber auf der anderen Seite hilfreich, HTTP zu kennen. Denn dann fällt es Ihnen leichter zu verstehen, was die entsprechenden Methoden und Eigenschaften der HTTP-Handler-Klassen bedeuten und wie man sie sinnvoll in der Programmierung einsetzen kann. Der nächste Abschnitt bietet deshalb unter anderem eine kompakte Einführung in das Protokoll HTTP.

**Das Konzept der
»Handler«**

1.4.5 Wie die Seite im Server verarbeitet wird

Für den korrekten Umgang mit ASP.NET-Applikationen müssen Sie einige Prinzipien der Anforderungsverarbeitung kennen. ASP.NET arbeitet ereignisorientiert. Das Prinzip der objektorientierten Programmierung gilt auch hier – ohne jede Ausnahme. Den Ablauf beim Laden und Ausführen einer Webseite auf dem Server muss man als Folge von Ereignissen betrachten. Sie können davon profitieren, da alle Ereignisse in entsprechend benannten

Methoden ausgeführt werden und Sie diese als Programmierer mit eigenem Code füllen können.

Nichtsdestotrotz können Sie weiterhin Code einfach in die HTML-Seite einbetten. Dieser wird nach den Initialisierungsfunktionen fortlaufend entsprechend dem Auftreten in der Seite ausgeführt. Da die ASP.NET-Komponente aus der HTML-Seite erst eine reguläre Klasse erstellt – natürlich in der verwendeten Sprache –, wird der Code in der Reihenfolge der Erfassung abgelegt. Davon weichen spezielle Methoden ab, die durch Ereignisse zu bestimmten, genau definierten, Zeitpunkten aktiviert werden. Die Darstellung in diesem Abschnitt gibt einen groben Überblick über die Abhängigkeiten auf der ASP.NET-Seite. Der IIS wird hier bewusst aus der Betrachtung ausgeklammert.

Am Anfang des Zyklus steht die Anforderung vom Browser:

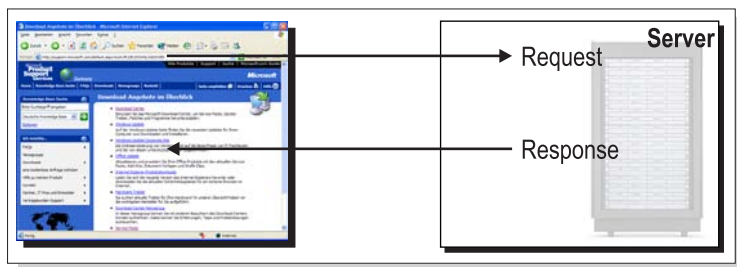


Abbildung 1.14: Basis der Verarbeitung: Anforderung und Antwort

Der Applikationszyklus einer Anforderung

Der Zyklus der Seitenverarbeitung ist Teil des applikationsweiten Verarbeitungszyklus der Anforderung vom Client. Dieser wird ausgelöst durch die vom Client empfangenen HTTP-Anforderung per GET oder POST. In der Folge kommt es zu einer Vielzahl von Ereignissen und Verarbeitungsschritten, bis im Ergebnis die Antwort feststeht und als Response an den Browser gesendet wird.

Eine genaue Betrachtung zu diesem Ablauf und praktische Hinweise zum Umgang damit finden Sie im Abschnitt 8.6, »Applikationsmanagement« ab Seite 802.

An dieser Stelle soll ein vereinfachter Blick auf den Applikationszyklus helfen. Beachten Sie, dass die Verarbeitung der angeforderten Seite hier eingebettet ist:

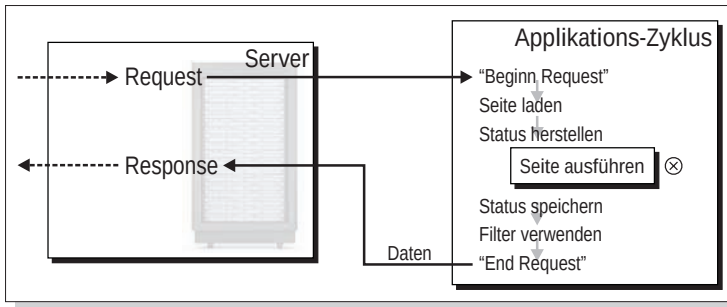


Abbildung 1.15: Der Applikationszyklus, hier etwas vereinfacht

Wird die Seite verarbeitet, beginnt der Lebenszyklus einer Seite und der darauf befindlichen Steuerelemente. Als Steuerelemente werden alle zur Anzeige in HTML eingesetzten und zugleich vom Programm verwalteten Codes der Seitenvorlagen bezeichnet.

Der Lebenszyklus einer Seite und ihrer Steuerelemente

Jedes Mal, wenn von einem Client eine *aspx*-Seite angefordert wird, beginnt deren so genannter Lebenszyklus. Der auslösende Moment ist Teil des zuvor gezeigten Applikationszyklus. Im Rahmen dieses Prozesses werden die von Ihnen programmierten Steuerelement, HTML-Code und sonstiger Programmcode analysiert und ausgeführt. Jeder Abschnitt in diesem Ablauf löst bestimmte Ereignisse aus. ASP.NET erkennt diese und führt daraufhin bestimmte Aktionen aus. Es ist aber auch möglich, im Code einige (selten aller) dieser Ereignisse an eigene Ereignisbehandlungsmethoden zu leiten und dort zu genau definierten Zeitpunkten Aktionen zu veranlassen. Bei der praktischen Programmierung wird genau dies gemacht. Praktisch jedes Beispiel in diesem Buch basiert auf dieser Technik, wenngleich in einer meist sehr einfachen Form. Für das Verständnis dieses Prozesses ist es sehr wichtig, die vom System durchlaufenen Phasen des Lebenszyklus zu kennen und zu entscheiden, wo der Programmcode am besten abläuft.

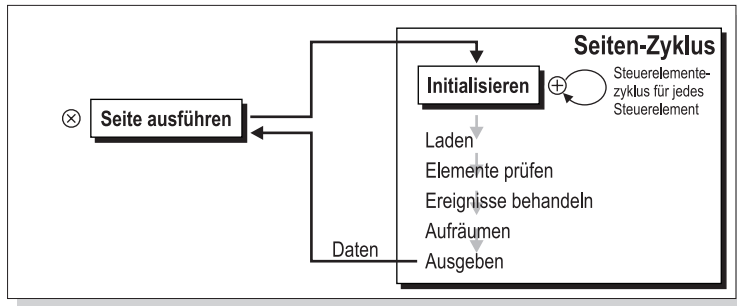


Abbildung I.16: Der Lebenszyklus einer Seite

Wenn Sie sich mit dem Zyklus der gesamten Seite auseinander setzen, ist es auch wichtig zu verstehen, dass jedes Steuerelement – beispielsweise ein Eingabefeld – ebenfalls einen solchen Zyklus durchläuft.

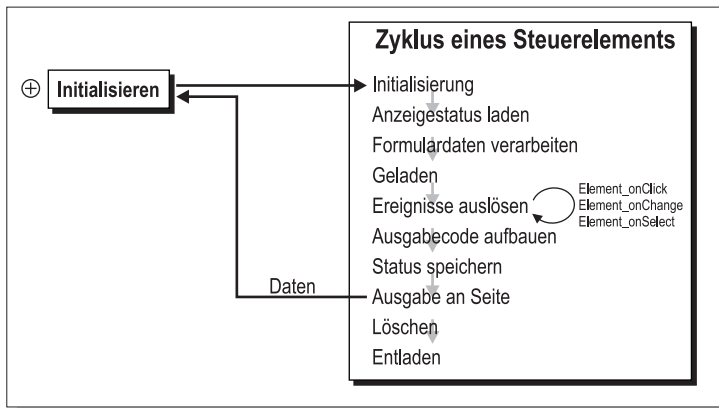


Abbildung I.17: Der Zyklus eines Steuerelements

Der Zyklus der Steuerelemente

Die folgende Tabelle zeigt die Phasen und deren Aktivitäten sowie das zugeordnete Ereignis und den Namen der dazu passenden Ereignisbehandlungsmethode, wie es für jedes Steuerelement der Seite durchlaufen wird. Anschließend wird auf den globalen Zyklus der Seite eingegangen.

Phase	Aktion	Ereignis	Methode
Initialisierung	Beginn der Verarbeitung, Lesen der Steuer-elemente	Init	OnInit
Anzeigestatus laden	Laden des Anzeigestatus (ViewState) und Zuwei-sen der Werte an die Steuerelemente	–	LoadViewState
Formulardaten verarbeiten	Verarbeiten der zurück-gesendeten Daten aus einem Formular (Post-Back)	–	LoadPostData
Laden	Alle Steuerelemente sind gelesen, gefüllt und be-kannt	Load	OnLoad
PostBack senden	Auslösen von Ereignis-sen, die an sich ändernde Steuerelemente gebun-den sind	–	RaisePostData-ChangedEvent
Vor der Ausgabe	Seite ist fertig, Änderun-gen an Steuerelementen können noch erfolgen	PreRender	OnPreRender
Status speichern	Aufbau des neuen Anzei-gestatus (ViewState)	–	SaveViewState
Ausgabe	Erzeugen der Ausgabe für den Client	–	Render
Löschen	Bereinigen des Speichers vor dem Entfernen des Steuerelements selbst	–	Dispose
Entladen	Entfernen des Steuerele-ments aus dem Speicher	UnLoad	OnUnLoad

Tabelle 1.4: Lebenszyklus der Steuerelemente mit Ereignissen

Dieser Zyklus wird nun für jedes Element durchlaufen. Die Daten werden jeweils gesammelt, da üblicherweise viele Steuerelemente auf einer Seite platziert sind.

Der Zyklus der Seite durchläuft eine ähnliche Phase, wie sie für jedes Steuerelement auftreten. Auch hier bestehen Eingriffsmöglichkeiten und es sind Überlegungen notwendig, wo der Code platziert wird.

Die folgende Tabelle zeigt die Phasen, deren Bedeutung und die zugeordneten Ereignisse und deren Ereignisbehandlungsmethoden.

Der Zyklus der Seite

Phase	Aktion	Ereignis	Behandlung
Initialisierung	Steuerelemente werden eingesammelt und der Anzeigestatus wird hergestellt	Init	Page_Init
Benutzercode initialisieren	Die Seite ist geladen und hergestellt	Load	Page_Load
Gültigkeit prüfen	Kontroll-Steuerelemente mit Gültigkeitsprüfung werden geprüft und das Ergebnis löst ggf. ein Ereignis aus	Kein direkt zugeordnetes Ereignis	
Ereignisbehandlung	Die Ereignisse, die an Steuerelemente gebunden sind oder von denen ausgelöst werden, werden nun verarbeitet	Die hier aufgerufenen Ereignisbehandlungsmethoden müssen Sie selbst programmieren	
Aufräumen	Die Seite wurde dargestellt und der benutzte Speicher wird anschließend freigegeben	UnLoad	Page_UnLoad

Tabelle 1.5: Lebenszyklus einer abgerufenen Seite mit Ereignissen

Der Zyklus der Steuerelemente wird also in der Phase der Initialisierung (Phase 1) des Lebenszyklus der Seite durchlaufen. Sie können deshalb im eigenen Code nicht Änderungen an Steuerelementen vornehmen, die vor dem Zuweisen des Status notwendig wären.

Typische Vorgänge innerhalb der Ereignisbehandlungsmethoden

Page_Init
Page_Load
Page_Unload

Es stellt sich nun die Frage, was Sie an den Zeitpunkten bestimmter Phasen realisieren können. Nach dem Senden einer Seite ist es beispielsweise sinnvoll, zu erkennen, ob es sich um ein zurück-gesendetes Formular handelt. Die Eigenschaft `IsPostBack` zeigt dies an. Ein guter Zeitpunkt zur Abfrage ist `Page_Load`, denn hier sind mit Sicherheit alle Steuerelemente (= Objekte der Formularfelder) der Seite geladen und mit den Werten gefüllt, sodass die Auswertung beginnen kann. Wird die Seite dagegen das erste Mal aufgerufen, fragen Sie eine Datenbank ab oder bauen den Anfangsdatenbestand anderweitig auf. Dann werden die Standardwerte der Steuerelemente für das erste Aussenden eines Formulars gesetzt.

Bei der Auswertung eines Ereignisses, die ein Steuerelement ausgelöst hat, ist es eventuell sinnvoll, die Gültigkeit der Daten aus dem Formular zu prüfen. ASP.NET kennt so genannte Kontroll-Steuerelemente, die Eingabewerte nach bestimmten Kriterien untersuchen. In der Folge dieses Tests wird die Eigenschaft `IsValid` des Steuerelements und als Summe aller Steuerelemente der Seite (`Page.IsValid`) gesetzt. Innerhalb der Ereignisbehandlungsmethoden ist ein guter Zeitpunkt auf das Ergebnis der Gültigkeitsprüfung zu reagieren.

Bleibt zuletzt noch ein Blick auf den Aufräumprozess. Bevor die Seite endgültig aus dem Speicher verschwindet – `Page_Unload` wird zuvor aufgerufen – ist ein guter Zeitpunkt, beispielsweise nicht persistente Verbindungen zu einer Datenbank zu schließen.

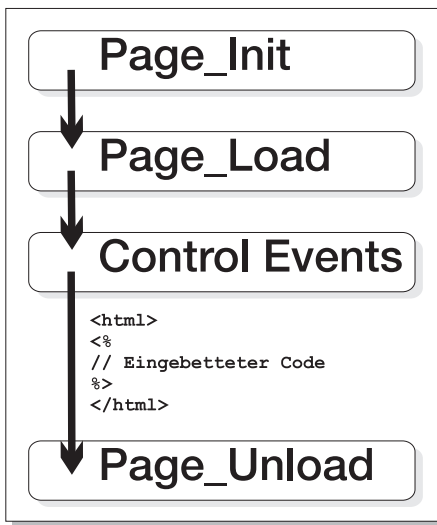


Abbildung 1.18: Ereignisfolge eines ASP.NET-Seitenzyklus

Sonstige Ereignisse

Neben diesen, in der normalen Programmierung häufig benötigten Ereignissen gibt es weitere, die nur unter bestimmten Bedingungen von Interesse sind, beispielsweise `PreRender` vor der Analyse der Seite oder `Error` beim Auftreten von Fehlern.

PreRender
Error

Programmierung der Ereignisbehandlungsmethoden

Direkte Programmierung

Alle Ereignisbehandlungsmethoden sind als *Public* und als Prozedur kennzeichnen. Sie müssen aber nicht alle deklarieren, sondern nur die, die sie wirklich benötigen. Am häufigsten dürfte `Page_Load` zum Einsatz kommen:

```
Public Sub Page_Load (sender As Object, e As EventArgs)
```

Was hat das zu bedeuten? Hier wird für das Ereignis »Page_Load« eine Ereignisbehandlungsmethode definiert. Diese überschreibt die vom System vorgegebene, damit dort eigene Aufgaben ausgeführt werden können. Übergeben wird der Methode das aufrufende Objekt im ersten Parameter, allgemein als *sender* bezeichnet (tatsächlich ist der Name frei wählbar) und immer vom Typ `Object`. Der zweite Parameter enthält das so genannte Ereignisargument, das bei den Standardereignissen der Seite vom Typ `EventArgs` ist. Auch hier wird häufig als Name *e* verwendet, aber auch dies ist frei wählbar.

Programmierung im Visual Studio .NET

Wenn Sie Code mit dem Designer des Visual Studio .NET erzeugen, wird das Resultat etwas anders aussehen. Hier wird die Ereignisbehandlungsmethode immer als *private* gekennzeichnet. Sie ist damit für den direkten Aufruf nicht sichtbar. Der Designer fügt dafür folgenden Code in Ihre Code-Datei ein, damit die Ausführung dennoch gelingt:

```
Public Class firsttest
    Inherits System.Web.UI.Page

#Region " Vom Web Form Designer generierter Code "
    'Dieser Aufruf ist für den Web Form-Designer erforderlich.
    <System.Diagnostics.DebuggerStepThrough(>
    Private Sub InitializeComponent()

        End Sub

    Private Sub Page_Init(ByVal sender As System.Object,
        ByVal e As System.EventArgs)
        Handles MyBase.Init
        'CODEGEN: Diese Methode ist für den Web Form-Designer
        erforderlich
        'Verwenden Sie nicht den Code-Editor zur Bearbeitung.
        InitializeComponent()
    End Sub

#End Region
```



```
Private Sub Page_Load(ByVal sender As System.Object,
                    ByVal e As System.EventArgs)
    Handles MyBase.Load
    ' Hier Benutzercode zur Seiteninitialisierung einfügen
End Sub
```

End Class

Wie funktioniert der automatisch generierte Code nun? Generell wird am Anfang der Seitenverarbeitung die Methode `OnInit` aufgerufen. Hier erfolgt der Aufruf von `InitializeComponent`, die vom Designer eingefügte Methode. Dort werden die mit dem Web-Form-Designer erzeugten Elemente verarbeitet.

ASP.NET verfügt über ein ereignisgesteuertes Modell. Das bedeutet, dass zu bestimmten Zeitpunkten im Verarbeitungszyklus der Seite bestimmte Prozeduren aufgerufen werden. Damit diese sich dafür zuständig fühlen, werden sie durch das Schlüsselwort *Handles* mit dem entsprechenden Ereignis verknüpft. Bei der Methode `Page_Load` sieht das dann folgendermaßen aus (unter Fortlassung der Parameter):

```
Private Sub Page_Load(...) Handles MyBase.Load
```

Es ist an dieser Stelle nicht notwendig, das System der Ereignisverarbeitung in allen Varianten zu verstehen. Denken Sie nur daran, den Inhalt der vom Designer erzeugten Zeilen nicht zu verändern, denn er wird beim erneuten Einsatz der entsprechenden Funktionen den Inhalt überschreiben.



1.5 Programmierprinzipien

Dieser Abschnitt führt in die grundlegenden Prinzipien von ASP.NET ein. Die ausführliche Betrachtung folgt dann im Teil B. Diese Einführung dient der systematischen Vorbereitung der nötigen Fertigkeiten in den folgenden Kapiteln, in denen die Grundlagen der ASP.NET-Programmierung vermittelt werden.

1.5.1 Basiselemente einer Applikation: Web Forms

»Web Forms« ist ein Oberbegriff für die Benutzerschnittstelle von Webseiten. Web Forms entstehen, in dem eine ASP.NET-Seite er-

stellt wird, mit der Benutzer interagieren können. Dabei geht es um mehr als reine Formulare. Jedes Element, dass HTML kennt, kann serverseitig verarbeitet oder erstellt werden. Und nicht nur das – ein Steuerelement kann auch umfangreichen HTML-Code, bestehend aus viele Tags, erzeugen.

HTML Server- Steuerelemente

Der Begriff »Web Forms« erinnert an den vergleichbaren Begriff aus der Windows-Programmierung, »Win Forms«. Die Prinzipien der ereignisgesteuerten Programmierung sind ähnlich und in einigen Fällen gleich. Es geht in beiden Fällen um die Schaffung der Benutzerschnittstelle. Im folgenden wird allgemein von Formularen gesprochen, weil dies der gängigste Begriff ist. Dabei ist es zwar oft der Fall, aber nicht zwingend erforderlich, dass der HTML-Code ein HTML-Formular enthält, dass das Tag `<form>` verwendet. Lassen Sie sich davon nicht irritieren. Der programmtechnische Zugriff auf HTML ist am einfachsten und in vielen Fällen auch ausreichend für den Aufbau von Formularen. Die Abbildung im Framework erfolgt mit den *HTML Server-Steuerelementen* (HTML Server Controls). Praktisch gibt es für jedes Steuerelement dieser Klasse eine direkte und eindeutige Entsprechung in HTML. HTML-Elemente werden programmtechnisch verfügbar gemacht, in dem sie mit dem Attribut `runat="server"` versehen werden. Dies ist unabhängig davon möglich, ob eine entsprechende Klasse existiert oder nicht, weil .NET zwei generischen Klassen besitzt, die alternativ verwendet werden können. HTML Server-Steuerelemente werden ausführlich im Abschnitt 6.2, »HTML Server-Steuerelemente (HTML Server Controls)« ab Seite 504 behandelt. Sowohl den Zugriff auf einzelne HTML-Tags als auch auf Sammlungen mehrerer Elemente erlauben die *Web Server-Steuerelemente* (Web Server Controls). Interessant sind diese Steuerelemente, weil sie eine sehr einfache programmtechnische Verwaltung in einem Objekt auch dann erlauben, wenn zur Darstellung viele HTML-Tags notwendig sind. Einige sind aber auch nur für ein Element zuständig, sodass sich Überschneidungen mit den einfacheren HTML Server-Steuerelementen ergeben. Sie können eigene Web Server-Steuerelemente entwerfen – dies sind die Benutzer-Steuerelemente (User Controls). Web Server-Steuerelemente werden im Abschnitt 6.4, »Web Server-Steuerelemente (Web Server Controls)« ab Seite 535 behandelt. Sollen die Eingaben der Benutzer vom Programm analysiert werden, bieten sich die *Kontroll-Steuerelemente* (Validation Controls) an. Diese erlauben sowohl eine client- als auch serverseitige Kontrolle. Für den Einsatz im Browser liefert ASP.NET browserunabhängige JavaScript-

Web Server- Steuerelemente

Kontroll- Steuerelemente

Bibliotheken mit Kontroll-Steuer-elemente werden ausführlich im Abschnitt 6.7, »Kontroll-Steuer-elemente (Validation Controls)« ab Seite 616 behandelt. *Benutzer-Steuer-elemente* (User Controls) erleichtern den Entwurf modularisierter Webformulare. Sie können häufig benutzte HTML-Elemente zusammenfassen und mehrfach in Seiten einbinden. Die interne Darstellung als Objekt erleichtert den Zugriff vom Programm aus. Eine Anwendung ist auch der Entwurf von Bibliotheken mit Steuer-elementen für spezielle Zwecke, beispielsweise mobile Clients. Benutzer-Steuer-elemente werden im Abschnitt 7.2, »Modularer Code mit Benutzer-Steuer-elementen« ab Seite 647 kurz vorgestellt. Für die professionelle Programmierung geeignet sind *Kunden-Steuer-elemente* (Custom Controls). Dies sind komplexe Definitionen eigener Steuer-elemente, die als Assembly vorliegen und bei denen nur übersetzter Code in Form einer DLL weitergegeben wird. Eine Einführung in die Programmierprinzipien finden Sie in Abschnitt 7.6, »Kundenspezifische Steuer-elemente (Custom Controls)« ab Seite 706.

Benutzer-Steuer-elemente

Kunden-Steuer-elemente

1.5.2 Code Behind

Bei der Arbeit mit größeren Applikationen ist die Verbindung von Code und HTML in einer Datei nicht immer praktikabel. Eine strengere Trennung – Code in einer Datei und HTML in einer anderen – ist besser geeignet. Wie am Anfang schon gezeigt wurde, stellt ASP.NET einige Automatismen bereit, die das Erstellen einfacher Seiten stark erleichtern. So erzeugte Visual Studio .NET den Code gleich in der für die Ereignisverarbeitung in hinterlegten Code-Dateien erforderlich Form.

Trennung von Code und Design

In diesem Buch wird fast durchgehend Code Behind verwendet, um diesen hervorragenden und wichtigen Programmierstil durchsetzen zu helfen. Wenn Sie es am Anfang auch als einfacher empfinden, den Code in die Seite zu schreiben, werden Sie dies möglicherweise als lästig empfinden. Es ist aber spätestens beim ersten richtigen Projekt die einzig vernünftige und beherrschbare Codierungsform. Je eher Sie sich daran gewöhnen, umso erfolgreicher und effizienter werden Sie später arbeiten.



Hinterlegten Code erzeugen

Wenn der Code getrennt wird, muss eine reguläre Klasse erstellt werden. Die Ablage erfolgt in einer cs-Datei, die der Compiler

übersetzen muss. Da Sie in dieser Klasse Zugriff auf die HTML-Elemente haben wollen, müssen diese Objekte verfügbar gemacht werden. Dazu stellt ASP.NET die *aspx*-Seite als Klasse zur Verfügung. Diese trägt den Namen `Page`. Die Definition ihrer eigenen Klasse würde dann folgendermaßen aussehen:

```
Public Class MyClass
    Inherits System.Web.UI.Page
```

Weiterhin ist zu beachten, dass ASP.NET den Import der wichtigsten Namensräume des .NET-Frameworks für Sie erledigt. Das ist nicht mehr der Fall, wenn Sie hinterlegten Code verwenden und ohne Visual Studio .NET arbeiten¹. Sie müssen dann selbst die nötigen Anweisungen aufschreiben. In VB.NET erfolgt dies mit der Anweisung *Imports*:

```
Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls
Imports System.Collections
```

Die gezeigten Namensräume sollten für die meisten Aufgaben mit ASP.NET ausreichen. Alle anderen bindet auch ASP.NET nicht automatisch ein, sodass sich hier bei hinterlegtem Code keine Änderungen ergeben.

Hinterlegten Code verwenden

Im nächsten Schritt muss eine Methode gefunden werden, den externen Code in der *aspx*-Seite zu verwenden. Dazu wird die Seitendirektive `@Page` eingesetzt:

```
<% @Page Inherits="classname" src="path/file.vb" %>
```

Sie können also die externe Code-Datei und die verwendete Klasse getrennt erreichen. Dadurch ist es möglich und oft sinnvoll, mehrere Klassen in einer Datei unterzubringen. Beachten Sie bei der Angabe der Klasse, dass auch der Namensraum erforderlich ist, wenn einer verwendet wurde. Die Beispiele im Buch liegen unter »Addison.VBNet«, in diesem Kapitel ist noch eine weitere

¹ Visual Studio .NET bindet bei VB.NET-Projekten die wichtigsten Namensräume auch im hinterlegten Code automatisch mit ein.

Ebene mit dem Namen »WebForm« eingesetzt worden. Dies dient nur der Organisation.

Interessant ist die Möglichkeit, die *vb*-Datei vorab zu kompilieren und dann als Binärdatei bereit zu stellen. Liegt diese im Verzeichnis *bin*, reicht die Angabe der benötigten Klasse:

```
<% @Page Inherits="classname" %>
```

Beachten Sie, dass die Verwendung von `Response.Write` in externen Klassen wenig sinnvoll ist. Sie sollten daran denken, dass die Ausgabe im Teil `Page_Load` vor der Erstellung der Seite erfolgt. Ausgaben sollten besser mit HTML Server-Steuerelemente oder Web Server-Steuerelemente erfolgen.

Ein Beispiel zeigt, wie hinterlegter Code praktisch verwendet wird. Zuerst eine *aspx*-Datei, die nur noch HTML und Direktiven enthält:

```
<%@ Page src="codebehind.aspx.vb"
    Inherits="Addison.VBNet.WebForm.CodeBehind"
    Language="vb"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
    Transitional//EN">
<html>
    <head>
        <title>Schaltfl&auml;che gestalten</title>
    </head>
    <body>
        <h1>Schaltfl&auml;che gestalten:</h1>
        <form runat="server" >
            <input type="button" id="btn"
                runat="server" onClick="clickBtn"/>
        </form>
        <div id="Bestaetigung" runat="server"/>
    </body>
</html>
```

Listing 1.4: Verwendung externen Codes (codebehind.aspx)

Interessant ist nur die erste Zeile, die die zu verwendende Klasse und die Quelldatei festlegt. Die eigentliche Arbeit steckt in der *vb*-Datei:

```
Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls
```

```
Imports System.Collections

Namespace Addison.VBNet.WebForm
    Public Class CodeBehind
        Inherits System.Web.UI.Page

        Protected WithEvents btn As HtmlInputButton
        Protected WithEvents Bestaetigung As HtmlGenericControl

        Private Sub Page_Load(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
            Handles MyBase.Load

            btn.Value = "Ich bin ein schicker Button"
            btn.Style("font-family") = "Arial"
            btn.Style("color") = "green"
            btn.Style("font-size") = "24pt"
        End Sub

        Public Sub clickBtn(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs) _
            Handles btn.ServerClick
            Bestaetigung.InnerHtml = "Meine Gestaltung basiert _
                                     auf:<br />"

            Dim keys As IEnumerator = btn.Style.Keys.GetEnumerator()
            Dim key As String
            While keys.MoveNext()
                key = keys.Current.ToString()
                Bestaetigung.InnerHtml += key + "=" + " " _
                                         + btn.Style(key) + "<br />"
            End While
        End Sub

    End Class
End Namespace
```

Listing 1.5: Code Behind-Datei (codebehind.aspx.vb)

In hinterlegtem Code müssen Sie alle Deklarationen – einschließlich der Namensräume – selbst erledigen. Die ersten Zeilen bestehen praktisch immer aus den entsprechenden Imports-Anweisungen. Danach folgt die Klasse, im Beispiel mit dem Namen CodeBehind. Diese Klasse muss von Page abgeleitet werden. Page ist die aktuelle Seite und durch die Vererbung wird der Zugriff auf Steuerelemente und Eigenschaften überhaupt erst möglich. Innerhalb der Klassen stehen die Methoden Page_Load, Page_Init usw. natürlich unverändert zur Verfügung. Wollen Sie hier Code aus-

führen, überladen Sie die entsprechende Methode, wie im Beispiel für `Page_Load` gezeigt.

Damit aus der Seite heraus auf die instanziierten Seitenobjekte zugegriffen werden kann, wie das beispielsweise für die Schaltfläche `btn` gezeigt wird, müssen diese als `Protected` (oder `Public`) deklariert werden:

```
Protected WithEvents btn As HtmlInputButton
```

Außerdem ist natürlich der richtige Datentyp anzugeben. Der Zugriff über globale Klassen, wie `HtmlControl` statt `HtmlInputControl` funktioniert zwar, schränkt aber die verfügbaren Eigenschaften und Methoden entsprechend ein.

Datentypen

Das Schlüsselwort `WithEvents` macht die Ereignisse des Objekts – hier also der Schaltfläche – im Code verfügbar und erlaubt die Bindung an die Ereignisbehandlungsmethode mit `Handles`. Dieses Verfahren wird noch detailliert erläutert, es ist hier nicht notwendig, dies vollständig zu verstehen. Das Beispiel ist jedoch für viele andere exemplarisch. Wer aus der Windows-Programmierung mit VB 6 kommt, kennt das Prinzip. Das eigentlich revolutionäre an ASP.NET ist die Übertragung des klassischen Windows-Programmierstiles auf die Webentwicklung, ohne dabei die Kompatibilität zu HTML zu verletzen.

Ereignisse

Code Behind im Visual Studio .NET

Der Designer nutzt andere Attribute um hinterlegten Code zu binden. Erzeugt wird in etwa folgende Zeile:

```
<%@ Page language="vb"
    Codebehind="login.aspx.vb"
    AutoEventWireup="false"
    Inherits="Addison.VBNet.Security.login" %>
```

Hier wird die Datei selbst mit dem Attribut `Codebehind` angegeben. Das Attribut `Inherits` gibt die Klasse an, die verwendet werden soll. Diese Klasse muss von `Page` erben.

1.6 Allgemeine Sicherheit im Framework

Sicherheit wird im Framework groß geschrieben. Mit vielfältigen Optionen kann eine rundum sichere Website erstellt werden. Da-

mit ist .NET weit besser für den Betrieb bei einem Massenhoster geeignet, als die Vorgängerversion ASP.

1.6.1 Sicherheitskonzepte

Um für einen Webserver für optimale Sicherheit zu sorgen, sollten Sie sich zuerst mit den prinzipiellen Sicherheitskonzepten auseinander setzen:

► Softwaresicherheit

Auch wenn es sich banal anhört: Das Aufspielen der neuesten Service Packs; Sicherheits-Updates und die Umsetzung von Sicherheitshinweisen des Softwareherstellers ist eine Pflichtübung.

► Authentifizierung

Hierunter wird die Erkennung eines Benutzers verstanden (»Wer bin ich?«). Falls es sich nicht um einen anonymen Zugang handelt, wird sich der Benutzer in der Regel mit Name und Kennwort identifizieren. Das gilt auch – wenn auch über Umwege – für die Nutzung von Diensten wie Passport. Ansonsten kann die Authentifizierung über den IIS und die integrierten Sicherheitsmodelle von Windows 2000 bzw. .NET-Server erfolgen.

► Autorisierung

Nachdem sich ein Benutzer authentifiziert hat, ist es Aufgabe der Autorisierung festzustellen, welche konkreten Rechte er hat. Hiermit wird also festgelegt, welche Ressourcen er lesen, ausführen oder schreiben darf. Damit das in der Praxis nicht zu kompliziert wird, bieten der IIS und ASP.NET einige vordefinierte Konten, die standardmäßig zum Einsatz gelangen.

► Impersonifizierung

Ist ein Benutzer identifiziert, kann ein vorher unter einem Standardkonto ablaufender Prozess die Identität des Benutzers annehmen. Dieser Vorgang wird als Impersonifizierung bezeichnet. Damit wird vor allem die Rechtevergabe erleichtert, ohne dass eine Schwachstelle in der Sicherheitskette aufgerissen wird.

Ausführliche Informationen zur Einrichtung der Sicherheitsbedingungen in ASP.NET finden Sie im Abschnitt 10.5, »Sicherheit« ab Seite 973.

I.7 Hinweise zum Stil – Codekonventionen

Auch Programmierung hat Stil. Microsoft hatte schon immer Richtlinien herausgegeben, die auf den Stil des Codes hinwiesen. Dies betrifft die Benennung von Variablen, Konstanten aber auch Methoden und Klassen. Mit .NET wurde hier einiges strenger und auch anders als bisher geregelt. Freilich ist dies nur eine Empfehlung und für die Lauffähigkeit nicht erforderlich. Ein guter Stil ist dennoch hilfreich, in größeren Projekten nicht die Übersicht zu verlieren.

Die hier beschriebenen Codekonventionen basieren auf den Microsoft-Empfehlungen und dienen als Grundlage für das Framework selbst.

Die erste Frage ist immer wieder: Deutsch oder Englisch? Prinzipiell ist es egal, solange Sie nicht mitten im Programm die Sprache wechseln. Deutsche Namen sind dann zu bevorzugen, wenn der Leser der Codes Deutsch spricht. Es gilt aber: Umlaute sind tabu! In allen anderen Fällen verwenden Sie Englisch, weil dies mehr Leute lesen können und die Namen teilweise kürzer sind.

**Deutsch oder
Englisch**

I.7.1 Schreibweise von Namen im Code

Grundsätzlich werden folgende Schreibstile für die Benennung von Bezeichnern gebraucht:

► *Pascal-Schreibweise*

Diese Schreibweise nutzen Großbuchstaben zur Trennung von Wortteilen, beispielsweise `CodeKonvention`. Das Wort beginnt außerdem immer mit einem Großbuchstaben. Diese Schreibweise ist die Standardform in .NET und wird für Klassen, Namensräume, Eigenschaften und öffentliche Variablen verwendet.

► *Camel-Schreibweise*

Bei dieser Schreibweise wird ähnlich wie bei der Pascal-Schreibweise ein Großbuchstabe zur Trennung der Wortteile eingesetzt. Der erste Buchstabe ist jedoch immer klein. Dadurch entsteht in der Mitte eine Art Höcker, was zur Namensvergabe führte: `blueHeader`.

Einsetzen sollten Sie diese Schreibweise für Parameter, die einer Methode übergeben werden.

► *Großbuchstaben*

Reine Großbuchstabenfolgen sollten nur in Ausnahmefällen verwendet werden. Wenn Sie viel mit Konstanten arbeiten, ist eine Kennzeichnung manchmal hilfreich: `START`.

Wenn ein Bezeichner nur aus zwei Buchstaben besteht, können Großbuchstaben die Lesbarkeit steigern: `System.IO`.

Generell ist Schnittstellennamen ein großes »I« voranzustellen (von Interface): `ICollection`.

► *Kleinbuchstaben*

Lokale Variablen, die einen geringen Sichtbereich haben oder temporär existieren, Schleifenvariablen und andere Hilfsvariablen sollten mit Kleinbuchstaben gekennzeichnet werden: `i`, `temp`.

Hinweise zur Groß- und Kleinschreibung

.NET beachtet grundsätzlich die Groß- und Kleinschreibung. VB.NET korrigiert dies insofern, als dass der Compiler dies erkennt und versucht, verschieden groß geschriebene Namen gleich zu behandeln. Achten Sie besser auf die richtige Schreibweise, als einem Automatismus die Wahl zu überlassen.

Über den Umgang mit Abkürzungen

Vermeiden Sie Abkürzungen oder kürzen Sie bei Bedarf so ab, dass in der Computertechnik anerkannte Begriffe erkennbare bleiben. Schreiben Sie also statt `MyBtn` besser `MyButton`. Wenn Abkürzungen oder Akronyme mehr als zwei Zeichen haben, verwenden Sie die Pascal-Schreibweise: `HttpPortScanner`.

1.7.2 Hinweise zur Benennung von Standardtypen

Wenn Sie Namensräume, Klassen usw. verwenden und Ihre Programme weitergeben, ist es für den Empfänger leichter, diese zu verwenden, wenn Sie sich auch hier an bestimmten Regeln halten.

Benennung von Namensräumen

Namensräume sollten hierarchisch aufgebaut werden. Gehen Sie nach folgendem Muster vor:

```
Firma.TechnologieOderProjekt.Funktion.Teilfunktion
```

Die Teile `Funktion` und `Teilfunktion` sind optional. In diesem Buch wird beispielsweise der Namensraum »Addison« für die oberste Ebene verwendet, gefolgt von der Einteilung nach der Programmiersprache und dem globalen Thema:

```
Addison.VBNet.Data
```

Benennung von Klassen und Schnittstellen

Hier gilt zuerst die Pascal-Schreibweise. Verwenden Sie niemals den Unterstrich zur Trennung von Wortteilen. Verwenden Sie niemals allgemeine Präfixe, wie sie beispielsweise bei Variablen eingesetzt werden können. Falsch wäre danach: `Konverter_Klasse`, `CDateiKonverter`. Richtig ist: `KonverterKlasse`, `DateiKonverter`.

Klassen sollten immer mit Substantiven benannt werden. Beachten Sie bei der Wahl des Namens, dass Schnittstellen ähnlichen Konventionen unterliegen und sich nur durch das führende »I« unterscheiden. Ein »I« als erster Buchstabe ist nur zulässig, wenn er Teil eines vollständigen Wortes ist: `IndexSpeicher`.

Schnittstellen stellen Sie das »I« voran; dies steht für »Interface«.

Benennung von Aufzählungen

Namen der Bestandteile von Aufzählungen bezeichnen Zustände, werden also mit Substantiven gekennzeichnet. Verwenden Sie immer den Singular, außer bei Bitfeldern. Bitfelder erlauben die Kombination mehrere Werte, hier ist der Plural angebrachter.

Benennung von Variablen und Feldern

Variablen und Felder sollten immer einen selbstbeschreibenden Namen tragen, der auf den Inhalt hinweist: `SchalterFarbe`, `AnfangsDatum`. Wenn Variablen Zuständen speichern, kann auch dies ein guter Name sein: `HasSaved`. Variablen sollten nach Möglichkeit mit Substantiven benannt und mit Adjektiven ergänzt werden.

Dagegen sollten Sie nicht den Typ als Namen verwenden. Falsch wäre danach: `doubleValue`. Dies ist nicht aussagekräftig. Verwenden Sie viele verschiedene Typen und ist die Unterscheidung für den Leser von großer Bedeutung, kann der Typ in seiner allgemeinsten Form vorangestellt werden: `dSchalterWert`, für einen *Double*-Typ. Verwenden Sie nicht die voll ausgeschriebenen Typen der verwendeten Programmiersprache, weil diese im Disassembler in Systemtypen verwandelt werden und dann solche Bezeichnungen irreführend sind.

Wo kommen die Typpräfixe her?

In diesem Buch werden Variablen sehr oft mit einem einfachen Typpräfix bezeichnet. Dies hat den Vorteil, dass Programme für Anfänger besser lesbar sind. In der Praxis sollten Sie diesen speziellen didaktischen Stil nicht kommentarlos übernehmen.

Die Notation mit Typpräfix wird als ungarische Notation bezeichnet. Der Name geht auf Charles Simonyi, der ungarischer Nationalität ist, zurück. Simonyi kam 1981 von Xerox zu Microsoft und ist dort Chefarchitekt im Forschungsbereich.

Bei statischen Feldern sollten Sie immer den Typ davor setzen. Dies dient der Erkennbarkeit über weite Strecken, weil diese Felder nicht instanziiert werden.

Benennung von Ereignissen

Wenn Sie Ereignisbehandlungsmethoden sehr oft verwenden, bietet sich eine Kennzeichnung an, wie sie auch von Visual Studio .NET vorgenommen wird: `ObjektName_Auslöser`, also eine Trennung von Objekt und Ereignisname mittels Unterstrich.

Für einen Mausklick sieht das dann folgendermaßen aus:

```
MyButton_Click
```

Dabei ist es üblich – und dies ist die Ausnahme – bei einer deutschen Benennung der Steuerelemente den ursprünglichen Namen des Ereignisses zu belassen: `Schalter_Click` ist besser als `Schalter_Klick`.

I.7.3 Hinweise für Web-Programmierer

Sicher hat jeder Programmierer seinen Stil und kennt viele mögliche Varianten für eine saubere Programmierung. Anfänger sollten sich jedoch nicht von fremden stilistischen Formen leiten lassen, ohne den Sinn zu erkennen, der hinter der einen oder anderen Schreibweise steckt. Die besonderen Anforderungen des Webs sind auch nicht jedem Profi völlig vertraut. Die folgenden Tipps zeigen, worauf es ankommt.

Eine optisch ansprechende Codierung wird erreicht, wenn Sie Folgendes beachten:

- ▶ Code-Konventionen einhalten
- ▶ Formatierung und Strukturierung beachten

**Werden Sie ein
»guter« Programmier-
er**

Trennen Sie Code vom Design

Die Trennung hilft, leicht beherrschbare Programmteile zu erhalten. Die Wiederverwendbarkeit des Codes wird gesteigert. Sie können mit Gestaltungswerkzeugen arbeiten, die eingeschlossenen Code nicht verstehen. In großen Teams können Sie das Design von Designern erledigen lassen und die reine Programmierarbeit Programmierern übergeben.

ASP.NET unterstützt Sie dabei mit einer ganze Reihe von Techniken, von denen so genannter hinterlegter Code (Code Behind) die wichtigste ist. Diese Technik wird deshalb in diesem Buch dominant eingesetzt.



Zur Wiederverwendbarkeit von Design und Code

Wenn Sie etwas ändern möchten, müssen Sie es nur an einer Stelle tun. Konsistente Gestaltungsmerkmale und Bedienerführung erleichtern den Nutzern die Navigation. Behandeln Sie Design-Elemente als Komponenten, die Sie immer wieder verwenden.

ASP.NET unterstützt Ihre Bemühungen in diese Richtung mit Benutzer- und Kunden-Steuerelementen. Setzen Sie sich damit auseinander und versuchen Sie, Steuerelemente generell zu verwenden, auch wenn die direkte HTML-Programmierung am Anfang noch einfacher erscheint.



Verwenden Sie keine komplexen URLs

URLs mit vielen Parametern zeigen Hackern die innere Struktur Ihrer Applikation und legen Angriffspunkte offen. Nutzer werden vielleicht auch Ihre Seite als Lesezeichen ablegen. Wenn Sie die innere Struktur ändern, verlieren Sie Leser, die über die Lesezeichenverwaltung des Browsers auf Ihre Site zugreifen.



ASP.NET unterstützt Ihre Bemühungen in exzellenter Weise durch Sitzungs- und Applikationsvariablen, Cookies und den flexiblen Zugriff auf Datenbanken.

Personalisieren Sie Ihre Seiten

Sie steigern damit die Verbindung zum Nutzer und animieren ihn, wieder zu kommen. Wer Fragen doppelt stellt, gilt als dumm. Also lassen Sie den Benutzer nicht immer wieder seine Präferenzen angeben, sondern merken Sie sich diese.

Verwenden Sie mehrsprachige Seiten, wenn Sie ein internationales Publikum ansprechen und Ressourcen, um angepasste Layouts vorzuhalten.



ASP.NET unterstützt Sie auf Basis des .NET-Frameworks in exzellenter Weise beim Aufbau lokalisierter oder ressourcenbasierter Anwendungen.

Unterstützen Sie Proxys und Caches

Nicht jeder Nutzer hat T-DSL oder 2-Mbit-Festverbindungen. Wenn der Browser-Cache unterstützt wird, verkürzen Sie aktiv die Ladezeiten. Dynamisieren Sie nur die Seiten, bei denen es wirklich sinnvoll ist.



ASP.NET verfügt über eine ganze Reihe von Techniken, mit denen Daten zwischengespeichert werden können oder die der Kontrolle von Zwischenspeichern (Caches) dienen. Setzen Sie sich damit aktiv auseinander, um professionelle Webapplikationen zu erhalten.

2 Visual Studio .NET

Für die professionelle Entwicklung von ASP.NET-Anwendungen ist das Visual Studio .NET von Microsoft die erste Wahl. In diesem Kapitel wird es vorgestellt und gezeigt, wie Sie mit denkbar wenig Aufwand erste Projekte schnell und produktiv damit erstellen können.

2.1 Einführung

Das Visual Studio .NET kann als Nachfolger des Visual Studio 6 betrachtet werden. Kennen Sie diese Version, werden Sie allerdings schnell feststellen, dass die Versions-Bezeichnung .NET (anstelle einer Version 7, die es aber dennoch intern führt) die logische Konsequenz darstellt - ist das Studio doch voll und ganz für die professionelle Entwicklung von .NET-Projekte aller Art ausgelegt.

2.1.1 Ausführungen von Visual Studio .NET

Visual Studio .NET gibt es in verschiedenen Ausführungen, auch *Editionen* genannt. Die derzeit angebotenen werden nachfolgend kurz vorgestellt:

- ▶ *Visual C#.NET Standard*
- ▶ *Visual Basic .NET Standard*
- ▶ *Visual C++.NET Standard*

Diese Versionen umfassen jeweils nur eine einzige Programmiersprache und eignen sich nicht zuletzt wegen des günstigen Preises gut für Einsteiger oder für Entwickler, die sich auf eine Sprache festgelegt haben. Allerdings ist die Entwicklungsumgebung nicht mit der des Visual Studios zu vergleichen.

Nur eine Programmiersprache

Voller Funktions- umfang	► <i>Visual Studio .NET Professional Edition</i> In diesem Paket finden Sie die wichtigsten für .NET verfügbaren Sprachen Visual Basic, C# und C++ neben der leistungsstarken Entwicklungsumgebung (IDE – <i>Integrated Development Environment</i>). Für die Entwicklung von ASP.NET-Anwendungen bedeutsam sind zusätzliche Komponenten wie HTML- und XML-Designer, MSDE oder die Visual Database Tools. Für die Fehlersuche steht außerdem ein leistungsfähiger Debugger zur Verfügung.
Für Profis und Teams	► <i>Visual Studio .NET Enterprise Developer Edition</i> Zusätzlich zur Professional Edition sind hier weitere Werkzeuge für die Entwicklung im Team enthalten. Desweiteren umfasst das Paket die folgenden Microsoft-Serverprodukte als Vollversionen, die zu Entwicklungs- und Testzwecken eingesetzt werden können: ► Windows 2000 Advanced Server ► SQL Server 2000 ► Commerce Server 2002 ► Host Integration Server 2000 ► Exchange Server 2000
Für den professionellen Softwareentwurf	► <i>Visual Studio .NET Enterprise Architect Edition</i> Als High-End Entwicklungssystem für .NET verfügt diese Edition neben allen Merkmalen und Komponenten der Enterprise Developer Edition weitere, welche die Erstellung von Unternehmensanwendungen im großen Stil vereinfachen. Hinzu kommen Visio-basierte Modellierungskomponenten für UML, mit deren Hilfe grafisch basiert komplexe Datenbank- und anderen Anwendungen erstellt werden können. Darüber hinaus umfasst das Paket neben den oben genannten Server-Vollversionen den Microsoft BizTalk-Server.
Beschränkung auf Professional Edition	Im vorliegenden Buch wird bewusst nur die Professional Edition eingesetzt – ohne auf diese wirklich umfassend und bis in alle Tiefen eingehen zu können. Dies würde sonst den Umfang hoffnungslos sprengen. Stattdessen wird versucht, in den nachfolgenden Abschnitten zu zeigen, wie Sie mit Hilfe des Visual Studios und seiner Hilfsmittel schnell und produktiv ASP.NET-Anwendungen unter Nutzung der Sprache VB.NET entwickeln können.

2.1.2 Installation von Visual Studio .NET

Die Installation von Visual Studio .NET Professional gestaltet sich dank eines ausgeklügelten Installationsprogrammes relativ problemlos. Nachfolgend finden Sie eine Schritt-für-Schritt-Anleitung, die insbesondere Anfängern über die eine oder andere Hürde helfen sollen. Gezeigt wird die Installation auf einem Windows XP Professional-System.

Voraussetzungen

Vor der Installation sollten Sie prüfen, ob Ihr Entwicklungssystem einige Voraussetzungen erfüllt. Visual Studio stellt insbesondere an die zu verwendende Hardware recht hohe Anforderungen:

Hohe Hardware-Anforderungen

► *Prozessor und RAM*

Ihr System sollte mindestens in der 500 MHz-Klasse spielen. Fast noch wichtiger ist eine ausreichende Ausstattung mit Hauptspeicher. 256 MByte sollte die untere Grenze darstellen – auch wenn es bereits mit 128 MByte funktioniert.

► *Freie Festplattenkapazität*

Voll installiert belegt das Visual Studio – einschließlich der MSDN-Bibliothek – etwa 2 GByte auf der Festplatte. Dieser Platz wird benötigt, wenn Sie Zugriff auf alle Sprachen und Komponenten sowie die ausgefeilte Online-Dokumentation sicherstellen wollen.

Wenn Sie ein schlankes und schnelles Entwicklungssystem für unterwegs auf einem vielleicht nicht mehr ganz aktuellen Notebook suchen, wird das Visual Studio also nicht unbedingt die erste Wahl sein können. Hier empfiehlt sich dann eher ein einfacherer Editor.

Für die Entwicklung von ASP.NET-Anwendungen benötigen Sie lokal den Internet Information Server ab Version 5. Damit schränkt sich die Auswahl an einsetzbaren Betriebssystemen im Grunde auf Windows XP Professional und Windows 2000 (Professional oder eine der Server-Versionen) ein.

Software-Voraussetzungen

Installation durchführen

Die Installation des Visual Studios läuft in drei Schritten ab. Diese erreichen Sie über das Startfenster des Setups, welches nach Einlegen der DVD automatisch erscheint.



Abbildung 2.1: Installations-Startfenster mit der Vorgabe der Schrittfolge

1. Prüfung und Installation wichtiger Windows-Komponenten

Für die Entwicklung von ASP.NET-Anwendungen wird geprüft, ob der IIS (inkl. der Frontpage-Servererweiterungen) auf dem System verfügbar ist. Stellt das Installationsprogramm hier fehlende Komponenten fest, werden Sie über eine ausführliche Anleitung zur korrekten Nachinstallation geführt. Erst nach deren Durchführung können Sie mit Punkt 2 fortfahren.

2. Installation von Visual Studio .NET

Im zweiten Schritt wird das Visual Studio selbst installiert. Dabei können Sie Einfluss auf den Installations-Ort und -Umfang nehmen. Beachten Sie, dass als Installations-Komponente auch das Framework SDK mit angeboten wird. Auch wenn Sie bereits das kleinere *.NET Framework Redistributable* installiert haben sollten, sollten Sie das SDK ausgewählt lassen. Vor allem die mitgelieferte Dokumentation erweist sich während der Arbeit als sehr wertvoll.

3. Installation von Service-Releases

Abschließend können Sie direkt über den Setup-Startbildschirm nach aktuellen Services-Packs beziehungsweise -Releases suchen. Dies wird erleichtert, wenn Sie das über eine aktive Internet-Verbindung durchführen.

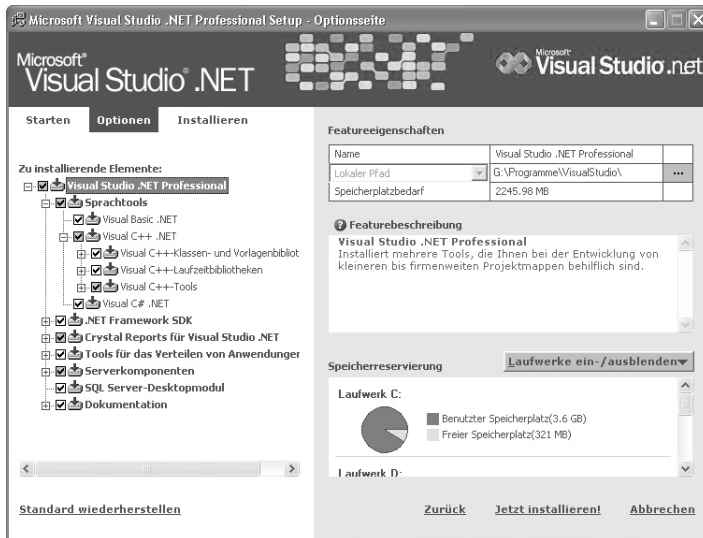


Abbildung 2.2: Auswahl des Installationsumfangs

2.2 Überblick über die IDE

Dieser Abschnitt gibt einen kurzen Überblick über die IDE. Wenn Sie zuvor bereits mit Visual Studio 6.0 oder Visual InterDev gearbeitet haben, werden Sie sich vermutlich schnell zurechtfinden.

2.2.1 Genereller Aufbau des Studios

Das Programm besteht aus mehreren unabhängigen Fenstern mit spezieller Bedeutung, die standardmäßig so angeordnet sind, dass Sie links Werkzeuge und Hilfsmittel finden, in der Mitte einen großen Bereich mit dem eigentlichen Editor und rechts die Projektverwaltung und eventuell Hilfe- und Ausgabefenstern. Sie arbeiten also quasi »von links nach rechts«: Werkzeug nehmen, Code schreiben, Speichern und Ausführen.

Wenn Sie das Studio starten, wird jedoch zuerst die Startseite angezeigt (siehe Abbildung 2.3).

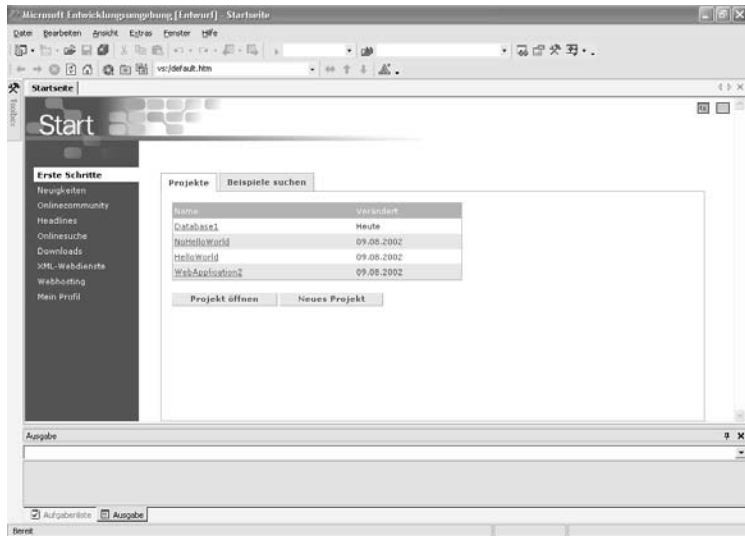


Abbildung 2.3: Startseite des Visual Studios

Im »laufenden Betrieb« bietet sich meist folgendes Bild:

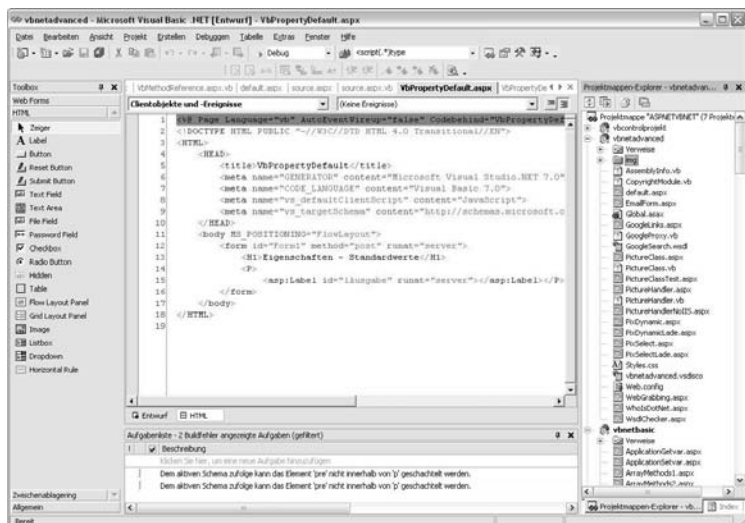


Abbildung 2.4: Typische Ansicht der Entwicklungsumgebung

Weitere Fenster lassen sich über das Menü ANSICHT ein- und ausblenden. Am Anfang sollten Sie sich mit dem PROJEKT-EXPLORER beschäftigen, denn für die Projektverwaltung ist Visual Studio .NET gut geeignet.

2.2.2 Verwaltung von Projekten

Visual Studio .NET bietet eine zweistufige Projektverwaltung. Diese basiert auf so genannten Projektmappen, die mehrere Projekte logisch zusammenfassen. Sie erscheinen dann unterhalb der Projektmappe in Form einer Baumstruktur. Die Programme aus diesem Buch sind in einer solchen Projektmappe enthalten.

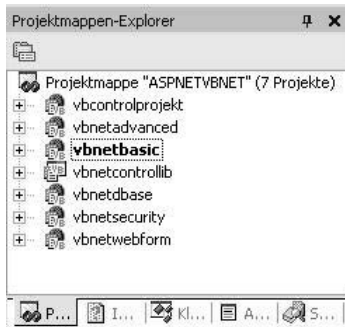


Abbildung 2.5: Die Projektmappe des Buches

Über das Kontextmenü können Sie weitere Projekte einer bestehenden Projektmappe hinzufügen. Wenn Sie dagegen neu beginnen, sollten Sie zuerst eine Projektmappe anlegen. Wenn Sie gleich mit einem Projekt starten, legt Visual Studio .NET eine Projektmappe automatisch an.

Bedeutung der Projekte

Visual Studio .NET erzeugt pro Projekt eine Assembly bzw. ein Programm. Die Verteilung der Daten auf Dateien spielt dabei keine Rolle. Der Compiler fasst alle Strukturen zusammen. Die Aufteilung dient mehrheitlich der Übersicht des Menschen, nicht der des Computers. Ein ähnliches Ordnungskriterium stellen die Namensräume dar. Diese mit dem Schlüsselwort *Namespace* erzeugten Strukturen ordnen Klassen in einen logischen Baum. Der Zugriff ist dadurch auch dann möglich, wenn mehrere Klassen denselben Namen tragen. Die Logik ist wiederum für den Compiler bedeutend, während die Aufteilung der Namensräume auf Dateien irrelevant ist. In VB.NET wird für jedes Projekt ein bestimmter Namensraum voreingestellt, sodass die wiederholte Angabe in den Code-Dateien nicht erforderlich ist.



Generell gilt also: Die oberste logische und physische Ebene bildet ein Projekt. Die logische Strukturierung innerhalb des Projekts erfolgt mit Namensräumen. Die Aufteilung auf Dateien hat keinen Einfluss auf das fertige Programm. Sie können einzelne Dateien nicht übersetzen oder zu einer eigenständigen Assembly werden lassen. Sie können aber Dateien aus einem Projekt ausschließen, wenn Sie deren Code nicht mehr benötigen. Dazu muss die Datei nicht gelöscht werden.

Eine Projektmappe erstellen

Um eine Projektmappe zu erstellen, gehen Sie, beginnend mit der Startseite, folgendermaßen vor:

1. Wählen Sie im Menü DATEI | NEU den Eintrag LEERE PROJEKT-MAPPE.
2. Lassen Sie im folgenden Dialog die Vorlage LEERE PROJEKT-MAPPE aktiviert und vergeben Sie der Mappe einen Namen.
3. Legen Sie die Projektmappe mit OK an.

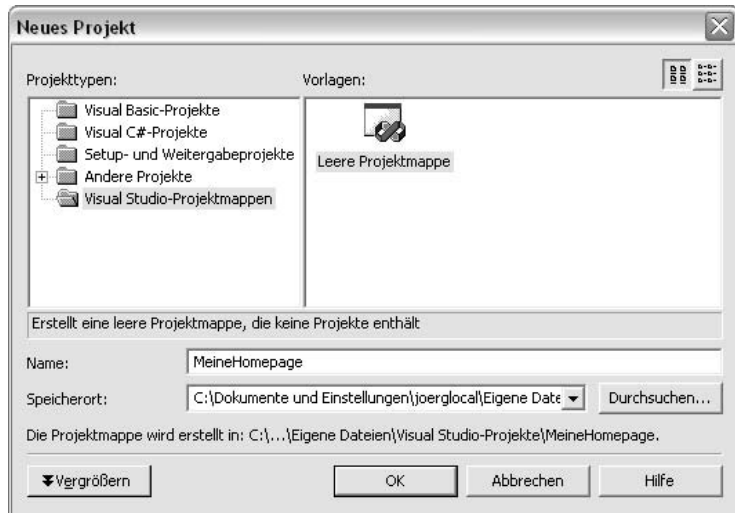


Abbildung 2.6: Anlegen einer Projektmappe

Damit ist die Mappe fertig. Fügen Sie nun ein Projekt hinzu, das später die Code-Dateien und andere Ressourcen aufnimmt.

Ein neues Projekt anlegen

Ein neues Projekt können Sie über das Kontextmenü der Projektmappe oder über DATEI | NEU | PROJEKT erzeugen. Im folgenden Dialog wählen Sie zuerst die Programmiersprache aus, die Sie verwenden möchten (In diesem Buch ist das immer VB.NET). Dann müssen Sie entscheiden, welche Art Applikation Ihr Projekt beherbergen soll. Am Anfang wird das immer auf der Vorlage ASP.NET-WEBANWENDUNG basieren.

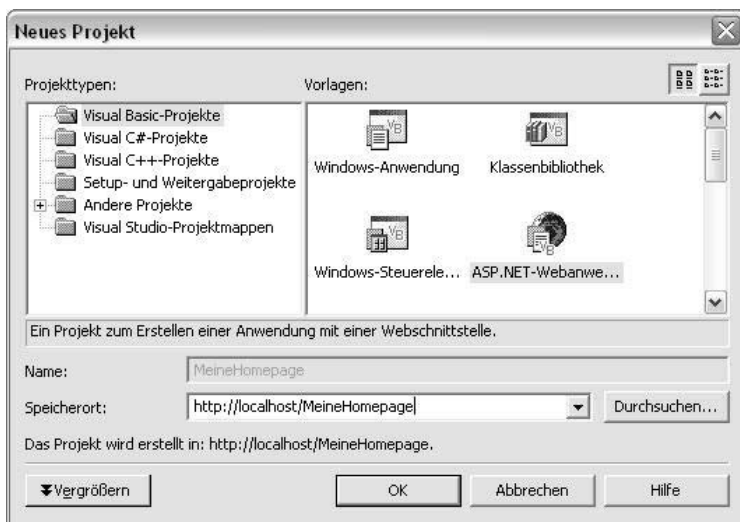


Abbildung 2.7: Ein neues Projekt für eine ASP.NET-Webanwendung erstellen

Mit dem Anlegen des Projekts führt Visual Studio .NET einige Schritte aus, die spürbar Zeit beanspruchen:

1. Es wird ein Projektverzeichnis angelegt, das als Ausgangspunkt für den Webserver dient, beispielsweise:
`c:\inetpub\wwwroot\MeineHomepage`
2. Es wird im IIS ein virtuelles Verzeichnis angelegt, dessen Lage und Namen Sie beim Erstellen des Projekts bestimmt haben.
3. Das virtuelle Verzeichnis wird zu einer eigenständigen Applikation konvertiert.
4. Das Projektverzeichnis wird mit einigen Dateien gefüllt:

► *AssemblyInfo.vb*

Eine Informationsdatei, mit der das Manifest der erzeugten Assembly gesteuert wird. Dies können Sie vorerst unbeachtet lassen.

► *Global.asax*

Diese Datei – und die im Projekt-Explorer unsichtbare Code-Datei *Global.asax.vb* – dient der Programmierung von verschiedenen Ereignissen, die während der Seitenverarbeitung auftreten.

► *<Projektname>.vsdisco*

Diese Datei dient als Informationsquelle für andere Server, wenn Sie Webservices programmieren. Normalerweise können Sie sie ignorieren.

► *Web.config*

Diese Datei konfiguriert Ihre Applikation. Der Einsatz ist auf Live-Servern notwendig. Auf einem Entwicklungsrechner hat die Verwendung mehr experimentellen Charakter.

► *WebForm1.aspx*

Damit Sie gleich loslegen können, hat Visual Studio .NET gleich eine Web Form erzeugt. Wenn Sie diese öffnen und **[F7]** drücken, gelangen Sie in die dazu erzeugte Code-Datei.

5. Ein Zweig mit Verweisen wird angezeigt. Dieser Eintrag dient dem Hinzufügen von Teilen des Frameworks oder von COM-Komponenten, die zusätzlich benötigt werden.

Die Eigenschaften eines Projekts

Allgemeine Einstellungen

Im Kontextmenü des Projekts finden Sie den Eintrag EIGENSCHAFTEN. Hierüber erreichen Sie einige Einstellungen, die auf alle neu erzeugten Dateien und das fertige Programm Einfluss haben. Unter ALLGEMEINE EINSTELLUNGEN sind folgende besonders wichtig:

► ASSEMBLYNAME

Name der Assembly, die dieses Projekt erzeugen wird.

► STANDARDNAMESPACE

Der Namensraum, der beim Erzeugen verwendet werden soll.

► STARTOBJEKT

Dieser Eintrag gibt das Objekt an, mit dem der Debugger starten soll.

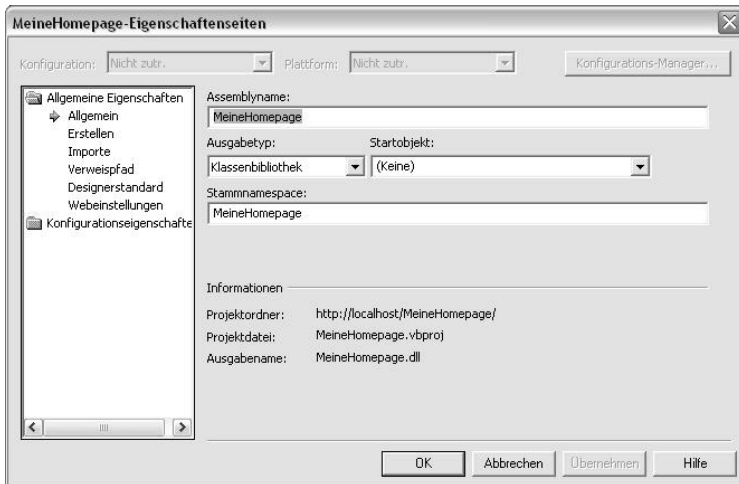


Abbildung 2.8: Allgemeine Eigenschaften eines Projekts

Unter KONFIGURATIONSEIGENSCHAFTEN sind folgende besonders wichtig:

Konfigurations-eigenschaften

► ERSTELLEN

Legen Sie hier den Ausgabepfad für die Assembly fest. Der Standardpfad ist *bin*.

► DEBUGGEN

Achten Sie hier darauf, dass die Option ASP.NET-DEBUGGEN AKTIVIEREN während der Entwicklungsphase aktiviert ist. Wichtig ist auch die Startaktion, die immer auf PROJEKT STARTEN stehen sollte. Außerdem ist die Startseite festzulegen. Für die Buchprojekte bietet sich *default.aspx* an, ein Programm, das alle aspx-Seiten zur Ausführung und zur Betrachtung des Quelltextes anbietet.

Weitere Optionen betreffen das Speichern von Konfigurationen und feinere Einstellungen der Übersetzungsbedingungen. Dies ist für fortgeschrittene Benutzer interessant und wird für die Abarbeitung der Beispiele im Buch nicht benötigt.

Weitere Optionen

Das Projekt mit Leben erfüllen

Sie können jetzt loslegen und programmieren. Wie es geht, zeigt der folgende Abschnitt, der Schrittweise in die Programmierung einer Musterapplikation und anschließend in die Technik der Datenbanknutzung einführt. Der Ablauf nutzt ein Visual Studio .NET-Projekt.

2.3 Erste Schritte in der Programmierung

Nachdem Sie sich einen ersten Überblick über das Visual Studio verschafft haben, sollten wir ein erstes Beispiel ausprobieren. Sie werden dabei sehen, mit wie wenig Aufwand Sie im Visual Studio Programme generieren können.

2.3.1 Kein *Hello World* im Studio

Als Einstiegsbeispiel soll auch hier unser abgewandeltes Hello World zum Zuge kommen. Starten Sie dazu zunächst im Studio ein neues Projekt über DATEI | NEU. Vergeben Sie dem Projekt einen sinnvollen Namen, beispielsweise *NoHelloWorld*.

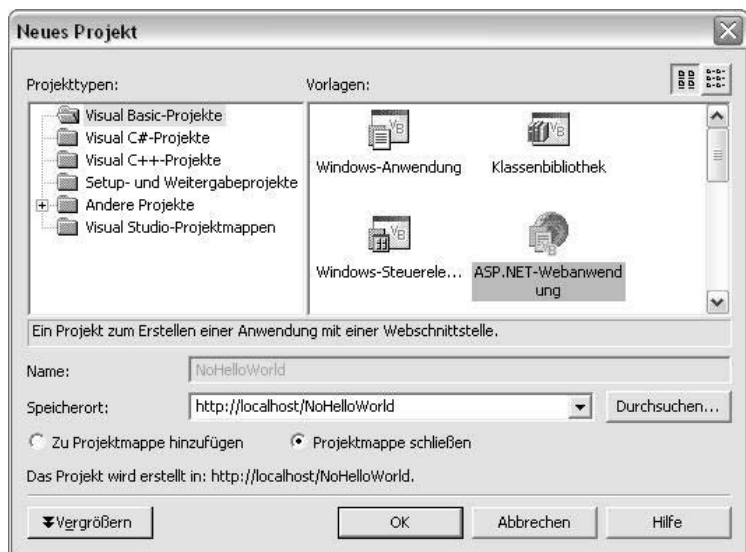


Abbildung 2.9: Start eines neuen Projekts

Sie sehen nun die Designer-Oberfläche vor sich. Eine erste, noch leere Webform wurde bereits angelegt und im entsprechenden Verzeichnis des IIS veröffentlicht. Darüber hinaus befinden sich dort bereits eine ganze Anzahl weiterer Dateien.

Für den Beginn wollen wir in dieser leeren Webform nur ein Label mit etwas Text definieren, welches dann mit Hilfe von ASP.NET dynamisch mit Inhalt gefüllt wird. Klicken Sie auf der TOOLBOX am rechten Rand auf LABEL. Ziehen Sie den Eintrag aus dieser Leiste auf die noch leere Webform oder ziehen Sie einfach, nachdem der Mauszeiger zu einem Plus-Zeichen geworden ist, auf der Fläche ein Rechteck in der gewünschten Größe. Nun sollte Ihr Bildschirm etwa so aussehen, wie in Abbildung 2.10 gezeigt.

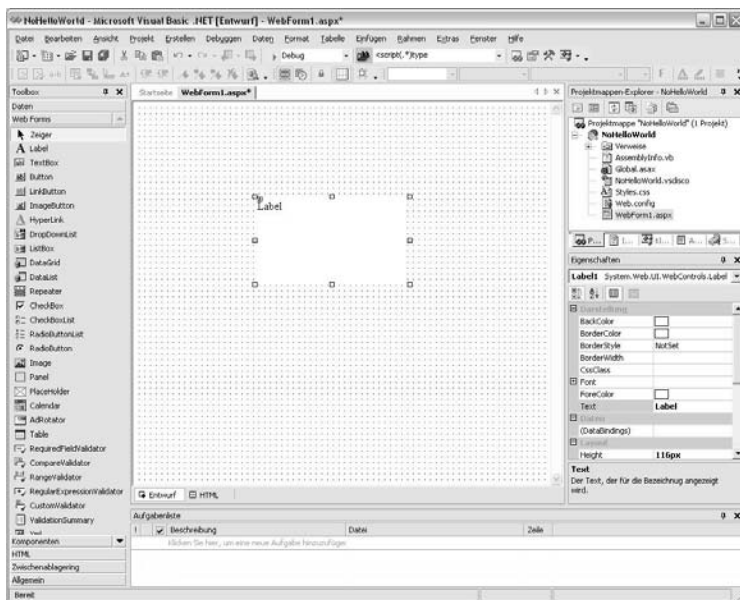


Abbildung 2.10: Das platzierte Label in der Webform

Am rechten unteren Bildrand erscheint das EIGENSCHAFTEN-Fenster für das Label, wenn Sie dieses mit der Maus markieren. Suchen Sie hier den Eintrag TEXT und geben Sie den folgenden Text ein: *Heute ist folgendes Datum:.* Doppelklicken Sie dann auf das Label. Es erscheint eine neue Ansicht – nämlich die auf den Code der neuen Anwendung.

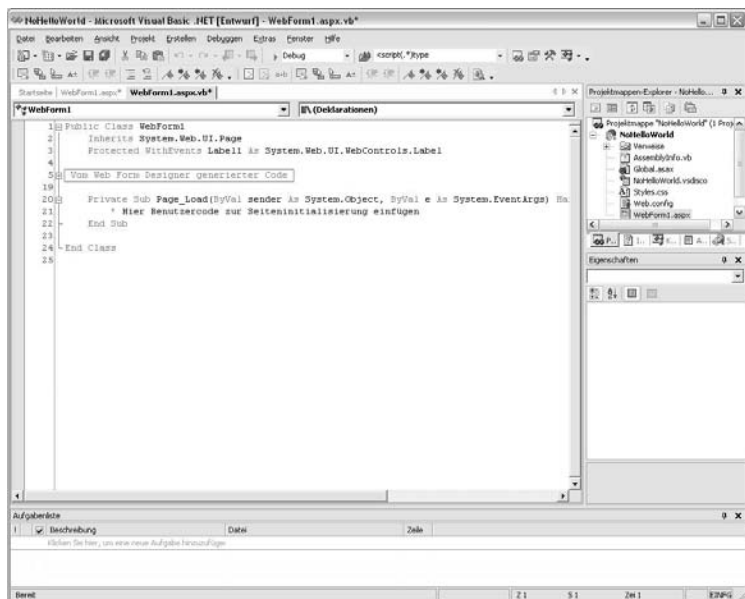


Abbildung 2.11: Umschalten in die Code-Ansicht

Was VS.NET erzeugt hat

Obwohl Sie noch keinen Code geschrieben haben, finden Sie hier schon einige Zeilen davon. Gehen Sie zu der Zeile, die durch den Kommentar ' Hier Benutzercode... ' gekennzeichnet ist. Fügen Sie direkt danach oder davor eine neue Zeile ein und geben Sie den folgenden Code ein:

```
Label1.Text = Label1.Text & DateTime.Now.Day.ToString() & ". " _  
              & DateTime.Now.Month.ToString() & ". "
```



Beachten Sie, dass unter VB.NET, anders als beispielsweise unter C#, eine Anweisung immer vollständig auf einer Zeile erscheinen muss. Sie können für eine bessere Übersichtlichkeit die Zeilen im Editor trennen, indem Sie am Zeilenende vor dem Umbruch einen Unterstrich, geführt von einem Leerzeichen, einsetzen.

Label1 ist der Bezeichner des von Ihnen angelegten Objekts. Sie können das im EIGENSCHAFTEN-Fenster des Labels überprüfen, wenn Sie dort den Eintrag (ID) aufsuchen. Zwischen der Designer- und der Codesicht können Sie über die Registerkarten am oberen Bildschirmrand, jedoch unterhalb der Symbolleiste, umschalten. Unter der Registerkarte WEBFORM1.ASPX finden Sie den Designer, unter *Webform1.aspx.vb* den Code im Code-Editor. Hier

wird schon das grundlegende Konzept deutlich, nach welchem Sie im Visual Studio bevorzugt programmieren sollten: *Code Behind*. Das bedeutet, dass Programmcode und HTML-Code strikt voneinander getrennt sind. Ausführlicher wurde darauf bereits in Abschnitt 1.5.2, »Code Behind« ab Seite 79 eingegangen.

Haben Sie alles korrekt eingegeben, können Sie Ihr Programm über einen Druck auf **F5** starten.

*An dieser Stelle soll im Vorgriff auf die Beschreibung des Debuggers die Bedeutung der Tastenkombinationen **Strg** + **F5** bzw. **F5** verraten werden. Mit **F5** starten Sie das Programm mit dem Debugger. Wenn Sie dagegen **Strg** + **F5** verwenden, wird es nur gestartet. Gespeichert und kompiliert wird es natürlich in jedem Fall automatisch.*



Zunächst wird der Übersetzungsvorgang gestartet und, falls dieser erfolgreich verläuft, die Assembly erzeugt und im Verzeichnis des IIS abgelegt. Dann sollte Ihr Internet Explorer starten und in etwa die folgende Website anzeigen:

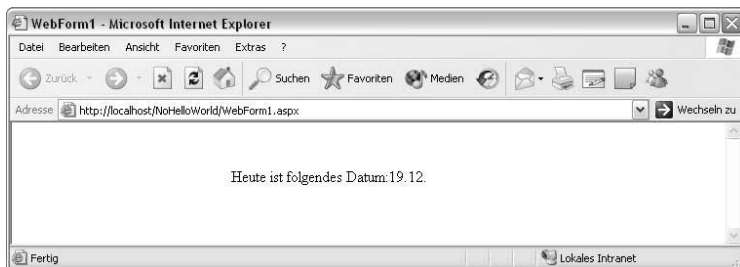


Abbildung 2.12: Das erste mit dem Studio erzeugte Programm

Zuerst erfolgt die Definition der Klasse `WebForm1`. Diese wurde mit dem Anlegen des Projekts automatisch von `System.Web.UI.Page` abgeleitet (Schlüsselwort *Inherits*). Sie muss selbst wiederum als `Public` deklariert sein, damit sie direkt ausgeführt werden kann. Unter anderem finden Sie im Abschnitt 4.3, »Objektorientierte Programmierung« ab Seite 176 dazu weitere Informationen.

Dann erfolgt die Deklaration der in der Webform angelegten Objekte. Hier ist zunächst nur ein einziges vorhanden: `Label1`. Dieses ist wiederum abgeleitet von der Klasse `System.Web.UI.WebControls.Label`.

Wie es funktioniert



Der Namensraum `System.Web.UI.WebControls` wird standardmäßig eingebunden, auch wenn er in der automatisch erzeugten Deklaration nicht explizit aufgeführt wird. Somit funktioniert auch die verkürzte Schreibweise `Protected Label As Label`.

Das Objekt ist deshalb als `Protected` gekennzeichnet, da es nur innerhalb Ihres neuen Namespaces (hier: `NoHelloWorld`) zum Einsatz kommt.

Das Schlüsselwort `WithEvents` ist ebenfalls automatisch erzeugt worden und an dieser Stelle ohne Funktion. Es ist nur dann erforderlich, wenn das Objekt auf Ereignisse reagieren soll. Mehr zum Thema Ereignisbehandlung finden Sie in Abschnitt »Ereignisse im Kontext von VB.NET« ab Seite 178.

Der Name der einzigen enthaltenen Methode `Page_Load` steht fest und darf nicht verändert werden. Diese Methode liefert zwar nichts zurück (`Sub`), wird aber ausgeführt, wenn die Seite geladen wurde. Genauer wird dieser Prozess in Abschnitt 2.6, »Grundlagen der Seitenverarbeitung« ab Seite 68 beleuchtet.

2.3.2 Ein erstes Eingabeformular

Als zweites Beispiel für die Programmierung mittels Visual Studio soll ein Eingabeformular dienen. Sie können dazu das vorherige Projekt erweitern oder mit einem neuen anfangen. Es soll ein kleines Formular erstellt werden, in welchem Sie Ihren Namen und Vornamen eingeben können. Bei Klick auf eine `Absenden`-Schaltfläche soll dann eine *Hallo*-Ausschrift mit Ihrem kompletten Namen unterhalb des Formulars angezeigt werden.

Schalten Sie zunächst in die Designer-Sicht um. Ziehen Sie dann mit der Maus aus der Toolbox die folgenden Objekte auf die Seite:

- ▶ Drei `Label`-Steuerelemente
- ▶ Zwei `TextBox`-Steuerelemente
- ▶ Ein `Button`-Steuerelement

Ordnen Sie diese so an, wie in nachfolgender Abbildung gezeigt wird:

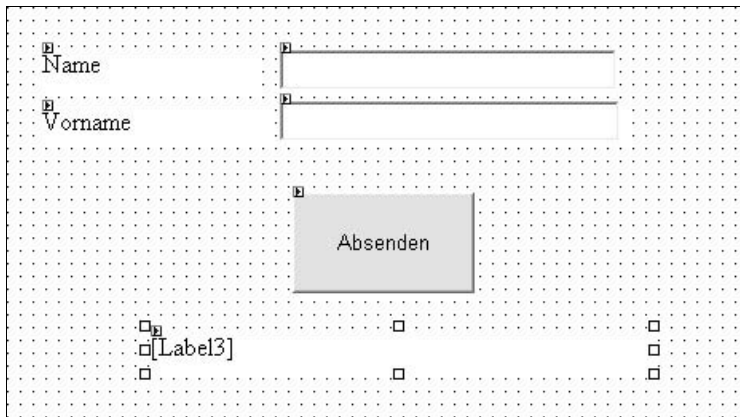


Abbildung 2.13: Ein einfaches Eingabeformular

Beschriften Sie die ersten beiden LABEL mit *Name* und *Vorname* über deren jeweiliges EIGENSCHAFTEN-Fenster (Funktionstaste F4, Eigenschaft TEXT). Geben Sie dem Button auf die gleiche Weise die Aufschrift *Absenden*.

Wenn Sie nun in die Code-Ansicht umschalten, finden Sie in etwa den folgenden bereits generierten Code vor:

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label2  ➤
        As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox1  ➤
        As System.Web.UI.WebControls.TextBox
    Protected WithEvents TextBox2  ➤
        As System.Web.UI.WebControls.TextBox
    Protected WithEvents Button1  ➤
        As System.Web.UI.WebControls.Button
    Protected WithEvents Label3  ➤
        As System.Web.UI.WebControls.Label
    Protected WithEvents Label1  ➤
        As System.Web.UI.WebControls.Label

#Region " Vom Web Form Designer generierter Code "

    'Dieser Aufruf ist für den Web Form-Designer erforderlich.
    <System.Diagnostics.DebuggerStepThrough()> ➤
    Private Sub InitializeComponent()
```

```

End Sub

Private Sub Page_Init(ByVal sender As System.Object,
                    ByVal e As System.EventArgs)
    Handles MyBase.Init
        'CODEGEN: Diese Methode ist für den Web Form-Designer
        erforderlich
        'Verwenden Sie nicht den Code-Editor zur Bearbeitung.
        InitializeComponent()
    End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object,
                    ByVal e As System.EventArgs)
    Handles MyBase.Load
        ' Hier Benutzercode zur Seiteninitialisierung einfügen

    End Sub

End Class

```

Ein erster Start Wenn Sie jetzt die Funktionstaste **[F5]** drücken, werden Sie feststellen, dass das Programm aufgerufen wird und grundsätzlich funktioniert.

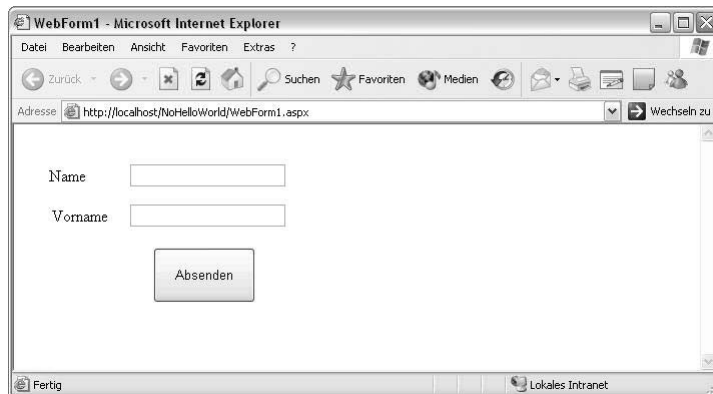


Abbildung 2.14: Das leere Formular nach dem ersten Start

Sie können sogar schon Werte in die beiden Formularfelder eingeben. Bei einem Klick auf den Button ABSENDEN passiert allerdings scheinbar nichts. Im Hintergrund sind allerdings der Webserver IIS und ASP.NET bereits kräftig am Werken. Das Formular wird

beim Klick auf den Button an den IIS gesendet. Das betrifft auch den Inhalt der beiden eventuell gefüllten Eingabefelder. Der Inhalt wird zwar noch nicht weiter durch das Programm verarbeitet, beim Zurücksenden des Formulars an den Browser werden diese Werte aber wieder übergeben. So kommt es, dass Sie nach dem Absenden nicht wieder ein geleertes Formular vorfinden, wie dies mit der klassischen HTML-Formularprogrammierung zwangsläufig der Fall ist.

Damit bei einem Klick auf ABSENDEN der Inhalt der beiden Felder verarbeitet werden kann, müssen Sie die Routine zur Behandlung des Klick-Ereignisses entsprechend erweitern. Führen Sie dazu im Designer einen Doppelklick auf den Button aus. Die Ansicht schaltet daraufhin in den Code und Sie finden die folgenden neuen Zeilen vor:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e  
As System.EventArgs) Handles Button1.Click
```

```
End Sub
```

Die automatisch angelegte Funktion `Button1_Click` dient der Ereignisbehandlung und kann jetzt mit sinnvollem Code ausgestattet werden. Um bei Klick ein *Hallo* an den Besucher, der seinen Namen eingegeben hat, zu senden, tragen Sie in diese Funktion Folgendes ein:

```
Label3.Text = "Hallo " & TextBox2.Text & _  
" " & TextBox1.Text & "!"
```

Wenn Sie jetzt das Programm starten, sollte nach dem Klick auf ABSENDEN das Eingabeformular etwa wie folgt aussehen (siehe Abbildung 2.15).

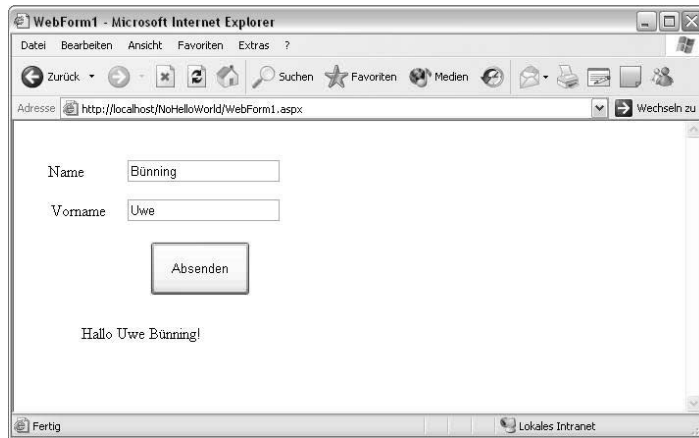


Abbildung 2.15: Das fertige Formular nach dem Ausfüllen und Absenden

Wie es funktioniert

Im Unterschied zum ersten Beispiel, bei dem nur eine Website mit einem dynamischen Inhalt generiert wird, muss diese Site auf eine Benutzeraktion reagieren. Dazu wird mit ASP.NET eine Ereignisbehandlung durchgeführt, welche der Code Designer automatisch implementiert. Für ein besseres Verständnis stellt die folgende Grafik die Prozesse dar, die dabei ablaufen:

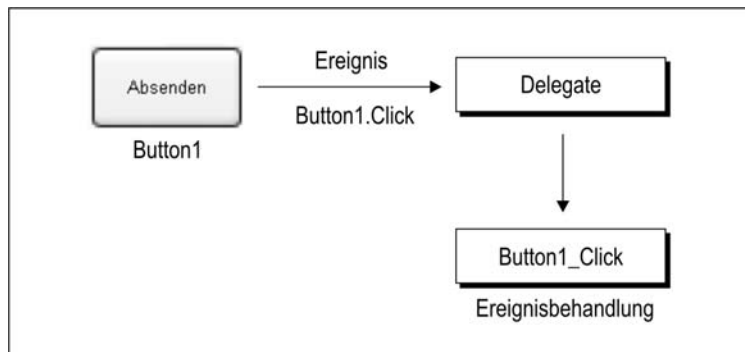


Abbildung 2.16: Prozess der Ereignisbehandlung

Den Klick behandeln

Beim Klick auf die Schaltfläche mit der Bezeichnung `Button1` wird ein Ereignis `Button1.Click` ausgelöst. Dieses Ereignis wird durch die Methode `Button1_Click` behandelt.

Was geschieht im HTML?

Interessant ist für dieses Beispiel, zu untersuchen, wie die Kommunikation zwischen Client (Browser) und dem Server (ASP.NET) abläuft. Wenn Sie sich im Browser-Fenster den HTML-

Code ansehen, werden Sie feststellen, dass hier nichts direkt auf Ereignisse hindeutet.

```
<form name="Form1" method="post" action="WebForm1.aspx"
id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwxMjI3NjcyMTg1Ozs+ytj3sLLuf+09ZaI5GuA0SCpfz5U=" />
<span id="Label1" style="height:30px;width:168px;Z-INDEX: 101;
LEFT: 64px; POSITION: absolute; TOP: 43px">Name:</span>
<span id="Label2" style="height:27px;width:175px;Z-INDEX: 102;
LEFT: 64px; POSITION: absolute; TOP: 89px">Vorname:</span>
<input name="TextBox1" type="text" id="TextBox1"
style="height:40px;width:213px;Z-INDEX: 103; LEFT: 241px;
POSITION: absolute; TOP: 39px" />
<input name="TextBox2" type="text" id="TextBox2"
style="height:36px;width:211px;Z-INDEX: 104; LEFT: 241px;
POSITION: absolute; TOP: 84px" />
<input type="submit" name="Button1" value="Absenden" id="Button1"
style="height:76px;width:223px;Z-INDEX: 105; LEFT: 164px;
POSITION: absolute; TOP: 140px" />
<span id="Label3" style="height:44px;width:435px;Z-INDEX: 106;
LEFT: 79px; POSITION: absolute; TOP: 232px">Label</span>
</form>
```

Listing 2.1: Auszug aus dem zum Browser gesendeten HTML-Code

Die für die nachfolgenden Betrachtungen wesentlichen Teile finden Sie im HTML-Formular. Per POST-Kommando soll das Formular an den Server gesendet werden. Dabei wird die *aspx*-Datei *Webform1.aspx* angefordert. Der *id*-Parameter¹ wird intern von ASP.NET erzeugt und verwendet. Im ersten versteckten Feld mit dem Namen `__VIEWSTATE` werden alle Felder des Formulars sowie deren Inhalt Base64-kodiert gespeichert. Weiter unten im Text wird das Beispiel erweitert, um Änderungen an den Eingabefeldern zu erkennen. Dort wird dann deutlich, wie ASP.NET mit einer Auswertung von `__VIEWSTATE` umgeht.

**Worauf es
ankommt**

Danach folgen die Eingabefelder mit ihren Beschriftungen. Durch ASP.NET wird ein Feld durch seine *id* erkannt. Über POST wird das Formular mit den Feldern und ihrem Inhalt an den Server geschickt. Hier wird `__VIEWSTATE` neu gesetzt und zusammen mit dem neuen HTML-Code, der nun auch die Feldwerte enthält (Parameter *value*), an den Browser zurück gesendet.

¹ Diesen können Sie auch selbst deklarieren, um sie beispielsweise über JavaScript anzusprechen. Darauf wird im vorliegenden Buch nicht weiter eingegangen. Wenn Sie dies nicht vorhaben, sollten Sie den *id*-Parameter generell unverändert lassen.

```

<form name="Form1" method="post" action="WebForm1.aspx"
id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwxMjI3NjcyMTg1O3Q8O2w8aTwXPjs+02w8dDw7bDxpPDEXP
js+02w8dDxwPHA8bDxUZXh00z47bDxIYWxsbyBKw7ZyZyBLcmF1c2U
h0z4+0z47Oz47Pj47PtTHk1+eKNCKAzIsM2wn92vKKntD" />

<span id="Label1" style="height:30px;width:168px;Z-INDEX: 101;
LEFT: 64px; POSITION: absolute; TOP: 43px">Name:</span>
<span id="Label2" style="height:27px;width:175px;Z-INDEX: 102;
LEFT: 64px; POSITION: absolute; TOP: 89px">Vorname:</span>
<input name="TextBox1" type="text" value="Krause" id="TextBox1"
style="height:40px;width:213px;Z-INDEX: 103; LEFT: 241px;
POSITION: absolute; TOP: 39px" />
<input name="TextBox2" type="text" value="Jörg" id="TextBox2"
style="height:36px;width:211px;Z-INDEX: 104; LEFT: 241px;
POSITION: absolute; TOP: 84px" />
<input type="submit" name="Button1" value="Absenden" id="Button1"
style="height:76px;width:223px;Z-INDEX: 105; LEFT: 164px;
POSITION: absolute; TOP: 140px" />
<span id="Label3" style="height:44px;width:435px;Z-INDEX: 106;
LEFT: 79px; POSITION: absolute; TOP: 232px">Hallo Jörg
Krause!</span>
</form>

```

Listing 2.2: Das ausgefüllte und zurückgesendete Formular

Ihnen sollte bewusst sein, dass der Inhalt der Eingabefelder nur deshalb erhalten geblieben ist, weil der Browser ein entsprechend *ausgefülltes* Formular vom Server zurück erhalten hat. ASP.NET hat dazu den Wert von `__VIEWSTATE` ausgewertet und daraus die Inhalte und Attribute der Felder des Formulars neu gesetzt.

Feldänderungen erkennen

Um zu erkennen, ob ein Benutzer ein Eingabefeld geändert hat, können Sie dessen Wert in einer Variablen speichern, deren Wert über das Sitzungsmanagement erhalten und danach mit dem neuen Wert vergleichen. Einfacher geht es aber, wenn Sie dies über die speziell dafür verfügbare Ereignisbehandlungs-Methode für Eingabefelder erreichen. Wenn Sie im Webform-Designer auf ein Eingabefeld doppelklicken, wird automatisch im Code eine entsprechende Methode deklariert:

```

Private Sub TextBox1_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs)
Handles TextBox1.TextChanged

End Sub

```

Dies ist die Methode, die zur Behandlung des Ereignisses `TextBox1.TextChanged` herangezogen wird, wenn sich der Inhalt des Feldes geändert hat. Das Ereignis `TextBox1.TextChanged` wird dabei folgendermaßen ausgelöst: Bei jedem Empfangen des Formulars wird durch ASP.NET das Feld `__VIEWSTATE`, in welchem der vorhergehende Stand der Felder mit ihren Werten gespeichert ist, mit den aktuellen Werten verglichen. Wird erkannt, dass ein Feld nun einen anderen Wert aufweist als in `__VIEWSTATE`, erfolgt ein Auslösen dieses Ereignisses. Über eine Deklaration einer entsprechenden Behandlungsmethode können Sie dann entscheiden, wie damit umgegangen wird. Für eine Demonstration können Sie das Beispiel wie folgt erweitern:

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object,   
                                ByVal e As System.EventArgs)   
                                Handles TextBox1.TextChanged   
    Label1.ForeColor = Color.Red   
End Sub
```

Wenn Sie das Programm jetzt aufrufen, werden Sie feststellen, dass sich das erste Label, die Bezeichnung *Name*, rot färbt, sobald Sie das Formular abgesendet und einen Namen eingetragen haben. Das ist korrekt, da sich der Feldinhalt im Vergleich zum ursprünglich gesendeten Formular verändert hat. Leider bleibt das Label rot, auch wenn Sie danach das Formular unverändert senden. Das liegt daran, dass auch das Farbattribut des Labels mit im `__VIEWSTATE` erhalten bleibt. Wollen Sie erreichen, dass sich das Label nur einfärbt, wenn sich das Feld geändert hat, können Sie dessen Attribut innerhalb der Methode `Page_Load` immer wieder auf *Schwarz* zurücksetzen:

```
Label1.ForeColor = Color.Black
```

Warum wird das Label wieder schwarz, wenn sich der Inhalt des Feldes *nicht* geändert hat? Um dies zu verstehen, sollten Sie sich die Abarbeitungsreihenfolge unter ASP.NET vor Augen führen:

Abarbeitungs- reihenfolge

1. Zuerst werden alle Objekte der Seite im Speicher des Servers aufgebaut. Hierfür wird auch das Feld `__VIEWSTATE` ausgewertet und die Feldinhalte und -Attribute damit gesetzt. Ergibt sich hier für das Label das Farbattribut *Rot*, wird dieses auch in dieser Farbe auf der Seite erzeugt.

2. Ist die Seite soweit fertig, erfolgt der Aufruf der `Page_Load`-Methode. In diesem Beispiel wird damit die Farbe des Labels explizit auf `Color.Black` gesetzt.
3. Nach `Page_Load` werden alle weiteren Ereignisse behandelt. Wird dabei über die Methode `TextBox1.TextChanged` erkannt, dass sich der Inhalt von `TextBox1` geändert hat, erfolgt der Aufruf von `TextBox1_TextChanged` und `Label1` wird auf `Color.Red` gesetzt.



Aufrufe wie `Color.Red` sind keine Konstanten, sondern Aufrufe statischer Klassen bzw. Klassenmitglieder. In diesem Fall wird die Klasse `Color` verwendet und die Eigenschaft `Red` gibt den entsprechenden Farbwert zurück.

Mit dem in diesem Abschnitt besprochenen Beispiel wird deutlich, dass die Verarbeitung von Formularen in ASP.NET nicht unbedingt trivial ist. Haben Sie allerdings die grundlegenden Prozesse verstanden, die dabei ablaufen, können Sie unter Zuhilfenahme der Automatismen zu einem sehr kompakten und eleganten Code kommen. Kennen Sie das alte ASP, werden Sie verstehen, was damit gemeint ist. Weiterführende Informationen zum Sitzungsmanagement von ASP.NET finden Sie in Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778.

2.3.3 Ein erstes Datenbankprojekt

Nach den ersten Programmbeispielen soll in diesem Abschnitt gezeigt werden, wie einfach es sein kann, mit dem Visual Studio Datenbankanwendungen zu programmieren. Die folgenden drei Schritte sind notwendig, um mit dem Visual Studio eine Seite mit einer Datenbankabfrage zu erstellen:

1. Sie erzeugen einen so genannten *Datenadapter*, mit dem die Verbindung zur Datenbank definiert wird. Sie haben dabei die Auswahl zwischen verschiedenen Providern, mit denen Sie unter anderem Datenbanken unter Access oder dem SQL Server sehr einfach über einen grafischen Assistenten einbinden können.
2. Für das Zwischenspeichern der Daten benötigen Sie ein *DataSet*-Objekt.
3. Für die Anzeige der Datensätze aus dem *DataSet* können Sie ein *DataGrid*-Steuerelement verwenden.

Datenadapter erzeugen

Im nachfolgenden Beispiel wird aus der Access-Beispieldatenbank *Shop.mdb* die Liste der *Adressen*-Tabelle ausgelesen und auf einer Website dargestellt. Diese Datenbank finden Sie auf der dem Buch beiliegenden CD.

Für das Nachvollziehen des Beispiels sollten Sie zuerst ein neues Projekt anlegen. Gehen Sie dann auf die Webform-Designer-Ansicht. Wählen Sie aus der TOOLBOX unter DATEN den Eintrag OLEDBDATAADAPTER und ziehen Sie diesen mit der Maus auf die Webform. Damit startet ein Assistent, der Sie durch die weitere Einrichtung begleitet. Folgende Schritte sind für die Einrichtung notwendig:

1. Nach Bestätigung des Begrüßungsfensters können Sie eine bestehende Verbindung auswählen oder eine neue erstellen.

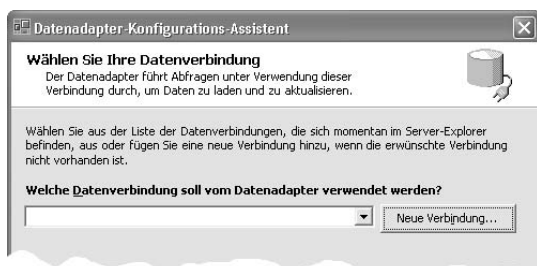


Abbildung 2.17: Auswahl einer Datenverbindung oder Erstellen einer neuen

Klicken Sie hier auf NEUE VERBINDUNG.

2. Danach wird das Fenster DATENVERKNÜPFUNGSEIGENSCHAFTEN geöffnet. Aktivieren Sie hier die Schaltfläche PROVIDER, um den Provider MICROSOFT JET 4.0 OLE DB PROVIDER zu markieren. Standardmäßig ist durch den Assistenten nämlich der OLE-Provider für den SQL-Server aktiviert.

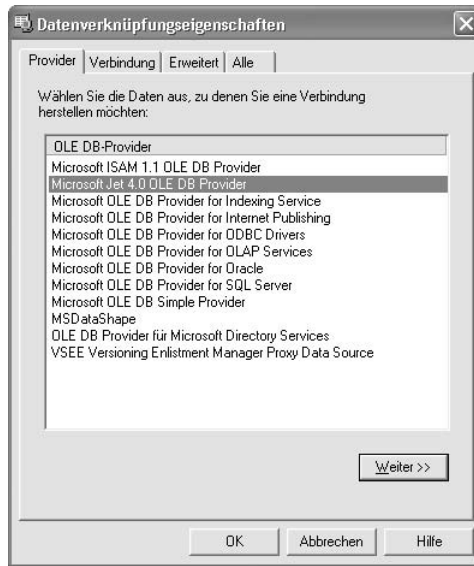


Abbildung 2.18: Auswahl des Jet-Providers für den Zugriff auf Access

3. In der Registerkarte VERBINDUNG legen Sie dann die weiteren Optionen für den Zugriff auf die Access-Datenbank *Shop.mdb* fest.

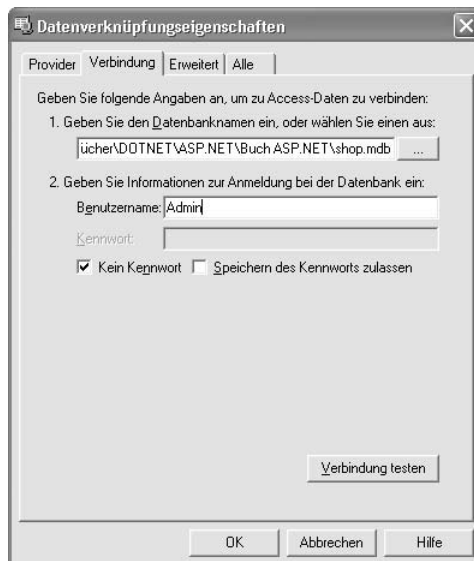


Abbildung 2.19: Auswahl der Datenbank *Shop.mdb*

Unter BENUTZERNAME geben Sie *Admin* ein. Ein Kennwort ist für die Beispieldatenbank nicht vergeben worden. Klicken Sie dann auf VERBINDUNG TESTEN. Konnte die Datenbank korrekt geöffnet werden, erscheint eine entsprechende Meldung.

4. Schließen Sie dann das Fenster wieder. Die anderen Optionen unter ERWEITERT und ALLE sind für dieses Beispiel nicht von Bedeutung. Es sollte nun wieder das Ausgangsfenster des Assistenten erscheinen, nur dass jetzt die neu erstellte Datenverbindung hier erscheint.

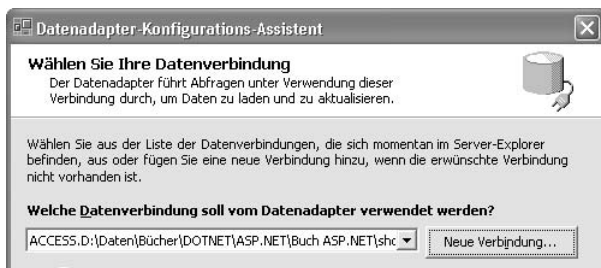


Abbildung 2.20: Neu erstellte Datenverbindung

Klicken Sie hier auf WEITER.

5. Wählen Sie im nächsten Fenster des Assistenten ABFRAGETYP AUSWÄHLEN den ersten Eintrag SQL-ANWEISUNGEN VERWENDEN.
6. Im danach folgenden Fenster des Assistenten können Sie manuell SQL-Anweisungen eingeben oder den ABFRAGE-GENERATOR aufrufen. Mit dem Generator können Sie selbst komplexe SQL-Abfragen erzeugen, ohne die SQL-Kommandos im Detail kennen zu müssen. Für dieses erste Beispiel sollten Sie im Generator nur die Tabelle *Adressen* auswählen. Markieren Sie dann die Felder dieser Tabelle, wie dies in nachfolgender Abbildung gezeigt wird.

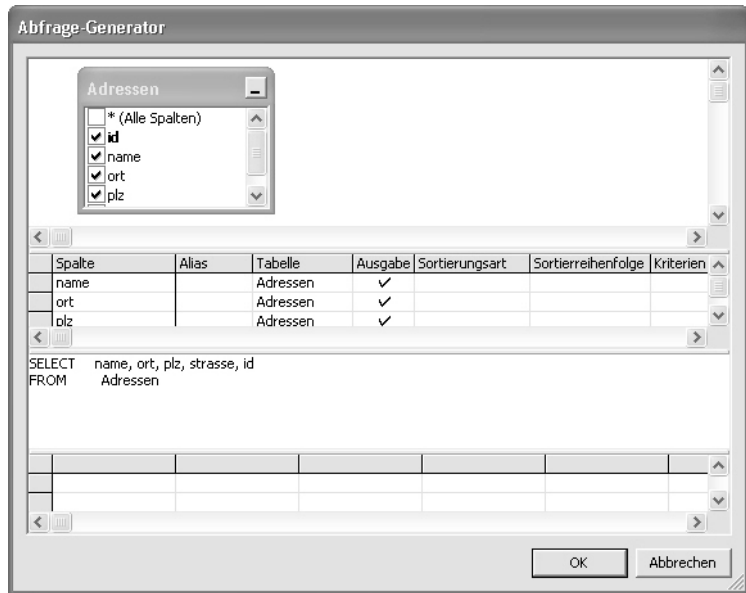


Abbildung 2.21: Felder der Tabelle Adressen auswählen

Sollen alle Felder ausgewählt werden, können Sie auch die erste Option * (ALLE SPALTEN) setzen. Das Sternchen steht für alle Felder (intern wird eine SQL-Anweisung erzeugt).

Nach einem Klick auf OK und FERTIGSTELLEN wird der neue Datenadapter erzeugt. Unterhalb der Webform-Arbeitsfläche werden dann diese beiden Objekte angezeigt:

► **OLEDBDATAADAPTER1**

Dies ist der Datenadapter, den Sie soeben erzeugt haben.

► **OLEDBCONNECTION1**

Diese Datenverbindung stellt die Verbindung zur Access-Datenbank *Shop.mdb* her und wird vom Datenadapter benutzt.

DataSet erzeugen

Um ein *DataSet* zu erzeugen, gehen Sie wie folgt vor:

1. Klicken Sie im Designer mit der rechten Maustaste auf den angelegten *DataAdapter* und wählen Sie aus dem dann erscheinenden Kontextmenü den Punkt **DATASET GENERIEREN**.

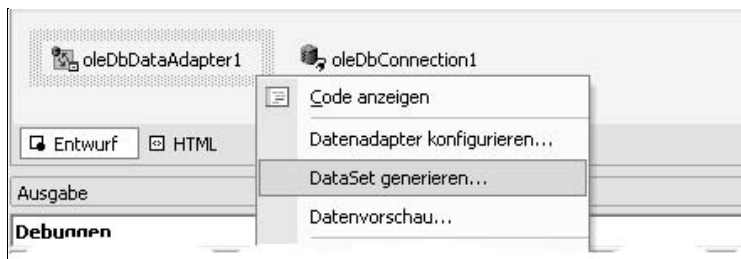


Abbildung 2.22: Erzeugen eines DataSets

2. Es erscheint daraufhin ein Dialogfenster, welches Sie für dieses Beispiel mit einem Klick auf OK bestätigen und schließen können.



Abbildung 2.23: Bestätigen der Einstellungen zum DataSet

Damit erscheint dieses *DataSet* im Designer als drittes Objekt am unteren Bildschirmrand.

Jetzt benötigen Sie noch ein **DATAGRID**-Steuerelement, mit dem Sie die Datensätze auf der Seite anzeigen können. Gehen Sie dazu wie folgt vor:

**DataGrid
erzeugen**

1. Wählen Sie unter **TOOLBOX** aus dem Bereich **WEBFORMS** den Eintrag **DATAGRID** und ziehen Sie diesen auf die Arbeitsfläche des Designers.
2. Verknüpfen Sie dann das neue *DataGrid* mit dem *DataSet*. Klicken Sie dazu im **Eigenschaften-Fenster** des *DataGrids* auf die Option **DATASOURCE**.

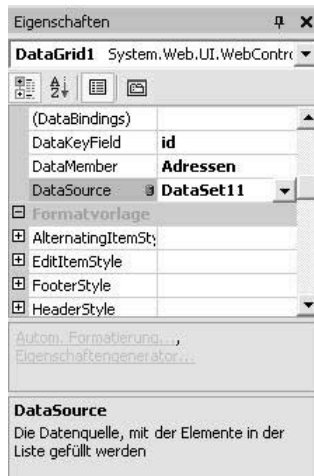


Abbildung 2.24: DataGrid mit dem DataSet verbinden

3. Wählen Sie dann im Feld DATAMEMBER die Tabelle *Adressen* und setzen Sie das DATAKEYFIELD auf *id*.

Wenn Sie nun das Programm starten, werden Sie feststellen, dass zunächst nur eine leere Website erzeugt wird. Das liegt daran, weil das *DataSet* noch nicht mit Daten gefüllt worden ist. Dies müssen Sie per Hand in der Methode *Page_Load* eintragen. Fügen Sie hier die folgenden zwei Zeilen ein:

```
oleDbDataAdapter1.Fill(dataSet11)
DataGrid1.DataBind()
```

Mit der ersten Anweisung füllen Sie das *DataSet* über den Datenadapter mit Daten. Gezeigt wird hier die einfachste Art des Aufrufs. Bei komplexeren *DataSets* mit mehr als einer Tabelle können Sie als zweite Option die gewünschte Tabelle angeben.

Mit der zweiten Programmzeile werden die Daten an das *Data Grid*-Steuerelement gebunden. Dies erfolgt über die bereits festgelegte Verknüpfung zum *DataSet*.

Wenn Sie nun das Programm starten, sollten Sie in etwa die folgende Ausgabe im Browser sehen:

name	ort	plz	strasse	id
Jörg Krause	Berlin	12683	Dornenweg 33	1
Lukas Werlich	München	89487	Spielallee 11	3
Carolin Scholz	Düsseldorf	40748	Eichenwall 4	4

Abbildung 2.25: Ausgabe der Datensätze

Editieren von Datensätzen

Nachdem Sie gesehen haben, wie einfach es ist, Datensätze auf dem Bildschirm anzuzeigen, wäre es natürlich interessant zu wissen, wie Sie diese direkt in dieser Anzeige ändern können. Die erforderlichen Einrichtungsschritte sind größtenteils im Visual Studio mit grafischen Mitteln realisierbar. Hinzu kommt ein Minimum an manuell einzutragenden Codezeilen. Ziel ist eine kleine Anwendung, über die die kleine Adressdatenbank aus dem Beispiel des vorhergehenden Abschnitts bearbeitet werden kann. Dabei werden die Datensätze in einer optisch ansprechenden Tabellenform auf dem Bildschirm präsentiert. Die Elemente jeder Zeile können über einen Klick auf eine BEARBEITEN-Schaltfläche in den Editiermodus versetzt und dann bearbeitet werden. Dabei wird diese Schaltfläche durch zwei andere Schaltflächen ersetzt, die dann das Beenden des Editierens mit Aktualisierung der Daten oder das Abbrechen ermöglichen.

Name	Strasse	Postleitzahl	Ortsname		
Jörg Krause	Dornenweg 33	12683	Berlin	Bearbeiten	
Lukas Werlich	Spielallee 11	89487	München	Aktualisieren	Abbrechen
Carolin Scholz	Eichenwall 4	40748	Düsseldorf	Bearbeiten	
Uwe Bönning	Grafenplatz 11	12345	Berlin	Bearbeiten	
Clemens Krause	Dornenweg 33	12683	Berlin	Bearbeiten	

Abbildung 2.26: Ziel: Anwendung zum Editieren einer Datenbank-Tabelle

Dieses einführende Beispiel in die etwas komplexere Materie der Datenbankprogrammierung ist bewusst einfach² gehalten worden. So werden hier nur die Daten einer einzigen Tabelle geändert. Das hat seine Ursache in der Funktionsweise der `DATAGRID`-Methode. Diese ist so konzipiert, dass mit ihr Anzeigen von Daten, beispielsweise auch aus Datenbanken, sehr schnell und durchaus optisch ansprechend mit nur wenig Programmieraufwand erstellt werden können. Etwas anders sieht es aus, wenn Sie diese Daten auch zum Bearbeiten anbieten wollen. Die Aktualisierung der Quelldatenbank muss dann weitgehend *zu Fuß* programmiert werden. Für einfache Strukturen ist dies trotzdem nicht sehr kompliziert, wie Sie mit diesem Beispiel sehen werden. Umfassender wird das Thema Datenbankprogrammierung in den Kapiteln 5, »Grundlagen der Datenspeicherung« ab Seite 345 und 9, »Datenbanken und ADO.NET« ab Seite 823 behandelt.

**Schritt 1:
DataGrid
anpassen**

In einem ersten Schritt sollten Sie das *DataGrid* so anpassen, dass nur noch die Spalten angezeigt werden, die für den Benutzer sichtbar sein sollen. Hinzu kommt eventuell eine Umsortierung. Für das folgende Beispiel sollen diese Spaltenbezeichnungen in genau dieser Reihenfolge im Gitter stehen:

1. Name
2. Straße
3. Postleitzahl
4. Ort

Dies sind auch die richtigen Bezeichner der Tabellenspalten – schließlich sieht es nicht besonders schön aus, wenn nur Kurzbezeichner wie *str* oder *plz* erscheinen.

**Eigenschaften-
Generator**

Sie richten die Struktur und das Aussehen des *DataGrids* im Eigenschaften-Generator ein. Diesen starten Sie, wenn Sie auf das *DataGrid* im Webforms-Designer mit der rechten Maustaste klicken und aus dem Kontextmenü den gleichnamigen Eintrag wählen.

² Auch wenn es nicht einfach erscheint: Gemessen an der erreichten Funktionalität ist diese Vorgehensweise sehr viel einfacher als in der konventionellen Programmierung.



Abbildung 2.27: Eigenschaften-Generator mit allgemeinen Einstellungen zum DataGrid

In der Registerkarte ALLGEMEIN sollten für dieses Beispiel nur die in Abbildung 2.27 gezeigten Einträge aktiv sein. Unter DATASOURCE muss das *DataSet* eingetragen sein, welches Sie auch im vorhergehenden Beispiel schon erstellt und verwendet haben. Zusätzlich markieren Sie HEADER ANZEIGEN. Damit erscheinen die Spaltenbezeichner der Tabelle. Nicht aktiviert sein darf für dieses Beispiel das Kontrollkästchen vor FOOTER ANZEIGEN. Der Footer (Fußzeile) ist das Pendant zur Tabellenüberschrift und wird als letzte Zeile angezeigt, wenn er aktiviert ist.

Über die Optionen der Registerkarte SPALTEN richten Sie dann die anzuzeigenden Spalten sowie weitere Tabellenelemente ein.

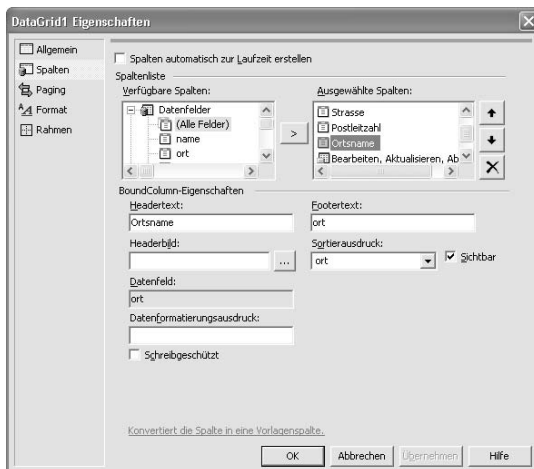


Abbildung 2.28: Spalten benutzerdefiniert einrichten

Deaktivieren Sie zuerst das Kontrollkästchen vor SPALTEN AUTOMATISCH ZUR LAUFZEIT ERSTELLEN. Markieren Sie dann unter VERFÜGBARE SPALTEN die Spalten, die in der Tabelle erscheinen sollen. Sortieren Sie diese über die Pfeil-Schaltflächen neben AUSGEWÄHLTE SPALTEN so um, dass die eingangs beschriebene Reihenfolge erreicht wird. Vergeben Sie für jede Spalte einen HEADERTEXT und einen FOOTERTEXT. Der HEADERTEXT ist dabei beliebig wählbar.



Für das Funktionieren dieses Beispiels ist es wichtig, dass Sie als FOOTERTEXT für jede Spalte den exakten Spaltennamen der Datenbank-Tabelle Adressen eintragen. Dies ist genau genommen ein Trick, damit für die manuell zu programmierende Aktualisierung der Datenbank nicht zu viele Codezeilen benötigt werden und das Beispiel zu komplex wird.

Schaltflächen hinzufügen

Jetzt fehlen noch die Schaltflächen, über die der Besucher der Website in den Editiermodus für die Zeile gelangen kann. Diese finden Sie weiter unten in der Liste unter VERFÜGBARE SPALTEN. Wählen Sie dazu unter SCHALTFLÄCHENSPALTE den Eintrag BEARBEITEN, AKTUALISIEREN, ABBRECHEN aus. Es erscheint dann pro Spalte eine Schaltfläche BEARBEITEN.

Eine optisch etwas ansprechendere Formatierung können Sie für Ihr DataGrid in der Registerkarte FORMAT auf sehr einfache Art erstellen. Für das vorliegende Beispiel wurden Anpassungen in den Rubriken HEADER und ELEMENTE vorgenommen.

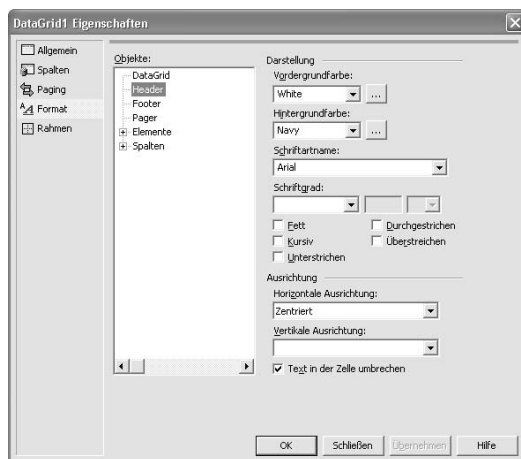


Abbildung 2.29: Optisch ansprechendere Formatierungen des DataGrids können hier einfach erstellt werden.

Auf weitere Gestaltungsmöglichkeiten wie das Umblättern über PAGING-Einstellungen oder alternative Rahmenformen soll hier nicht weiter eingegangen werden.

Im Ergebnis der Anpassungen im ersten Schritt sollte das *DataGrid* im Webforms-Designer etwa folgendes Aussehen haben:

Name	Strasse	Postleitzahl	Ortsname	
abc	abc	abc	abc	Bearbeiten
abc	abc	abc	abc	Bearbeiten
abc	abc	abc	abc	Bearbeiten
abc	abc	abc	abc	Bearbeiten
abc	abc	abc	abc	Bearbeiten

Abbildung 2.30: Aussehen des angepassten DataGrids

Im zweiten Schritt erstellen Sie die Methoden für die Behandlung der Ereignisse, die beim Klick des Benutzers auf die Schaltflächen im *DataGrid* ausgelöst werden. Gehen Sie dazu wie schon seit vielen Jahren unter Visual Basic üblich in den Code-Editor zurück und dann wie folgt vor:

1. Wählen Sie im Objekt-Inspektor das entsprechende *Datagrid*-Objekt aus.
2. Wählen Sie dann nacheinander die Ereignisse *CancelCommand*, *EditCommand* und *UpdateCommand* aus.

Schritt 2:
Ereignisbehand-
lungs-Methoden

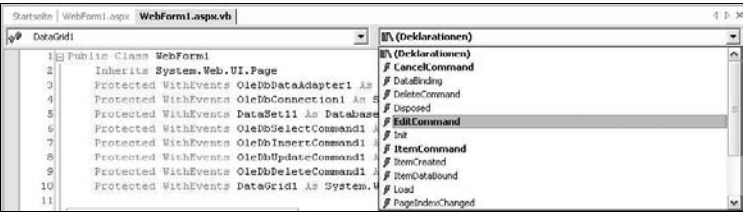


Abbildung 2.31: Auswahl eines Ereignisses zur Erzeugung der Ereignisbehandlungs-
methode

Als Ergebnis erhalten Sie automatisch die Deklarationen für die entsprechenden Ereignisbehandlungsmethoden. Diese sind noch leer und müssen dann per Hand mit Code versehen werden.

```
Private Sub DataGridView1_EditCommand(ByVal source As Object,   
                                     ByVal e As DataGridViewCommandEventArgs)   
    Handles DataGridView1.EditCommand   
   
        DataGridView1.EditItemIndex = e.Item.ItemIndex   
        DataGridView1.DataBind()   
End Sub   
   
Private Sub DataGridView1_UpdateCommand(ByVal source As Object,   
                                         ByVal e As DataGridViewCommandEventArgs)   
    Handles DataGridView1.UpdateCommand   
   
        Dim i As Integer   
        Dim tbControl As TextBox   
        Dim colName As String   
        For i = 0 To e.Item.Cells.Count - 2   
            tbControl = CType(e.Item.Cells(i).Controls(0), TextBox)   
            colName = DataGridView1.Columns(i).FooterText   
            DataSet11.Tables("Adressen")   
                .Rows(e.Item.ItemIndex)(colName)   
                = tbControl.Text   
        Next   
        OleDbDataAdapter1.Update(DataSet11)   
        DataGridView1.EditItemIndex = -1   
        DataGridView1.DataBind()   
End Sub   
   
Private Sub DataGridView1_CancelCommand(ByVal source As Object,   
                                         ByVal e As DataGridViewCommandEventArgs)   
    Handles DataGridView1.CancelCommand   
   
        DataGridView1.EditItemIndex = -1   
        DataGridView1.DataBind()   
End Sub
```

*Listing 2.3: Die fertigen Ereignisbehandlungs-Methoden für das Beispiel
(für den Druck etwas formatiert)*

**Wie es
funktioniert**

Bis auf die Methode für die Behandlung des Update-Ereignisses sind die manuell zu erstellenden Codebestandteile sehr einfach. Der in allen drei Methoden vorhandene Aufruf von `DataGridView1.DataBind()` baut das *DataGrid* neu auf und übergibt es an die Seite. Abhängig davon, welche Zeile selektiert ist, wird diese zum Editieren angeboten. In der Methode `DataGridView1_CancelCommand` wird

der Index `DataGrid1.EditItemIndex` auf den Wert `-1` und damit außerhalb des *DataGrids* gesetzt. Somit ist keine Zeile zum Editieren hervorgehoben. Das ist auch die Voreinstellung beim ersten Aufruf der Seite.

Klickt der Benutzer auf eine der Schaltflächen BEARBEITEN, erfolgt der Aufruf von `DataGrid1_EditCommand`. In dieser Methode wird der Index auf den übergebenen Wert `e.Item.ItemIndex` gesetzt und damit die betreffende Zeile zum Editieren angeboten.

Wie bereits erwähnt, ist die Aktualisierung der Quell-Datenbank für das *DataGrid* nicht trivial, da eine derartige Rückkopplung hier leider nicht vorgesehen ist. Für das vorliegende einfache Beispiel mit nur einer Access-Datenbank-Tabelle, bei der auch noch alle Spalten vom gleichen Typ `TEXT` sind, werden mit einer ersten *For*-Schleife alle Felder des *DataSets* `dataSet11` mit den Feldern der betreffenden Reihe im *DataGrid* überschrieben. Eine Prüfung, welches Feld in der Reihe wirklich geändert worden ist, erfolgt hierbei zur Vereinfachung nicht. In der ersten Zeile innerhalb der Schleife wird einer Variablen `tbControl` vom Typ `TextBox` jeweils der Inhalt der über den Index `i` ausgewählten Zelle des *DataGrids* zwischengespeichert. Das funktioniert übrigens nur dann, wenn auch alle Eingabefelder den Typ `TextBox` aufweisen!

In der zweiten Zeile innerhalb der Schleife werden die jeweiligen Spaltennamen passend zum Index `i` im *DataGrid* ermittelt und in `colName` zwischengespeichert. Damit diese zur Vereinfachung mit den Spaltennamen der hier verwendeten Access-Tabelle *Adressen* übereinstimmen, wurden sie als `FOOTERTEXT` im *DataGrid* »versteckt« (siehe Abbildung 2.28). In der letzten Programmzeile der *For*-Schleife wird `colName` dann zur Identifikation der zu ändernden Spalte im *DataSet* verwendet. Sie können natürlich statt des Umwegs über `colName` mit Indizes arbeiten. Die erste Spalte in der Tabelle *Adressen* mit der Spalte *name* können Sie dann folgendermaßen adressieren:

```
dataSet11.Tables("Adressen").Rows(e.Item.ItemIndex)(1)
```

Allerdings sollten Sie die Aktualisierung des *DataSets* über eine *For*-Schleife nur dann einsetzen, wenn die Spalten im *DataGrid* und in der Tabelle in der gleichen Reihenfolge vorliegen.

Über `oleDbDataAdapter1.Update(DataSet11)` wird abschließend ein Aktualisierungsbefehl via Datenadapter an die Datenbank gesendet. Dabei werden die entsprechenden Datensätze in der Daten-

bank mit den bisher abgekoppelt gehaltenen Datensätze im *DataSet11* abgeglichen.



Damit das Beispiel wie gezeigt mit Schaltflächen im DataGrid funktionieren kann, ist noch eine Anpassung der Page_Load-Methode nötig:

```
Private Sub Page_Load(ByVal sender As System.Object,   
                      ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
    OleDbDataAdapter1.Fill(DataSet11)  
    If Not Page.IsPostBack Then  
        DataGrid1.DataBind()  
    End If  
End Sub
```

Klammern Sie den Aufruf von DataGrid1.DataBind() in die If-Bedingung wie oben gezeigt ein. Das ist nur dann erforderlich, wenn Sie im DataGrid Schaltflächen verwenden wollen. Die einfacheren, aber optisch nicht so ansprechenden Links erfordern dies nicht.

2.4 Fehlersuche und Debugging

Mit steigender Größe eines Projekts nehmen das Testen und die Fehlersuche immer mehr Raum ein. Damit steigt auch die Bedeutung professioneller Werkzeuge. Nur mit Texteditor, Webserver und Browser ausgestattet wird Ihre Produktivität, was möglichst fehlerfreien und stabilen Code angeht, sicher endlich sein. Sehr große und komplexe Projekte werden ohne professionelle Werkzeuge für die Fehlersuche und den Test sicherlich kaum zu realisieren sein. Mit ASP.NET halten im Vergleich zu ASP revolutionäre Neuerungen in die Welt der Webentwickler Einzug. Da ist das Visual Studio, mit dem Sie ASP.NET-Anwendungen ähnlich komfortabel wie normale Windows-Programme entwickeln können. Eine zweite, nicht minder wichtige Komponente ist der integrierte Debugger. Dieser wird in Abschnitt 2.4.3, »Der Visual Studio-Debugger im Detail« ab Seite 134 näher vorgestellt.

Kostenlose Alternative

Von Microsoft selbst wird mit dem kostenlosen Framework SDK der SDK-Debugger angeboten. Dieser bietet nicht den Funktionsumfang des großen Bruders aus dem Visual Studio, genügt aber durchaus gehobenen Ansprüchen. In der MSDN-Referenz finden Sie dazu ausreichend Informationen.

2.4.1 Fehlerarten

Bevor Sie sich dem professionellen Umgang mit dem Debugger widmen, sollte Ihnen klar sein, welche Art Fehler prinzipiell auftreten können und ob, beziehungsweise wie, der Debugger zu deren Behebung gute Dienste zu leisten vermag.

Syntaxfehler

Mit Syntaxfehlern sind alle Fehler gemeint, die bereits im Code Designer erkannt werden und einen Start beziehungsweise das Kompilieren des Programms verhindern. Im Visual Studio werden Syntaxfehler mit einer blauen Schlangenlinie gekennzeichnet.

Mit dem Visual Studio und seiner interaktiven Eingabehilfe *IntelliSense* ist es fast unmöglich, Syntaxfehler zu produzieren. Wenn Ihnen diese doch unterlaufen, werden Sie sofort und unmissverständlich darauf hingewiesen. Sie sollten auf jeden Fall immer die AUFGABENLISTE, die sich standardmäßig am unteren Rand mit dem Informationsfenster AUSGABE des Compilers befindet, im Blick behalten.

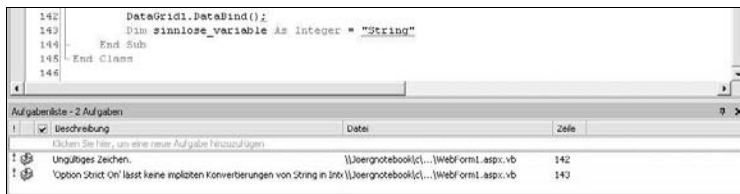


Abbildung 2.32: Ein Blick in die Aufgabenliste, Kategorie Aufgaben oder Buildfehler, hilft immer.

In dieser Liste finden Sie Ihre Versäumnisse und Nachlässigkeiten im Klartext.

Die Einträge in der Aufgabenliste, insbesondere die, die den ungenutzten Ressourcen zugeordnet sind, werden nicht immer sofort aktualisiert. Über ein schnelles Neukompilieren des Programms (über den Hauptmenüpunkt ERSTELLEN) werden diese wieder auf den neuesten Stand gebracht.



Compilerfehler

Auch wenn syntaktisch alles richtig ist, kann es zu einem Compilerfehler kommen. So können Sie beispielsweise eigene Methodenbezeichner verwenden, ohne dass das hilfreiche *Intellisense* im Studio daran Anstoß nimmt. Wenn Sie dann allerdings vergessen, Ihre Methode auch zu deklarieren, wird zwangsläufig der Compiler darüber stolpern. Im Studio wird eine Fehlermeldung wie in nachfolgender Abbildung gezeigt erscheinen, wenn Sie versuchen, das Programm zu starten.

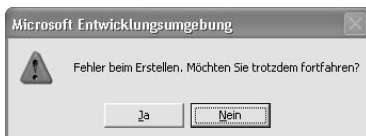


Abbildung 2.33: Anzeige eines Compilerfehlers im Studio

Wenn Sie außerhalb des Visual Studios einen fehlerhaften Quelltext versuchen zu starten, erhalten Sie eine eindeutige Fehlermeldung über den Browser.

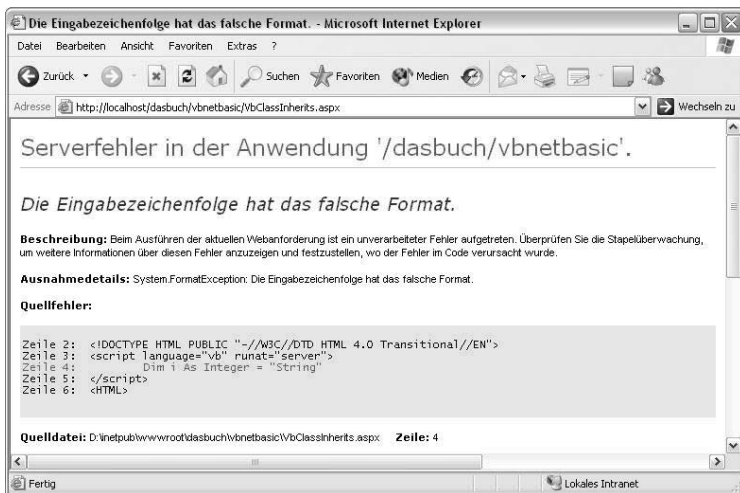


Abbildung 2.34: Compilerfehlermeldung im Browser

Details nur unter localhost

Im Fenster des Browsers bekommen Sie ebenfalls detaillierte Informationen zur Art des Fehlers und dem vermutlichen Ort der Ursache. Über weitere Links am unteren Ende der Ausgabe kön-

nen Sie zusätzliche Detailinformationen abrufen. Alle diese Informationen erhalten Sie übrigens nur, wenn Sie die Seite über den lokal installierten Webserver abrufen. Außerhalb gibt es für Benutzer nur allgemeine Laufzeitfehler-Informationen, natürlich ohne Herausgabe von Quelltext-Teilen. Diese Fehlermeldungen können Sie anpassen, sodass Besucher der Website zur Not mit einer »vernünftigen« Fehlermeldung bedient werden können.

Laufzeitfehler

Kann ein Programm kompiliert werden, heißt das noch lange nicht, dass es auch stets stabil und fehlerfrei funktioniert. Fehler, die jetzt, während der normalen Laufzeit auftreten, werden deshalb *Laufzeitfehler* genannt. Sie sind besonders schwer zu lokalisieren. Damit schlägt die Stunde der Debugger. Dieser kann beispielsweise ein Programm schrittweise abarbeiten oder Ihnen Informationen über den Inhalt von Variablen liefern. Da selten das gesamte Programm Schritt für Schritt abgearbeitet werden soll, können Sie so genannte *Haltepunkte* definieren, ab denen der Debugger mit der interaktiven Schrittfolge durch den Code beginnen soll.

**Schwer zu finden:
Debugger helfen**

Damit ein Programm mit der Hilfe eines Debuggers bearbeitet werden kann, muss der Compiler zusätzlichen Code, auch *Debug-Code* genannt, im Programm unterbringen. Das können Sie über entsprechende Einstellungen im Visual Studio, in Konfigurationsdateien oder im Quellcode selbst steuern. Informationen zur Konfiguration finden Sie im nachfolgenden Abschnitt.

**Voraussetzungen
zum Debuggen**

2.4.2 Das Debuggen von Anwendungen aktivieren

Wie im vorhergehenden Abschnitt bereits erwähnt, benötigt ein Debugger zusätzliche Informationen im Code, damit er die direkte Verbindung zum Quellcode herstellen kann. Nur so ist es beispielsweise möglich, dass der Debugger genau an einer bestimmten Programmzeile an einem Haltepunkt mit der schrittweisen Abarbeitung beginnen oder den Inhalt von Variablen, die Sie ihm im Klartext angeben, ausgeben kann. Die Debug-Möglichkeit können Sie für Ihre Anwendungen an verschiedenen Stellen definieren.

Debuggen im Visual Studio

Im Studio können Sie die Debug-Möglichkeit auf diesen Ebenen einstellen:

► *Global im Studio*

Im Studio finden Sie in der Standard-Symbolleiste ein Auswahlmenü, über welches Sie einfach und schnell zwischen der Verarbeitung mit und ohne Debugging umschalten können.

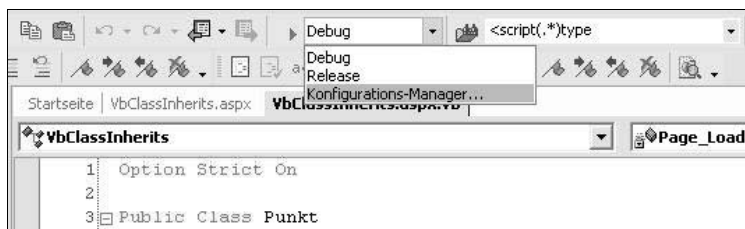


Abbildung 2.35: Umschalten der Verarbeitungsart im Studio

Über dieses Menü haben Sie ebenfalls den schnellen Zugriff auf den Konfigurations-Manager, über den Sie für umfangreichere Projekte mit mehreren Projektdateien diese Optionen einstellen und definieren können.

Tasten- kombinationen

Über die Tastenkombination **[Strg] + [F5]** starten Sie das Programm explizit ohne Debuggen, mit **[F5]** dagegen immer mit Debugger.

► *Projektmappe*

Im Eigenschaften-Fenster zu einer Projektmappe finden Sie unter KONFIGURATIONSEIGENSCHAFTEN den entsprechenden Punkt, der für jedes einzelne Projekt angezeigt wird. Mit Hilfe vorgefertigter Konfigurationen können Sie dies global umschalten (Auswahlmenü KONFIGURATION).

► *Einzelnes Projekt*

Für ein einzelnes Projekt finden Sie den entsprechenden Punkt im dazugehörigen Eigenschaftenfenster.

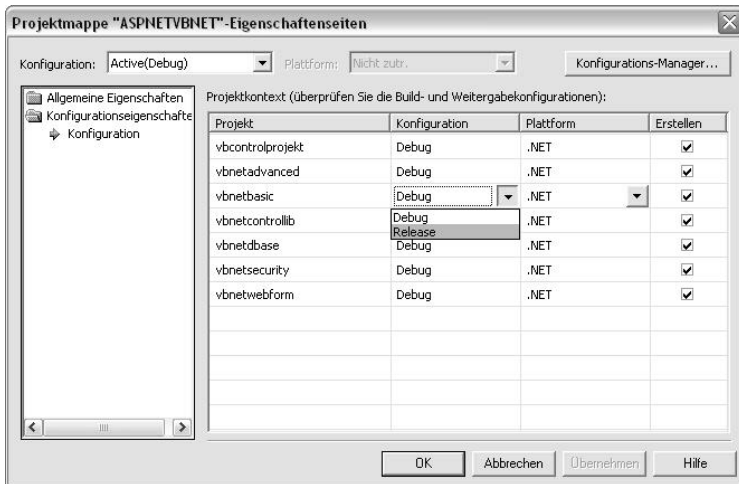


Abbildung 2.36: Debug-Konfiguration für die Projekte einer Projektmappe einstellen

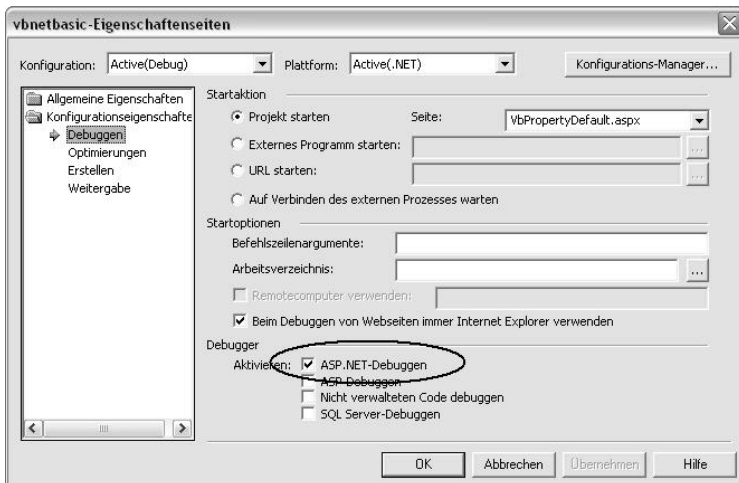


Abbildung 2.37: Debug-Einstellungen für ein Projekt

Wichtig ist, dass Sie für das Debuggen von ASP.NET-Code das in Abbildung 2.37 markierte Kontrollkästchen aktivieren. Zu den weiteren Optionen, die beispielsweise das Debuggen von ASP-Anwendungen oder nicht verwaltetem Code betreffen, finden Sie weiterführende Informationen in der Online-Hilfe.

Auf Ebene der IIS-Verzeichnisse

Web.config Unterhalb des IIS-Verzeichnisses `\Inetpub\wwwroot\` finden Sie standardmäßig die Verzeichnisse Ihrer Anwendungen. Hier können Sie das Debugging in der Datei *Web.config* ein- und ausschalten.

Innerhalb von ASPX-Dateien

Innerhalb Ihrer ASPX-Dateien können Sie die Debug-Option im Rahmen der `@Page`-Direktive definieren:

```
<% @Page Debug="True"%>
```

2.4.3 Der Visual Studio-Debugger im Detail

Durch die Integration in das Visual Studio werden Sie bereits mit den ersten Schritten in diesem neuen Entwicklungssystem vom Debugger begleitet. Wenn Sie die Debugging-Möglichkeiten (oder besser: *Unmöglichkeiten?*) von ASP kennen sollten, werden Sie sicherlich angenehm von den umfassenden Möglichkeiten dieser Lösung überrascht sein. Nachfolgend werden die wichtigsten Verfahren und Möglichkeiten erklärt, die Sie beim Debuggen von ASP.NET-Anwendungen brauchen werden.

Haltepunkte setzen

Klick in Rahmen Um einen Haltepunkt zu setzen, ab dem der Debugger mit der schrittweisen Abarbeitung des Codes beginnen soll, klicken Sie einfach in den grauen Rahmen links neben dem Code an die gewünschte Stelle. Der Debugger wird dann genau vor der markierten Zeile einsetzen und das Programm stoppen.

Es gibt noch weitere Möglichkeiten, Haltepunkte zu setzen:

- Gehen Sie mit dem Cursor zu der gewünschten Stelle und betätigen Sie die Tastenkombination `Strg + B`. Es erscheint ein Dialogfenster, in welchem Sie weitere Optionen zum Haltepunkt definieren können (siehe weiter unten im Text). Allerdings müssen Sie bei dieser Art der Haltepunkt-Definition alle relevanten Einstellungen wie die genaue Position im Quelltext erst noch mitteilen.

- Klicken Sie mit der rechten Maustaste in den linken grauen Rahmen und wählen Sie aus dem Kontextmenü NEUER HALTEPUNKT. Hier erscheint ebenfalls wie eben beschrieben das Dialogfenster mit weiteren Optionen.

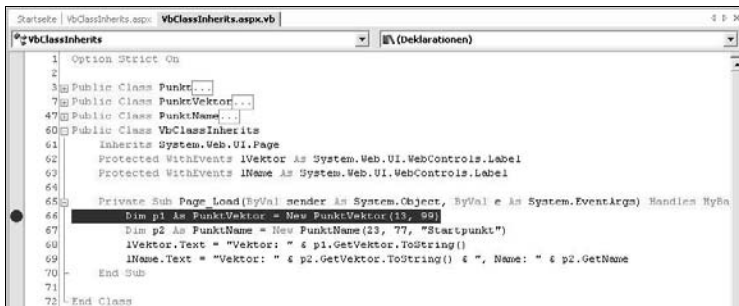


Abbildung 2.38: Ein gesetzter Haltepunkt im Quellcode

- Wählen Sie aus dem Kontextmenü am Rand den Punkt HALTEPUNKT EINFÜGEN.

Zu einem Haltepunkt können Sie weitere Optionen einstellen und damit die Verarbeitung durch den Compiler beeinflussen. Für einen bestehenden Haltepunkt erhalten Sie dieses Fenster über einen Klick mit der rechten Maustaste auf die Markierung im Quelltext und die Wahl des Punktes HALTEPUNKTEIGENSCHAFTEN.

Weitere Optionen



Abbildung 2.39: Eigenschaften eines Haltepunktes

Interessant sind in diesem Fenster vor allem die Möglichkeiten, Bedingungen zu definieren oder anzugeben, nach wie vielen Durchläufen der Debugger das Programm stoppen soll. So können Sie beispielsweise innerhalb von Schleifen die Werte überwachen, wenn eine bestimmte Anzahl von Durchläufen erreicht worden ist.

Haltepunkte deaktivieren

Wenn Sie verschiedene Haltepunkte in einem größeren Projekt gesetzt haben und dieses zu Testzwecken zwischendurch einmal ohne Einsatz des Debuggers laufen lassen wollen, können Sie temporär auch alle Haltepunkte deaktivieren. Diese gehen dabei nicht verloren. Wählen Sie dazu im Hauptmenü den Punkt **DEBUGGEN | ALLE HALTEPUNKTE DEAKTIVIEREN**.

Haltepunkte löschen

Zum Löschen aller Haltepunkte wählen Sie im Hauptmenü den Punkt **DEBUGGEN | ALLE HALTEPUNKTE LÖSCHEN**. Einzelne Punkte löschen Sie, indem Sie einfach auf den Punkt im Rahmen links klicken.

Schrittweise Abarbeitung steuern

Wenn Sie im Debug-Modus arbeiten, können Sie sich mit Hilfe dieser Methoden durch die Verarbeitung des Programms bewegen:

► *Einzelschritt*

F11

Über einen Druck auf die Funktionstaste **F11** bewegen Sie sich schrittweise durch das Programm, beginnend beim ersten Haltepunkt. Dabei wird auch einzelschrittweise in alle aufgerufenen Methoden hinein verzweigt.

► *Prozedurschritt*

F10

Über **F10** gehen Sie schrittweise durch die Methode, beginnend am ersten Haltepunkt. In Methoden, die von hier aus aufgerufen werden, wird nicht verzweigt. Diese werden wie ein einziger Schritt behandelt und komplett abgearbeitet. Damit sparen Sie sich, Code zu debuggen, der nichts mit Ihrem Problem zu tun hat.

Anzeige der Werte von Variablen und Eigenschaften

Richtig interessant wird das Debuggen, wenn Sie während der Abarbeitung die für Sie wichtigen Werte von Variablen oder Eigenschaften direkt einsehen können.

Die einfachste und schnellste Möglichkeit, den Wert einer Variablen zu erfahren, besteht darin, im Debug-Modus den Mauszeiger in der Codeansicht darüber schweben zu lassen.



Dazu haben Sie diese weiteren Möglichkeiten:

► Schnellüberwachung

Über das Hauptmenü **DEBUGGEN | SCHNELLÜBERWACHUNG** oder die Tastenkombination **[Strg] + [Alt] + [Q]** öffnen Sie das Dialogfenster für die Konfiguration des Überwachungsfensters. Klicken Sie hier auf **HINZUFÜGEN**, um bestimmte Werte zu überwachen. Das eigentliche Überwachungsfenster erscheint erst nach Abschluss der Konfiguration – standardmäßig am unteren Bildschirmrand – und gibt dann die Werte der zu überwachenden Objekte aus. Sie können in dieses Überwachungsfenster auch direkt die Werte eingeben.

[Strg+Alt+Q]

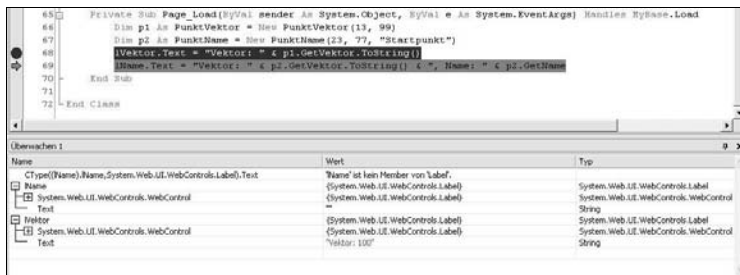


Abbildung 2.40: Überwachungsfenster mit Ausgabe von Werten und Manipulationsmöglichkeit

Ist ein Objekt außerhalb seines Gültigkeitsbereichs, wird auch dies angezeigt. Elemente löschen Sie aus dieser Liste, indem Sie dieses mit der Maus markieren und dann auf **[Entf]** drücken.

Sie können Werte direkt im Überwachungsfenster ändern. Diese werden dann sofort in das laufende Programm übernommen. So können Sie beispielsweise testen, wie Ihr Programm auf bestimmte Werte reagiert oder Sie kürzen Schleifen ab, indem Sie den Index einfach erhöhen.



► *Befehlsfenster*

Strg+Alt+A

Das Fenster für die Schnellüberwachung zeigt zwar alle gewünschten Werte an, kann aber ebenso schnell unübersichtlich werden. Für die einfache Einsicht in bestimmte Werte kann deshalb das Befehlsfenster ergänzend oder alternativ interessant sein. Öffnen Sie dieses über die Tastenkombination **Strg** + **Alt** + **A** oder das Menü **ANSICHT | ANDERE FENSTER | BEFEHLSFENSTER**.



Abbildung 2.41: Das Befehlsfenster

Im Wesentlichen wird hier zwischen einem Befehls- und einem unmittelbaren Modus unterschieden. Der Befehlsmodus ist durch das Zeichen **>** als Prompt gekennzeichnet. Um den Wert eines Objekts anzuzeigen, geben Sie ein Fragezeichen, gefolgt von einem Leerzeichen und dem Bezeichner ein. Mit zwei Fragezeichen wird der Bezeichner direkt in die Schnellüberwachung übernommen. Ansonsten dient dieser Modus der Angabe von Befehlen für Visual Studio .NET, wie sie auch über das Menüsystem ausgelöst werden können.

Um in den unmittelbaren Modus zu wechseln, geben Sie das Kommando **immed** ein. Danach können Sie die Bezeichner direkt eingeben. Zum Befehlsmodus wechseln Sie zurück über die Eingabe von **>cmd**. Weitere Informationen erhalten Sie, wenn Sie im Befehlsmodus **help** eingeben (oder **>help** im un-

mittelbaren Modus). Auch im unmittelbaren Modus werden Ausgaben durch ein »?-«-Zeichen eingeleitet. Änderungsausdrücke lassen sich direkt eingeben.

Weitere Befehle sind im Befehlsfenster möglich. Geben Sie hier `>alias` ein, um eine Liste der aktuellen Aliase und ihrer Definitionen anzuzeigen. Geben Sie `>cls` ein, um den Inhalt des Fensters BEFEHL zu löschen. Die History-Liste, die die letzten Eingaben speichert, bleibt dabei erhalten. Drücken Sie auf  um zu älteren Eingaben zu gelangen.

**Befehlsfenster
löschen mit `>cls`**

Debuggerprobleme

Manchmal hat der Debugger selbst Probleme und will nicht starten. Generell sollten Sie auf Folgendes achten:

- ▶ Sie müssen Mitglied der Gruppe »Debuggerbenutzer« sein, wenn Sie den Worker Process debuggen sollen (das ist die ausführende Instanz für ASP.NET).
- ▶ Das Debugging selbst muss natürlich aktiviert sein.
- ▶ Das virtuelle Verzeichnis im IIS sollte anonym erreichbar sein und die Windows-Authentifizierung unterstützen.
- ▶ Setzen Sie die Sicherheitseinstellungen in der Konfiguration des Internet Explorers in den Zonen Intranet und Internet auf die Standardwerte. Insbesondere müssen Sie die standardmäßige Cookiebehandlung und ActiveX-Scripting zulassen.
- ▶ Prüfen Sie, ob die Anmeldeeinstellungen im Internet Explorer auf automatische Anmeldung stehen. Gehen Sie dazu folgendermaßen vor:
 1. Öffnen Sie die INTERNETOPTIONEN.
 2. Wechseln Sie zur Registerkarte SICHERHEIT.
 3. Wählen Sie LOKALES INTRANET, oder wenn Sie das Projekt so erstellt haben, dass der vollständig qualifizierte Name des Servers verwendet wird, die Stufe INTERNET.
 4. Klicken Sie jeweils auf STUFE ANPASSEN.
 5. Suchen Sie den Zweig BENUTZERAUTHENTIFIZIERUNG. Wählen Sie unter ANMELDUNG die Option AUTOMATISCHE ANMELDUNG MIT AKTUELLEN BENUTZERNAMEN UND KENNWORT.

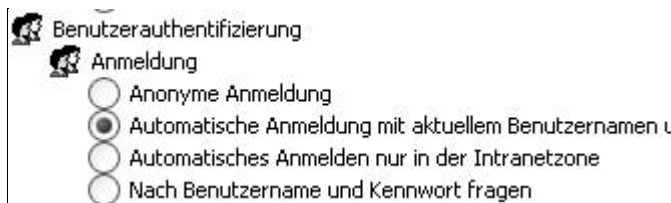


Abbildung 2.42: Eine Ursache der Debuggerprobleme sind IE-Einstellungen

- Wenn Sie die Maschine mit dem »IIS Lock Down Tool« geschützt haben, ergänzen Sie in der Konfigurationsdatei *urlscan.ini* Folgendes:

Im Abschnitt [AllowVerbs], fügen Sie `DEBUG` hinzu. Unter [AllowExtensions] ergänzen Sie `.aspx` und `.asmx`. Der IIS muss danach neu gestartet werden. Geben Sie dazu an der Eingabeaufforderung `C:\>iisreset` ein.

- Installieren Sie ASP.NET neu (nicht Visual Studio .NET), indem Sie Folgendes an der Visual Studio .NET-Befehlsaufforderung eingeben:

```
C:\>aspnet_regiis.exe -i
```

Anschließend starten Sie den IIS neu, indem Sie an der Eingabeaufforderung Folgendes eingeben:

```
C:\>iisreset
```

Hilfe aus dem Internet

Wenn alles nicht hilft, lesen Sie folgendes Dokument (englisch) sorgfältig:

<http://www.gotdotnet.com/team/csharp/Information/Whitepapers/howtosolvedebuggerproblems.doc>

Außerdem können Sie mit den Knowledge Base-Artikel Q306172 starten, um mögliche Debuggerprobleme zu eruieren.

2.5 Makros in Visual Studio .NET

Falls Sie Vorgänge, die häufig auszuführen sind, automatisieren möchten, können Sie Makros programmieren. Die Makrosprache ist Visual Basic .NET. Prinzipiell sind alle Teile von Visual Studio .NET programmierbar. Das fußt auf einem so genannten Automatisierungsmodell, der das Studio komplett als Instanz im Programm zur Verfügung stellt.

2.5.1 Erste Schritte mit Makros

Wenn Sie sich mit Makros beschäftigen möchten, suchen Sie zuerst den MAKRO-EXPLORER auf. Er lässt sich über EXTRAS | MAKROS | MAKRO-EXPLORER oder über die Tastenkombination **[Alt]-[F8]** aufrufen. Sie können nun beispielsweise den Zweig SAMPLES öffnen und sich die mitgelieferten Beispiele ansehen. Wählen Sie dazu im Kontextmenü eines Makros den Eintrag BEARBEITEN. Sie gelangen so in den Makro-Editor. Dies ist eine mit Visual Studio vergleichbare Entwicklungsumgebung für Makros. Sie können hier Ihre Hilfsprogramme entwickeln, debuggen und ausführen.

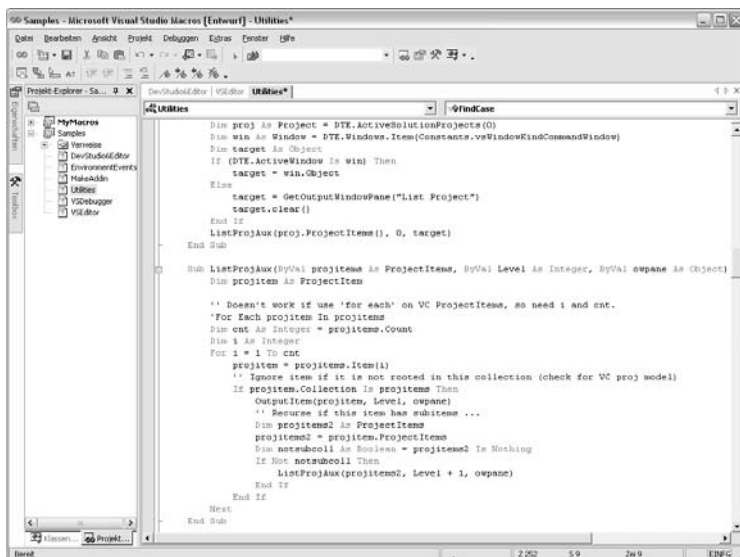


Abbildung 2.43: Der Makro-Editor in Visual Studio .NET

Manchmal genügt es jedoch, nicht gleich neue Makros zu programmieren, sondern lediglich ein paar Schritte aufzuzeichnen, um sich viel Tipparbeit zu sparen. Dazu verwenden Sie den Makrorecorder.

2.5.2 Der Makrorecorder

Wählen Sie zur Aufzeichnung aller folgenden Aktionen die Tastenkombination **[Alt]+[Umschalt]+[R]**. Wenn Sie fertig sind, wiederholen Sie die Tastenkombination. Nun erscheint das fertige

Makro im Makro-Explorer im Zweig MYMACROS unter RECORDINGMODULE. Sie können es von dort umbenennen oder weiter mit dem Makroeditor bearbeiten.



Abbildung 2.44: Ablage eines aufgezeichneten Makros

3 Einführung in Visual Basic .NET

Dieses Kapitel führt in die Sprache VB.NET ein. In diesem Buch ist VB.NET die Programmiersprache, in der ASP.NET-Seiten erstellt werden. Dieses Kapitel soll kein vollwertiges VB.NET-Buch ersetzen. Der vermittelte Stoff reicht aber völlig aus, sämtliche Beispiele des Buches zu verstehen. Zwei spezielle Abschnitte behandeln die Unterschiede zu VB 6 und zu VBScript.

3.1 Überblick für Umsteiger von VBScript

Wenn Sie von ASP 3.0 auf ASP.NET umsteigen, werden Sie schnell feststellen müssen, dass dies keine neue Version eines bekannten Programms ist, sondern etwas völlig Neues. Die Wahl von VB.NET als Programmiersprache (gegenüber C#) verschafft nur einen marginalen Vorteil. Die Syntax ist zwar entfernt ähnlich aber das Verhältnis zwischen VBScript und VB.NET entspricht etwa dem von JavaScript zu Java. So vergleichbar die Namen klingen, so sehr unterscheiden sich die Sprachen in Syntax und Konzept.

Damit der Einstieg dennoch etwas erleichtert wird, finden Sie in diesem Kapitel eine Übersicht der wesentlichsten Unterschiede. Lesen Sie anschließend unbedingt Abschnitt 3.4, »Spracheinführung« ab Seite 153. Erst in der vollständigen Sprachdarstellung finden Sie die wichtigen Neuigkeiten für den Umgang mit dem .NET-Framework. Der Überblick hier frischt nur Ihr VBScript-Wissen etwas auf.

3.1.1 Variablen, Geltungsbereiche und Definitionen

VB.NET sollte als eine typstrenge Sprache verstanden werden. Visual Studio .NET unterstützt dies durch die Vorgabe entsprechender Optionen. Sie müssen dann generell Variablen deklarieren.

Option Explicit

Dies erfolgt in der bekannten Weise mit dem Schlüsselwort `Dim` bzw. den später noch vorgestellten Modifikatoren. Es gibt allerdings nachwievordas Schlüsselwort `Option Explicit`. Damit kann auch festgelegt werden, dass die Deklaration implizit erfolgen soll. Meist ist das keine gute Idee. Deshalb sieht die Voreinstellung folgendermaßen aus:

```
Option Explicit On
```

Option Strict

Lassen Sie den Parameter allerdings weg, nimmt der Compiler `Off` an. Neben der Deklaration ist auch die Angabe des Datentyps erforderlich. Bedingt durch das dem .NET-Framework zugrunde liegende Common Type System, das allen Sprachen einheitliche Datentypen bereitstellt, kommt der exakten Typdeklaration eine große Bedeutung zu. Es gibt deshalb keinen allgemeinen Datentyp `Variant` mehr. In manchen Fällen wird dafür der Typ `Object` verwendet. Die Verarbeitung erfolgt aber intern anders, sodass dies nicht als Austausch eines Schlüsselwortes zu verstehen ist. VB.NET kann bei elementaren Datentypen erkennen, was gemeint ist, ohne dass das entsprechende Schlüsselwort explizit geschrieben wird. Möchten Sie das nicht, hilft `Option Strict On`. Etwas freizügiger programmiert es sich mit `Option Strict Off`.



In diesem Buch folgen alle Programme dem Prinzip der typstrengen Schreibweise, sind also so programmiert, dass beide Optionen, `Strict` und `Explicit` auf `On` stehen. Machen Sie es sich zur Gewohnheit, so zu arbeiten.

Mehr Ausführungen dazu finden Sie im Abschnitt 3.4.2, »Variablen und Datentypen« ab Seite 154. Wenn die Datentypen festgelegt werden, muss es Konvertierungsmöglichkeiten geben. Das Framework bietet dafür alle erdenklichen Methoden. VB.NET hilft aber bei der Konvertierung, indem nach Möglichkeit automatisch umgewandelt wird.

Datentypen

Die Definition der Datentypen erfolgt mit dem Schlüsselwort `As`:

```
Dim var As String
```

```
Dim i As Integer
```

Es ist möglich, mehrere Variablen auf einer Zeile zu deklarieren:

```
Dim i, j, k As Long
```

Ebenso kann sofort eine Wertzuweisung erfolgen, was bei den Beispielen im Buch recht oft erfolgt:

```
Dimm As Integer = 1
```

Das Framework liefert in erster Instanz fast immer Klassen. Die Verwendung ist erst möglich, wenn ein Objekt daraus instanziiert wird, was mit `New` erfolgt:

```
Dim classvar As Queue = New Queue()
```

Da durch die strenge Deklaration der Typen Umwandlungen nötig sind, kam eine neue Umwandlungsfunktion hinzu, `CType`. Die von VBScript bekannten Funktionen wie `CInt` oder `CBool` gibt es weiterhin. In .NET gibt es unzählige Typen, da jede Ableitung einer Klasse einen Datentype ergibt. `CType` erlaubt beliebige Umwandlungen, natürlich nur soweit, wie es von den jeweiligen Objekten unterstützt wird:

```
Dim i As Integer = 23  
Dim s As String = CType(i, String)
```

CType

In VBScript waren Variablen immer global sichtbar. Lediglich innerhalb von Prozeduren oder Funktionen definierte Variablen waren nur dort verwendbar. VB.NET besitzt ein ausgefeiltes Konzept der Geltungsbereiche. Da generell objektorientiert programmiert werden muss (siehe Abschnitt 3.4.8, »Namensräume und Klassen« ab Seite 176), sind feinere Abstufungen unbedingt erforderlich. Es gibt nun private Variablen (`Private`) und nach wie vor global sichtbare (`Public`). Daneben kann noch der Sichtbereich in der Vererbungskette festgelegt werden.

Geltungsbereiche

3.1.2 Allgemeine Hinweise zur Notation

Wenn Prozeduren¹ aufgerufen werden, müssen immer runde Klammern gesetzt werden.

Die Schlüsselwörter `Let` und `Set` gibt es nicht mehr. Zuweisungen werden ohne weiteres Schlüsselwort mit dem Gleichheitszeichen vorgenommen. Ein Hinweis auf ein Objekt, wie es in VBScript mit `Set` erfolgte, ist nicht mehr notwendig. Alle Variablen sind Objekte, ohne Ausnahme.

¹ Eigentlich handelt es sich immer um Methoden, die präzise Benennung wird später im Zusammenhang mit der objektorientierten Programmierung geklärt.

Geänderte Sprachanweisungen

- If** Einige Vereinfachungen in der Notation fielen der Überarbeitung zum Opfer. So ist die verkürzte Schreibweise von `If Then` nicht mehr erlaubt. Jedes `If` verlangt zwingend sein `End If`.

```
If i = 23 Then
    ' tu was
End If
```

Allerdings fällt das in der Praxis kaum auf, wenn Sie Visual Studio.NET als Editor verwenden. Tippen Sie nach `If i = 23` auf `[Enter]`, wird der komplette Block durch ein Editormakro erzeugt.

- IIf** Neu ist eine Kombinationsanweisung `IIf`, die den `If`-Befehl verkürzt, aber leider sehr langsam ist:

```
val = IIf(Ausdruck, "True-Wert", "False-Wert")
```

- While** Die `While`-Anweisung ist etwas logischer, statt `Wend` wird sie nun mit `End While` geschlossen und entspricht damit einem etwas verständlicheren Schema. Für die anderen Schleifen `For` (endet immer noch mit `Next` statt mit `End For`) und `Do` (endet mit `Loop` und nicht mit `End Do`) gilt das nicht.

Optimierungsstrategien

Da VB.NET auf einem Compiler basiert, können bei der Verarbeitung Optimierungen vorgenommen werden. Das passiert auch fleißig, ohne dass Sie es bemerken. Sie müssen daran aber in manchen Fällen denken, beispielsweise bei folgendem Konstrukt:

```
If Function1() = 12 or Function2() = 0 Then
```

- Neu: OrElse, AndAlso** Hier wird zuerst, wie erwartet, *Function1* aufgerufen. Ergibt der Ausdruck auf der linken Seite danach `True`, kann der gesamte Ausdruck aufgrund der Oder-Verknüpfung nur noch `True` sein. Der Compiler optimiert die rechte Seite dann einfach weg, *Function2* wird nicht ausgeführt. Das kann zu unglücklichen Seiteneffekten führen, wenn Sie in *Function2* irgendeine Aktion starten, die unabhängig vom Rückgabewert notwendig ist, beispielsweise eine Datenbankverbindung öffnen oder schließen. Zur Lösung des Problems gibt es neue Boolesche Operatoren mit den Namen `OrElse` und `AndAlso`. Mehr dazu finden Sie in Abschnitt »Bedingungen« ab Seite 165.

Die getrennte Auswertung der beiden Seiten einer Booleschen Verknüpfung hat weitere Vorteile beim Programmieren. So können Sie abhängige Aktionen zusammenfassen, ohne das es zu einem Fehler kommt.

```
If Not objekt Is Nothing And objekt.wert = 0 Then
```

Misslingt hier der linke Teil, bricht der Optimierer ab. Das verhindert, dass der rechte Teil in einem Laufzeitfehler läuft. So kann man die Optimierung ausnutzen, um kompakteren Code zu schreiben.

Operatoren

Neues gibt es auch bei originären Operatoren. So können Zeichenketten außer mit & auch mit + verknüpft werden. In allen Fällen sind Leerzeichen zwischen den Operanden und dem Operator erforderlich. Visual Studio .NET fügt diese beim Tippen automatisch ein.

3.1.3 Umgang mit den Funktionen und Bibliotheken

VBScript wurde durch mehrere externe Bibliotheken erweitert. Dazu gehören Hashes mit `Scripting.Dictionary` oder der Zugriff auf das Dateisystem mit `Scripting.FileSystemObject`. Solche Aufgaben übernimmt, umfangreicher und ausgefeilter als jemals zuvor, das Framework. Vergessen Sie wirklich alles über externe Bibliotheken.

Bibliotheken

VBScript kennt viele Funktionen, für mathematische Berechnungen (beispielsweise `Abs`), für Datum und Zeit (`Date`, `Now`). Einiges gibt es noch aus Kompatibilitätsgründen, aber im Grunde genommen sind sämtliche Funktionen durch das weit umfassendere Framework ersetzt worden. Die Spracheinführung zu VB.NET zeigt die Framework-Klassen und geht nicht mehr auf alte VB/VBA-Funktionen ein.

3.2 Überblick über Neuerungen gegenüber VB 6

Dieser Abschnitt zeigt die wesentlichen Neuerungen in VB.NET gegenüber VB 6. Umsteiger von VB 6 werden jedoch darüber hinaus weitere Besonderheiten von ASP.NET lernen müssen. Deshalb ist anzuraten, auch die Spracheinführung zumindest zu überfliegen.

3.2.1 Variablen und Datentypen

Deftype-Anweisungen

In Visual Basic 6 konnte der Standarddatentyp einer Variablen mit den *Deftype*-Anweisungen festgelegt werden. Diese Möglichkeit und die entsprechenden »Def«-Schlüsselwörter *DefBool*, *DefStr* usw. sind ersatzlos entfallen.

Currency

Auch der Datentyp *Currency* ist ersatzlos entfallen. Dafür wird mit *Decimal* gearbeitet. Die Ursache dafür liegt im Wesentlichen in der notwendigen Kompatibilität zum Common Type System (CTS), das die allen .NET-Programmen zugrunde liegenden Datentypen liefert.

Date

Neu ist auch die Art und Weise, wie Datumsangaben gespeichert werden. Während VB 6 für *Date* hier intern *Double* verwendet, wird in VB.NET der Typ *DateTime* aus dem CTS eingesetzt.

Empty

Variablen können leer sein. Da es sich – meist – um Objekte und damit um Referenztypen² handelt, wird statt *Empty* als Prüfung auf fehlende Definition nun *Nothing* verwendet:

```
If object Is Nothing Then
```

Variant

Der Datentyp *Variant* ist nicht mehr vorhanden. Stattdessen wird *Object* eingesetzt. Allerdings verhält sich *Object* in vielen Fällen anders und verfügt über neue Eigenschaften.

In VB 6 können einige Funktionen explizit zwischen Zeichenketten (*String*) und dem Typ *Variant* unterscheiden. Dies erfolgt durch ein angehängtes Dollarzeichen, beispielsweise *Chr\$*. In .NET gibt es in den Fällen, wo eine Funktion verschiedenen Datentypen zurückgeben kann, so genannte Überladungen der Methode. Das

² Darauf wird später noch detailliert eingegangen. Werttypen verfügen über eigene Prüfkonstanten, beispielsweise *String.Empty* für eine leere Zeichenkette. Referenztypen sind meist Klasseninstanzen, weshalb die Prüfung mit *Nothing* gelingt.

bedeutet, dass mit demselben Namen mehrere Versionen existieren, wobei der Compiler aus dem Kontext des Aufrufes heraus die passende Methode verwendet. Die alte Schreibweise wird jedoch weiterhin akzeptiert, da VB.NET dies als Konvertierung in den Typ `String` interpretiert. Sie sollten jedoch vermeiden, Code mit typischer VB 6-Syntax zu schreiben.

3.2.2 Funktionen

Viele Funktionen, die VB 6 standardmäßig bot, sind nun Teil des Frameworks. Deshalb ändert sich in vielen Fällen sowohl der Name als auch die Syntax. Die Klassen aus dem Framework sind in Namensräumen organisiert. Einige werden standardmäßig immer eingebunden, wenn Sie mit Visual Studio .NET arbeiten. Deshalb verhält sich VB.NET oft ähnlich. Tatsächlich ist die innere Funktionsweise anders.

Soweit VB.NET zu VB 6 kompatible Funktionen und Modi beherrscht, wurde dies im Buch nicht weiter beachtet. Stattdessen finden Sie konsequent die Anwendung des Frameworks und der darin enthaltenen Klassen.



Grundsätzlich sind alle Funktionen im objektorientierten Framework Methoden. »Methode« ist ein Begriff aus der objektorientierten Programmierung. In Abschnitt 3.4.8, »Namensräume und Klassen« ab Seite 176 wird dies näher erläutert.

Der Visual Basic-Kompatibilitätsmodus

Speziell für die Bedürfnisse der VB 6-Programmierer enthält das .NET-Framework das Modul `Microsoft.VisualBasic`. Dort sind in verschiedenen Klassen statische Methoden definiert, die direkt VB 6-Funktionen ersetzen. Diese können also, wie bisher auch, »unmittelbar« eingesetzt werden. Ein Beispiel sind Funktionen für Zufallszahlen. Im Framework werden diese normalerweise über die Klassen `Random` erzeugt. In VB 6 erledigte dies die Funktion `Rnd`. Das funktioniert auch in jedem ASP.NET-Programm, weil der Namensraum `Microsoft.VisualBasic` standardmäßig eingebunden wird. Eine Gegenüberstellung der wichtigsten Funktionen wird in den folgenden Abschnitten gezeigt, wobei sich die Auflistungen auf den Teil beschränken, der für ASP.NET von Interesse sein könnte.

Mathematische Funktionen

Die mathematischen Funktionen sind im Namensraum `System` enthalten. Die folgende Tabelle zeigt die alten und neuen Namen:

VB 6-Funktion	Methode aus dem Framework
Abs	Math.Abs
Atn	Math.Atan
Cos	Math.Cos
Exp	Math.Exp
Log	Math.Log
Round	Math.Round
Sgn	Math.Sign
Sin	Math.Sin
Sqr	Math.Sqrt
Tan	Math.Tan

Tabelle 3.1: Auswahl mathematischer Funktionen in VB 6

Wenn der Zugriff auf Namensraum und Klasse nicht explizit mit `Imports System.Math` erfolgt, schreiben Sie `System.Math.<Method>`.

Die Laufzeitbibliothek liefert außerdem die Funktionen `Rnd` und `Randomize`. Vermeiden Sie die Nutzung bei neuen Programmen und verwenden Sie stattdessen die Klasse `Random`.

Datum und Uhrzeit

VB 6 war auch wegen der Datums- und Zeitfunktionen beliebt. Das .NET-Framework legt hier noch zu und bietet ein komplexes Datums- und Kalendersystem.

VB 6-Funktion	Methode oder Eigenschaft aus dem Framework	Herkunft
Date	Now, Today (Eigenschaften)	DateTime-Struktur
CVDate	DateValue	Laufzeitbibliothek
Date\$	DateString	Laufzeitbibliothek
Calendar	CurrentCulture (Eigenschaft)	System.Globalization
Time	TimeOfDay	DateTime-Struktur

Tabelle 3.2: Auswahl wichtiger Datums- und Zeitfunktionen aus VB 6

Zeichenketten

Auch die Zeichenketten finden sich komplett im Framework wieder, umfangreicher und leistungsfähiger als in VB 6. Das Laufzeitmodul sichert jedoch auch hier eine weitgehende Kompatibilität, sodass VB 6-Code leicht verwendet werden kann.

VB 6-Funktion	VB.NET-Methode
LTrim	String.TrimStart
Trim	String.Trim
RTrim	String.TrimEnd
Left	String.Substring
Right	String.Substring
Len	String.Length
Mid	String.Substring
Join	String.Join
Split	String.Split

Tabelle 3.3: Auswahl wichtiger Zeichenkettenfunktionen aus VB 6

Beachten Sie, dass die Klasse `String` weit mehr Methoden und Eigenschaften liefert.

3.3 Prinzip der ASP.NET-Abarbeitung

Auf die Prinzipien der Arbeitsweise von Webseiten wurde bereits in Abschnitt 1.4, »Grundprinzip der Programmierung mit ASP.NET« ab Seite 59 hingewiesen. Viele Programme dieses Kapitels haben folgenden grundsätzlichen Aufbau:

Eingebetteter Code

```
<%@ Page Language="vb" %>
<html>
  <head><title>VB.NET lernen</title></head>
  <body>
    <h1>Eingebetteter Code:</h1>
    <%
      Dim hw As String = "Hello World"
      Response.Write(hw)
    %>
  </body>
</html>
```

Listing 3.1: Standard-Programm mit eingebettetem VB.NET-Code (HelloWorld.aspx)

Die erste Zeile enthält eine Seitendirektive. Darauf wird in Abschnitt 10.2, »Konfiguration einzelner Seiten: Die Direktiven« ab Seite 954 detailliert eingegangen. An dieser Stelle schreiben Sie die Zeile so ab, wie sie ist. Sie dient der Festlegung von VB.NET als Standardsprache. Sie sollten diese Angabe immer machen, auch wenn VB.NET die Standardsprache ist und die Angabe deshalb entfallen könnte.

Dann folgt die HTML-Seite, die Ihr Browser anzeigen soll. Der Code selbst ist in ASP-Tags `<% %>` eingebettet. Dieses Prinzip hat sich gegenüber ASP nicht geändert – es sind nur weitere, alternative Wege hinzugekommen, unter anderem die so genannte Datenbindung. Dies wird ausführlich in Abschnitt 6.6.2, »Aufbau der Vorlagen in Daten-Steuerelementen« ab Seite 570 diskutiert werden.

Dateierweiterung

Vergessen Sie nun nicht, das Programm mit der Dateierweiterung `.aspx` zu speichern. Führen Sie es im Browser aus, wobei auf einem Entwicklungssystem der Servername »localhost« verwendet wird (`»http://localhost/pfad/dateiname.aspx«`).



Abbildung 3.1: Ausführung des Programms HelloWorld.aspx

Wenn das gezeigte Bild auch bei Ihnen zu sehen ist, können Sie alle nachfolgenden Beispiele ausführen und bequem VB.NET und ASP.NET lernen.



In diesem Buch wird propagiert, dass generell hinterlegter Code verwendet werden soll. Der Vorschlag, am Anfang eingebetteten Code zu verwenden, widerspricht dieser Idee. Allerdings ist es für die ersten Schritte in einer neuen Sprache wichtig, die Programmierumgebung soweit wie möglich zu vereinfachen, um sich auf das Wesentliche konzentrieren zu können. Deshalb wird hier – ausnahmsweise – eingebetteter Code verwendet. Später werden Sie dies nur noch in Ausnahmefällen verwenden.

3.4 Spracheinführung

Dieser Abschnitt enthält eine kompakte Spracheinführung, die wenig Vorwissen voraussetzt und auf den Bedarf dieses Buches »zurechtgestutzt« wurde. Aus Platzgründen kann VB.NET nicht umfassend behandelt werden. Konsultieren Sie bei Bedarf die Online-Hilfe und die MSDN-Bibliothek, die umfassende und vollständige Informationen bietet. Eine Unterscheidung zwischen »Neu« (VB.NET) und »Alt« (VB 6) wird in diesem Abschnitt nicht mehr vorgenommen.

3.4.1 Schreibweise und Notation

In VB.NET werden Anweisungen jeweils auf eine Zeile geschrieben. Ein explizites Befehlsendezeichen gibt es nicht. Wenn es unbedingt erforderlich ist, können mehrere Anweisungen mit einem Doppelpunkt getrennt werden:

```
Dim i As Integer : Dim s As String
```

Bei der Arbeit mit dem .NET-Framework ergeben sich manchmal sehr lange Zeilen, die dann schwer lesbar sind. In solchen Fällen können Sie am Zeilenende einen Unterstrich setzen und auf der folgenden Zeile fortfahren. Dies ist nur ein Hinweis für den Compiler, die Zeilen zusammen zu ziehen. Sehr häufig wird diese Technik im vorliegenden Buch verwendet, um die langen Methodendeklarationen lesbarer zu gestalten:

```
Public Sub Page_Load(sender As Object, e As EventArgs) _  
    Handles MyBase.Load
```

Das Trennzeichen für Methoden und Eigenschaften beim Aufruf aus einem Objektkontext heraus ist der Punkt:

```
Objekt.Methode()
```

Zwingend erforderlich sind in allen Fällen die runden Klammern zur Aufnahme der Parameter, auch dann, wenn keine Parameter vorhanden sind und die Klammern leer bleiben. Ausführlicher wird dies in den Abschnitten zu Klassen und Methoden betrachtet, beispielsweise im Abschnitt 3.4.8, »Namensräume und Klassen« ab Seite 176.

3.4.2 Variablen und Datentypen

Variablen werden in VB.NET mit dem Schlüsselwort *Dim* deklariert. Dabei wird der Sichtbereich der Variablen verwendet, den VB.NET im aktuellen Kontext als Standard annimmt. Dies bedeutet, dass die spätere Verwendbarkeit auf einen bestimmten Bereich des Programms eingeschränkt sein kann.

Variablendeklarationen

Beispiele für typische Variablendeklarationen finden Sie nachfolgend:

```
Dim var As String
Dim zahl As Long
Dim z1, z2, z3 As Integer
```

Hier wird deutlich, dass es nicht ausreicht die Variable als solche, also deren Namen, festzulegen. Sie müssen immer einen Datentyp angeben. Der nächste Abschnitt geht darauf detailliert ein.

Mit der Deklaration kann auch gleich eine Wertzuweisung verbunden werden:

```
Dim start As Integer = 1
```

Einführung in das Common Type System

CTS-Datentypen

Intern verwendet VB.NET die Datentypen, die vom Framework geliefert werden. Die folgende Tabelle zeigt, welche Basistypen es gibt und wie sich diese in VB.NET und im Framework gegenüberstehen. Wenn VB.NET einen Typ durch ein eigenes Schlüsselwort abbildet, wird dies als »nativer Typ« bezeichnet. Als nicht native Datentypen werden hier solche beschrieben, die in VB.NET nicht direkt, sondern ausschließlich über das Framework, zur Verfügung stehen.

Typ	Beschreibung	Wertebereich	VB.NET-Notation
System.Boolean	Logischer Wert	True oder False	Boolean
System.Byte	Ein Byte	0 bis 255	Byte
System.Char	Ein Zeichen	Ein Unicode-Zeichen	Char
System.String	Unicode-Zeichenkette	Zeichenfolge bis 2 GByte	String
System.DateTime	Zeit- und Datums-werte	64-Bit-Zahl, die den Bereich vom 1.1.0001 (Jahr 1) bis 31.12.9999 repräsentiert	Nicht nativ
System.Decimal	Zahlen mit 28 Dezimalstellen	Dezimalzahlen sind auf die Stellenzahl genau und können beispielsweise für Währungs-werte eingesetzt werden.	Decimal
System.Double	64-Bit-Gleitkommazahl	$-1,79^{308}$ bis $+1,70^{308}$	Double
System.Int16	16-Bit-Ganzzahl	-32.768 bis +32.768	Short
System.Int32	32-Bit-Ganzzahl	-2.147.483.648 bis +2.147.483.648	Integer
System.Int64	64-Bit-Ganzzahl	-9.223.372.036.854.775.808 bis +9.223.372.036.854.775.808	Long
System.Single	4-Byte-Gleitkommazahl	$-3,4 \times 10^{38}$ bis $+3,4 \times 10^{38}$	Float
System.TimeSpan	Positive oder negative Zeitspanne	-10675199.02:48:05.4775808 bis +10675199.02:48:05.47758 (Tage.Stunden:Minuten: Sekunden.Millisekunden)	Nicht nativ
System.Array	Datenfeld	Feld, das beliebige andere Typen enthalten kann, auch weitere vom Typ Array	Notation mit typ()-Syntax
System.String	Unicode-Zeichenkette	Zeichenketten bis 2 Mrd. Zeichen (2 GByte)	String
System.Object	Basistyp	Universeller Speicher, anfangs 4 Byte groß	Nicht nativ

Tabelle 3.4: Datentypen im Framework im Vergleich zu VB.NET

An der Beschreibung des letzten Typs, `System.Object`, wird klar, dass allen Typen Objekte zugrunde liegen. Auf Objekte wurde

Bedeutung des Typs Object

noch nicht weiter eingegangen, dies soll jedoch nachgeholt werden. Denn die Welt von .NET ist eine Welt der Objekte. Sie müssen – ausnahmslos – objektorientiert programmieren. Es liegt in der Natur von Objekten, sowohl über Eigenschaften als auch Methoden zu verfügen. Die folgende Information sollten Sie, wenn Sie mit der objektorientierten Programmierung noch nicht vertraut sind, einfach als gegeben hinnehmen. `System.Object` verfügt, als Basisklasse aller Datentypen, über folgende Methoden:

Methoden	Beschreibung
<code>Equals</code>	Vergleicht zwei Objekte auf Gleichheit
<code>GetHashCode</code>	Gibt einen Identifikator des Objekts zurück. Hashes sind spezielle Arten von Aufzählungen. Werden Variablen als Schlüssel verwendet, können die Elemente anhand der Hashcodes identifiziert werden.
<code>GetType</code>	Gibt ein Typ-Objekt zurück, mit dem der Typ selbst untersucht werden kann. Diese Art des Zugriffs auf interne Definitionen wird Reflektion genannt.
<code>ToString</code>	Mit dieser Methode wird der Typ in eine Zeichenkette umgewandelt. Sie haben dies bereits im ersten Beispiel verwendet.

Tabelle 3.5: Methoden aus `System.Object`

Wenn Sie Variablen deklarieren, müssen Sie nur dann das Framework direkt verwenden, wenn der Typ in VB.NET nicht direkt verfügbar ist. Die Deklaration aus dem ersten Beispiel könnte auch folgendermaßen aussehen:

```
Dim hw As System.String = "Hello World"
```

Dies erzeugt exakt denselben Code – intern greift VB.NET nämlich immer auf die Datentypen des Frameworks zurück. Die Nutzung der VB-Notation spart lediglich Schreibarbeit.

Typdeklarationen mit Literalzeichen und Typsymbolen

Visual Basic kennt neben der direkten Deklaration von Typen auch eine spezielle Syntax zum Kennzeichnen von Konstanten oder Literalen. Dies ist immer dann angebracht, wenn die Zuordnung nicht eindeutig gelingt. Das folgende Beispiel zeigt, wie ein einzelnes Zeichen vom Typ `Char` zugewiesen wird – durch Anhängen des Literals »c«:

```
Dim cX As Char = "x" c
```


Das Literalzeichen `c` wird dem Literal »x« nachgestellt – ohne Leerzeichen – um die Zuordnung auch dann zu ermöglichen, wenn die strenge Typprüfung mit `Option Strict On` eingeschaltet wurde. In solchen Fällen wäre der Compiler nicht in der Lage, die Zeichenkette (String) "x" von dem Zeichen (Char) "x" zu unterscheiden. Neben den Literalzeichen, die nur Literalen folgen dürfen, gibt es noch so genannte Typsymbole. Diese dürfen auch Variablen folgen und können so eine Konvertierung erzwingen. Es ist jedoch besser, einen Typ aus dem Framework zu verwenden und die Konvertierung gegebenenfalls mit `CType` vorzunehmen.

Literalzeichen

Die folgende Tabelle zeigt alle Typsymbole und Literalzeichen³ auf einen Blick:

Zeichen	Datentyp	Einsatzfall
%	Integer	Dim i%
&	Long	Private lng&
@	Decimal	w@ = 9.99
!	Single	Dim s!
#	Double	Shared z#
\$	String	C\$ = "abc"
S	Short	256S
I	Integer	11I
L	Long	2045L
D	Decimal	19.99D
F	Single	0.55f
R	Double	1.99R
C	Char	"x"c

Tabelle 3.6: Literalzeichen und Typsymbole

Die Typsymbole können auch mit Literalen verwendet werden. Umgekehrt können jedoch die Literalzeichen nur mit Literalen, nicht mit Variablen zum Einsatz kommen. Während Zahlen jeder Art immer allein stehen, können Zeichen, Zeichenketten und Datumsangaben durch spezielle Zeichen umschlossen werden, um auf den Typ des Literals hinzuweisen. Bei Zeichen und Zeichenketten sind dies die bekannten doppelten Anführungszeichen. Datumsangaben werden von #-Zeichen eingeschlossen: #23.05.1964#.

3 Als Literalzeichen sind auch Kleinbuchstaben erlaubt.

Allerdings machen Zahlen oft Probleme, wenn der Compiler bei der Analyse auf einen anderen Typ schließt als der Programmierer bei der Entwicklung »gedacht« hat. Folgendes Beispiel führt zu einem Fehler:

```
Dim zahl As Decimal
zahl = 10000000000000000000
```

Hier wird auf `Long` erkannt, weil die Zahl keine Kommastelle enthält. Schreibt man ».`0`« am Ende, wird dagegen auf `Double` erkannt, was wiederum falsch ist. Letztlich hilft nur das Literalzeichen »`D`« als Suffix des Zahliterals.



Eine Besonderheit stellen Hexadezimalzahlen dar. Diese lassen sich in der gewohnte Schreibweise durch den Präfix `!` & `H` in der Literalform darstellen: `&H00FF00` (Farbe Grün im RGB-Farbraum). Intern werden die Zahlen zu `Integer` oder `Long` und unterliegen denselben Bedingungen in der Darstellung wie Dezimalzahlen auch. Ähnlich funktioniert das auch bei Oktalzahlen, hier ist das Präfix `&O` (Zahl »`0`«).

Typumwandlungen

Im Zusammenhang mit Typen und Typumwandlungen soll auch der Hinweis auf die originären VB-Konvertierungsfunktionen nicht fehlen. Mit `CInt`, `CBool` usw. können Sie native Datentypen direkt wandeln. Die folgende Tabelle zeigt alle Typumwandlungsfunktionen und deren Rückgabetypen.

Umwandlungsfunktion	Zieldatentyp
<code>CBool</code>	Boolesch
<code>CByte</code>	Byte
<code>CChar</code>	Char
<code>Cdbl</code>	Double
<code>CDec</code>	Decimal
<code>CInt</code>	Integer
<code>CLng</code>	Long
<code>CObj</code>	Object
<code>CShort</code>	Short
<code>CSng</code>	Single
<code>CStr</code>	String
<code>CDate</code>	Date

Tabelle 3.7: Native Typumwandlungsfunktionen

Universeller ist die Typumwandlungsfunktion `CType`, die jede Konvertierung vornehmen kann, solange die damit verbundenen Objekte dies unterstützen:

```
var = CType("x", Char)
```

Der zweite Parameter ist der Bezeichner des Typs bzw. einer Klasse des Frameworks, gegebenenfalls vollständig qualifiziert durch Angabe des Namensraumes:

```
var = CType(controlObject, System.Web.UI.WebControls.Button)
```

Neben den direkten Typkonvertierungen gibt es auch Funktionen, die Darstellungskonvertierungen vornehmen. Neu im Framework sind die statischen `Format`-Methoden, die einige Klassen liefern. Darauf wird an mehreren Stellen im Buch noch eingegangen. VB.NET liefert jedoch standardmäßig alle alten Basic-Funktionen im Kompatibilitätsmodus mit. Die folgende Tabelle zeigt die wichtigsten:

Funktion	Bedeutung
Hex	Dezimalzahl als Hexadezimale Form ohne Präfix, Typ <code>String</code>
Oct	Dezimalzahl als oktale Form ohne Präfix, Typ <code>String</code>
Str	Zahl als Zeichenkette
Val	Zeichenkette als Zahl. Leerzeichen werden entfernt, Präfixe beachtet.
Chr, ChrW	Ermittelt den Zeichencode zu einem Zeichen, »W« steht für Unicode.
Asc, AscW	Ermittelt das Zeichen zu einem Zeichencode, »W« steht für Unicode.

Tabelle 3.8: Funktionen zur Darstellungsumwandlung von Typen

Beachten Sie, dass bei den Umwandlungen von Zeichen in Zeichencodes die aktuelle Zeichentabelle verwendet wird.

Das Typkonzept genauer betrachtet

Die bisherige Einführung umfasste nur die einfachen Datentypen. Das Typkonzept von .NET ist weit umfassender. Unterteilen kann man es global in folgende Typenklassen:

- ▶ Werttypen
- ▶ Referenztypen

Werttypen	Die Werttypen werden verwendet, um einer Variablen einen Wert von definiertem Typ zuzuweisen. Das wurde bereits unter dem Begriff Datentypen beschrieben. Diese Werttypen umfassen außer den einfachen Datentypen auch noch Strukturen und Aufzählungen. Beides finden Sie im Abschnitt 4.3.4, »Zugriff auf Aufzählungen« ab Seite 231.
Referenztypen	Die Referenztypen umfassen Objekte, Klassen, Interfaces, Delegates, Zeichenketten und Arrays. Der Unterschied zu den Werttypen besteht weniger in der Verwendung, als mehr in der Art der Speicherung. Vereinfacht betrachtet, werden Variablen im Speicher an definierter Stelle abgelegt und nehmen dort einen definierten Platz in Anspruch. Bei Variablen, die sehr viele Daten enthalten, wäre es sehr uneffektiv, diese in einem linearen Speicher abzulegen. Da ständig Verschiebungen ausgeführt werden, wenn sich der Inhalt vergrößert oder Lücken entstehen, wenn er sich verkleinert, würde das System langsam werden. Deshalb wurden die Referenztypen eingeführt, die im Speicher nur als Verweis konstanter Länge existieren und auf einen speziellen Speicherbereich, den Heap (Haufen) zeigen.

Benutzerdefinierte Datentypen

Manchmal kann es sinnvoll sein, eigene Datentypen zu deklarieren. Dazu dient die `Structure`-Anweisung. Da Datentypen in .NET immer Objekte sind und diese aus Klassen stammen, verfügen sie auch über Eigenschaften und Methoden. Die Nutzung der Strukturen setzt deshalb Kenntnisse der objektorientierten Programmierung voraus. Abschnitt 3.4.8, »Namensräume und Klassen« ab Seite 176 geht auf dieses Thema ausführlich ein.

Logische Werte

Logische Werte	Logische Werte werden auch als Boolesch bezeichnet. Diese haben zwei mögliche Zustände: Wahr und Falsch; in VB.NET durch die Schlüsselwörter <i>True</i> und <i>False</i> gekennzeichnet.

Woher kommt der Ausdruck Boolesch?

George Boole (1815 – 1864) wurde 1815 im englischen Lincolnshire geboren. In einigen Biografien wird er als Ire bezeichnet, was genau genommen nicht korrekt ist. Allerdings verbrachte er den größten Teil seines Lebens in Irland. Er war ein englischer Mathematiker und Logiker, der einige grundlegende Regeln der Logik formulierte.

3.4.3 Konstanten

Konstanten verhalten sich wie Variablen, deren Inhalt sich während der Laufzeit nicht ändern kann. Definiert werden sie mit dem Schlüsselwort `Const`:

```
Const breite As Integer = 80
```

Konstanten erfordern, wie jede Variable, die Angabe eines Datentyps.

Vordefinierte Konstanten

Bei der Ausgabe von Text in Dateien oder im Browser sind manchmal spezielle Codes notwendig, beispielsweise für Zeilenumbrüche. VB.NET bietet hier eine Auswahl vordefinierter Werte:

Konstante	ASCII-Code und Bedeutung
vbCrLf	13+10, Wagenrücklauf und Zeilenvorschub
vbCr	13, Wagenrücklauf
vbLf	10, Zeilenvorschub
vbNewLine	13+10, Wagenrücklauf und Zeilenvorschub
vbNullChar	0
vbNullString	Zeichenkette der Länge 0, nur extern verwendet
vbTab	9, Tabulator
vbBack	8, Rückschritt
vbFormFeed	12, Seitenvorschub (im Drucker)
vbVerticalTab	11, Vertikaler Tabulator (nur Drucker)

Tabelle 3.9: Typische Konstanten zum Aufbau von Zeichenketten

3.4.4 Kommentare

Kommentare in VB.NET beginnen immer mit einem Kommentarzeichen oder dem Schlüsselwort `Rem`. Sie enden am Zeilenende:

```
Dim i As Integer ' Zählvariable
```

```
Rem (c) 2003 by Addison Wesley
```

Kommentare in VS.NET

Um mehrere Zeilen zu kommentieren, gibt es keine explizite Unterstützung durch die Sprache. Wenn Sie Visual Studio .NET einsetzen, hilft der Editor jedoch dabei. Markieren Sie dazu den Bereich, der auskommentiert werden soll. Wählen Sie dann **BEARBEITEN | ERWEITERT | AUSWAHL KOMMENTIEREN** oder die Tastenkombination `[Strg]-[K]` und `[Strg]-[C]`. Soll der Kommentar wieder entfernt werden, hilft der Befehl **BEARBEITEN | ERWEITERT | AUSWAHLKOMMENTIERUNG ENTFERNEN** oder die Tastenkombination `[Strg]-[K]` und `[Strg]-[U]`.

3.4.5 Operatoren

In Programmen wird viel gerechnet. VB.NET kennt die üblichen Operatoren und bietet wenig überraschendes bei der Schreibweise. Zwei haben Sie bereits kennengelernt: Die Zuweisung mit `=` und die Verkettung von Zeichenketten mit `>>+<<`. Daneben ist auch der Verkettungsoperator `>&<` einsetzbar, der diese Aufgabe in VBScript und VB 6 exklusiv übernahm.

Arithmetische Operatoren

VB.NET kennt die elementaren arithmetischen Operatoren:

```
x + y    ' Addition
x - y    ' Subtraktion
x * y    ' Multiplikation
x / y    ' Division mit dem Ergebnistyp Double
x \ y    ' Division mit dem Ergebnistyp Integer
x Mod y  ' Modulus (Rest der Ganzzahldivision)
x ^ y    ' Potenz xy
```

Bei der Division wird immer dann eine Gleitkommadivision durchgeführt, wenn einer der beiden Werte vom Typ *Double* oder *Single* ist. Für eine Ganzzahldivision müssen beide Operanden *Integer* (oder einer der Ganzzahltypen) sein.

In der Praxis können Sie die Operatoren für interessante Effekte einsetzen. So eignet sich der Modulus-Operator `Mod`, um bei HTML-Tabellen die Zeilen abwechselnd farbig hinterlegt dar-

zustellen. Das folgende Beispiel greift bereits einer Steueranweisung vor, der *For*-Schleife. Probieren Sie es dennoch aus, die Erklärung dazu folgt weiter hinten in diesem Kapitel.

Beachten Sie außerdem, dass die Abarbeitung innerhalb der Seite zwar möglich, aber konzeptionell nicht gewollt ist. Für dieses Beispiel wurde diese Variante gewählt, um den Code so weit wie möglich zu vereinfachen.



```
<%
Dim rd As String = "<td width="100">A</td>  ⚡
                  <td width="100">B</td>  ⚡
                  <td width="100">C</td>"
Response.Write("<table border="0" cellpadding="4">")
Response.Write("<th>Spalte A</th><th>Spalte B</th><th>Spalte C</th>")
Dim i As Integer
For i = 0 To 10
    If i Mod 2 = 0 Then
        Response.Write("<tr bgcolor="Gray">" & rd & "</tr>")
    Else
        Response.Write("<tr bgcolor="White">" & rd & "</tr>")
    End If
Next
%>
```

Listing 3.2: Verwendung des Modulus-Operators (*VbOperatorModulus.aspx*)

Das Erkennen jeder zweiten Zeile erfolgt durch die Berechnung $i \text{ Mod } 2 = 0$. Wenn das Ergebnis der Berechnung 0 ist (kein Divisionsrest), wird die Zeile grau dargestellt.

Damit sind Sie eigentlich schon mitten in der praktischen ASP.NET-Programmierung. In Abschnitt 4.2, »Steueranweisungen« ab Seite 158 wird gezeigt, wie *If* und *For...Next* und viele andere Anweisungen in VB.NET funktionieren.

Der einfachste Operator ist der Zuweisungsoperator, der beispielsweise für die Übertragung von Werten in eine Variable Verwendung findet. Sie können die grundlegenden arithmetischen Operatoren mit diesem Operator verbinden:

Zuweisungsoperator

`zahl += 45` ' addiert 45 zu der Variablen zahl

`zahl *= andere_zahl` ' multipliziert zahl mit andere_zahl

Weitere Zuweisungsoperatoren sind : `+=`, `-=`, `/=`, `%=` und `*=`.

Operatoren		
Spalte A	Spalte B	Spalte C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C
A	B	C

Abbildung 3.2: Ausgabe einer HTML-Tabelle mit reihenweise wechselnden Farben

Bitoperatoren

Wenn Variablen Werte enthalten, die sich in Byte- oder Bitform darstellen lassen, können Manipulationen mit Bitoperatoren sinnvoll sein. Als Datentyp kommt beispielsweise *byte* in Betracht. Der Operator *And* führt eine binäre UND-Verknüpfung durch, *Or* steht für eine ODER-Verknüpfung, während *Not* den Bitwert negiert. Das exklusive Oder (*Xor*) ist immer dann 1, wenn einer der beiden Operatoren 1 ist (also exklusiv), nicht aber beide. Die Operatoren entsprechen der Booleschen Algebra. Das Verhalten kann der Auflistung in der folgenden Tabelle entnommen werden.

x	y	x And y	x Or y	x Xor y	Not x	Not y
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	1	0

Tabelle 3.10: Verhalten der Bitoperatoren in VB.NET

Das folgende Beispiel zeigt das Prinzip. Denken Sie dabei an die interne Darstellung der Zahl 3 (00000011) bzw. 5 (00000101). Der gewählte Datentyp *Byte* arbeitet ohne Vorzeichen.

```
<%  
Dim bNull As Byte = 3  
Dim bEins As Byte = 5  
Response.Write ((bEins and bNull).ToString())
```



```
Response.Write ("<br>")
Response.Write ((bEins or bNull).ToString())
Response.Write ("<br>")
Response.Write ((Not bEins).ToString())
Response.Write ("<br>")
Response.Write ((Not bNull).ToString())
%>
```

Listing 3.3: Testprogramm für die Bitoperatoren (VbOperator8Bit.aspx)

Möglicherweise erscheint Ihnen die Verwendung von `ToString` hier etwas eigenartig. Immerhin wird die Methode an die Klammer gesetzt. Aber in VB.NET sind alle Elemente immer Objekte, also auch die Zwischenergebnisse einer Berechnung. Das Objekt `bNull`, verknüpft mit dem Objekt `bEins`, ergibt ein Objekt `(bEins And bNull)`, das natürlich in eine Zeichenkette verwandelt werden kann.



Abbildung 3.3: Wirkungsweise der Bitoperatoren

Neben den beschriebenen gibt es noch logische Operatoren, mit denen logische Ausdrücke gebildet werden können, also solche, die *True* oder *False* ergeben. Diese werden im Abschnitt »Bedingungen« ab Seite 165 erklärt.

**Logische
Operatoren**

3.4.6 Verzweigungen

Verzweigungen sind ein elementarer Bestandteil auch kleiner Programme. Dabei wird der ein oder andere Programmteil in Abhängigkeit von einer Bedingung ausgeführt.

Bedingungen

Bedingungen sind das bestimmende Element zur Steuerung von Verzweigungen. Es gibt praktisch kaum ein Programm, das völlig ohne die Steuerung mit Bedingungen auskommt. Das Programm

fällt mit Hilfe der Bedingung eine einfache Entscheidung: Ja oder Nein. Entsprechend müssen Ausdrücke, die in den Bedingungsanweisungen eingesetzt werden, logische Ausdrücke sein. Das Ergebnis muss für VB.NET als Wahr (*True*) oder Falsch (*False*) interpretierbar sein.

Ausdrücke Um Ausdrücke zu konstruieren, werden Operatoren benötigt. Man kann dabei logische Operatoren und Vergleichsoperatoren unterscheiden. Den eigentlichen Programmablauf steuern dann Anweisungen wie *If*, die nachfolgend vorgestellt werden. Auch die Anweisungen zur Schleifensteuerung nutzen logische Ausdrücke.

**Logische Operatoren
And, Or, Not** In vielen Abfragen wird eine logische Entscheidung verlangt. Mit speziellen Operatoren können Sie Ausdrücke verknüpfen. Das Ergebnis eines korrekten logischen Ausdrucks ist immer *False* oder *True*. Die folgenden Ausdrücke zeigen die Anwendung:

```
x And y      ' UND
x Or y       ' ODER
Not x        ' negiert den Wert (true == false, false == true)
```

**Optimierende Operatoren
AndAlso, OrElse** VB.NET arbeitet bei der Abarbeitung mit einer einfachen internen Optimierung. Wenn bei einem UND-Vergleich der erste Operator *False* ist, kann der Ausdruck nur *False* werden, unabhängig vom Zustand des zweiten. Entsprechend wird der Ausdruck einer ODER-Verknüpfung *True*, wenn der erste Operator *True* ist. Auch hier ist es nicht zwingend notwendig, den zweiten anzugeben. Dieses Verhalten kann in der Praxis unerwünscht sein. Wenn der zweite Operand ein Methodenaufruf ist, wird die Methode möglicherweise niemals aufgerufen. Dies kann aber notwendig sein, wenn die Methode neben dem Rückgabewert auch andere Aktionen ausführt. Deshalb gibt es zwei weitere Operatoren, die die Optimierung unterdrücken und den rechten Teil eines Ausdrucks immer auswerten:

```
x AndAlso y  ' UND mit Auswertung der rechten Seite
x OrElse y   ' ODER mit Auswertung der rechten Seite
```

Vergleichsoperatoren Um komplexe logische Ausdrücke erstellen zu können, benötigt man auch Vergleichsoperatoren. Die folgende Tabelle gibt einen Überblick:

Operator	Bedeutung
=	Gleichheit
<>	Ungleichheit
<	Kleiner als
>=	Größer als oder gleich
<=	Kleiner als oder gleich
>	Größer als

Tabelle 3.11: Die Vergleichsoperatoren auf einen Blick

Bedingungen mit *If...End If* auswerten

If (dt. falls oder wenn) testet eine Bedingung und führt den dahinter stehenden Block aus, wenn die Bedingung *True* ist, ansonsten wird der Block übergangen:

```
If Bedingung Then Anweisung
```

Wenn mehrere Anweisungen von der Bedingung abhängig sind, muss ein Block gebildet werden:

```
If Bedingung Then  
    Anweisung-1  
    Anweisung-2  
End If
```

Die *If*-Anweisung besteht aus dem Schlüsselwort und dem in runden Klammern stehenden logischen Ausdruck. Wenn der Ausdruck *True* ist, wird die nachfolgende Anweisung oder der Block (in geschweiften Klammern) ausgeführt. Ist der Ausdruck *False*, wird die Programmausführung mit der nächsten Anweisung oder dem nachfolgenden Block fortgesetzt. Das folgende Beispiel erzeugt eine Ausgabe, wenn die Variable *tag* einen bestimmten Wert enthält:

```
<%Dim tag As String = "Thursday"  
If tag = "Monday" Then Response.Write("Heute ist Montag")  
If tag = "Tuesday" Then Response.Write("Heute ist Dienstag")  
If tag = "Wednesday" Then Response.Write("Heute ist Mittwoch")  
If tag = "Thursday" Then Response.Write("Heute ist Donnerstag")  
If tag = "Friday" Then Response.Write("Heute ist Freitag")  
If tag = "Saturday" Then Response.Write("Heute ist Samstag")  
If tag = "Sunday" Then Response.Write("Heute ist Sonntag")  
%>
```

Listing 3.4: Einfache Verwendung von *If...Then* (VbKeywordIf.aspx)

Die Ausgabe ist wenig überraschend:



Abbildung 3.4: Ausgabe des Programms

Das Beispiel ist sicher primitiv. Praktisch kann immer nur eine der Alternativen erfüllt sein, denn *tag* enthält einen ganz bestimmten Wert. Allerdings besteht auch die – zumindest theoretische Chance – dass keine der Bedingungen zutrifft. Diesen Zustand abzufangen, ist mit *If*-Anweisungen dieser Art schon schwieriger. Im Abschnitt »Mehrfachbedingungen mit *Select Case*« ab Seite 169 finden Sie eine elegantere Alternative.

Else Wenn man weiter zu komplexeren Entscheidungsbäumen vordringt, wären Verschachtelungen unvermeidlich. Prinzipiell kann die *If*-Anweisung unbegrenzt verschachtelt werden, das heißt, in jedem Block kann sich wiederum eine *If*-Anweisung verstecken. Das führt in aller Regel nicht zu besonders gut lesbarem Code und sollte vermieden werden. Es gibt fast immer elegantere Lösungsmöglichkeiten. Oft ist es notwendig, nicht nur auf das Eintreten eines Ereignisses zu reagieren, sondern auch die negative Entscheidung zu behandeln. In solchen Fällen wird die *If*-Anweisung um *Else* (dt. sonst) ergänzt. Die Anweisung oder der Block hinter *Else* wird ausgeführt, wenn die Bedingung nicht zutrifft.

```
Dim tag As Short = 7
If tag = 0 Or tag = 7 Then
    Response.Write ("Wochenende!")
Else
    Response.Write ("Arbeitszeit")
End If
```

Listing 3.5: *If* mit alternativem *Else*-Zweig (*VbKeywordElse.aspx*)

Das letzte Beispiel könnte man auch mit *If* nach dem *Else* schreiben. Praktisch wird dabei in dem *Else*-Zweig noch eine weitere *ElseIf*-Anweisung eingebaut. Das folgende Beispiel zeigt eine mögliche Anwendung:

```
Dim tag As Short = 5
If tag = 0 Or tag = 7 Then
    Response.Write ("Wochenende!")
ElseIf tag = 5 Then
    Response.Write ("Fast Wochenende")
Else
    Response.Write ("Arbeitszeit")
End If
```

Listing 3.6: Folgen von If und Elseif (VbKeywordIfElse.aspx)

Mehrfachbedingungen mit Select Case

Wenn Sie, wie gezeigt, mehrere aufeinanderfolgende Bedingungen gegen ein und dieselbe Variable testen möchten, ist die *If*-Anweisung sehr aufwändig. Mit *Select Case* steht eine weitere Anweisung zur Verfügung, die solche Listen eleganter aufbaut:

```
<%
Dim stunde As Short = 9
Select Case stunde
    Case 8
        Response.Write("Guten Morgen")
    Case 9
        Response.Write("Bisschen spät heute?")
    Case 10
        Response.Write("Jetzt gibt's Ärger")
    Case 11
        Response.Write("Lass dich krankschreiben")
    Case Else
        Response.Write("Sonstwann am Tage...")
End Select
%>
```

Listing 3.7: Mehrfachbedingung mit Select Case aufbauen (VbKeywordSelect.aspx)

Der prinzipielle Aufbau ist aus dem Beispiel ersichtlich. In der *Select Case*-Anweisung selbst steht die zu testende Variable, in den *Case*-Abschnitten jeweils der Testwert, der auf Gleichheit getestet wird. *Case 8* entspricht also *stunde = 8*.

Im Beispiel wurde bereits der Befehlsbestandteil *Case Else* genutzt. Dieser Zweig der *Select*-Anweisung wird ausgeführt, wenn keine andere Bedingung zutrifft. Die Position ist wichtig, *Case Else* muss immer am Ende der Liste stehen.

Case Else

Weitere Entscheidungsfunktionen der Kompatibilitätsbibliothek

Drei in der VB-Welt beliebte Entscheidungsfunktionen haben in der Kompatibilitätsbibliothek auch den Wechsel nach VB.NET überlebt.

Die Bedingungs-funktion

VB.NET kennt eine Kurzschreibweise für einen einfachen Bedingungsbehehl, die so genannte Bedingungsfunktion:

```
IIf(Bedingung, Wahr-Ausdruck, Falsch-Ausdruck)
```

Damit lassen sich Abfragen oft kürzer, aber selten lesbarer gestalten. Außerdem ist `IIf` eine Funktion und kein Sprachelement, wodurch der Aufruf langsamer ist als der einer echten `If`-Anweisung.

Listenauswahl

VB.NET kennt zwei weitere Entscheidungsfunktionen, die kein Äquivalent in anderen Sprachen kennen. Mit *Choose* kann aus einer Liste ein Wert anhand eines Indizes ausgewählt werden. Der Index darf 0 oder maximal die Anzahl der Element minus Eins sein. Folgende Syntax gilt:

```
var = Choose(2, "Wert1", "Wert2", "Wert3", "Wert4")
```

Ähnliche Aufgabe erfüllt *Switch*, womit anhand logischer Bedingungen eine Auswahl aus einer Liste getroffen werden kann. VB.NET akzeptiert maximal sieben Parameterpaare. Die Syntax können Sie folgendem Beispiel entnehmen:

```
var = Switch(z < 1, "Z0", z > 1 And z < 4, "Z3")
```

Beachten Sie, dass die Parameter nur paarweise auftreten können.

Unbedingte Sprünge mit GoTo

Spaghetticode?

Wenn von VB.NET als inzwischen moderner Programmiersprache die Rede ist, mag das Auftreten eines regulären *GoTo*, also des unbedingten Sprungs zu einer Marke, wie ein Rückschritt erscheinen. Denn seit den Zeiten von GW-BASIC wurde der mit *GoTo* mögliche Spaghetti-Code verteufelt. Tatsächlich fehlten aber früher elegante Schleifenanweisungen, sodass auf den Sprung nur selten verzichten werden konnte. Es ist richtig, dass man generell jedes erdenkliche Problem auch ohne direkte Sprünge lösen kann, leider nicht immer elegant. Es gibt seltene Fälle, in denen *GoTo* tatsächlich einiges an Schreiarbeit und damit Fehlerquellen in der

Programmlogik erspart. Trotzdem wird es nur sehr selten gebraucht, und wenn Sie sich nicht sicher sind, dass das vorliegende Problem tatsächlich nur damit den kürzesten Code ergibt, verzichten Sie lieber auf den Einsatz.

In den Fällen, in denen es angebracht erscheint, definieren Sie das Sprungziel mit einem so genannten Label:

Sprungziel:

Dann können Sie dieses – allerdings nur innerhalb desselben Blockes – folgendermaßen erreichen:

```
GoTo "Sprungziel"
```

Aus Methoden, Blöcken, Schleifen etc. können Sie nicht heraus springen. Insofern ist das moderne *GoTo* in VB.NET ziemlich resistent gegen den Missbrauch zur Produktion von Spaghetticode.

Bedingte Sprünge auf eine Marke im Fehlerfall

Prinzipiell können Sie auch beim Auftreten eines Laufzeitfehlers auf ein Sprungziel verzweigen. Dies erfolgt durch folgende Anweisung:

```
On Error GoTo Sprungziel
```

Aufheben kann man die Wirkung folgendermaßen:

```
On Error GoTo 0
```

Mit VB.NET hält allerdings eine moderne und leistungsfähige Fehlerbehandlung Einzug, die auf *Try...Catch...Finally* basiert. Sie wird in Abschnitt 3.4.10, »Ausnahmebehandlung« ab Seite 208 ausführlich vorgestellt.

3.4.7 Schleifen

Schleifen benötigen Sie, um Programmteile mehrfach durchlaufen zu lassen. Neben der Einsparung an Tipparbeit ist vor allem die variable Festlegung der Schleifendurchläufe interessant. Schleifen ohne feste Laufvariable werden durch eine Bedingung gesteuert. Der Zustand des logischen Ausdrucks bestimmt, ob die Schleife weiter durchlaufen wird oder nicht.

Steuerung am Eintrittspunkt: While...End While



Die häufigste Schleifenart ist die While...End While-Schleife (dt. während), die in fast jeder Programmiersprache zu finden ist. Die Bedingung wird mit jedem Eintritt in die Schleife getestet. Solange der Ausdruck True zurückgibt, wird die Schleife durchlaufen. Wenn der Ausdruck also schon beim Eintritt in die Schleife False ergibt, wird die Schleife überhaupt nicht durchlaufen.

```
<%  
Dim counter As Integer = 0  
Dim test As Integer = 6  
While test > counter  
    Response.Write ("Aktueller Zähler: " & counter.ToString() &  
" <br/>")  
    counter += 1  
End While  
%>
```

Listing 3.8: Einfache While...End While-Schleife (VbKeywordWhile.aspx)

In diesem Skript werden zuerst zwei Variablen definiert. Eine wird als Laufvariable mit wechselnden Werten eingesetzt, die andere zur Steuerung der Abbruchbedingung. Mit dem ersten Aufruf der While-Anweisung wird die Bedingung geprüft. Es kann also vorkommen, dass die Befehle im Körper der Schleife überhaupt nicht ausgeführt werden.

Schlüsselwort - while

```
Aktueller Zähler: 0  
Aktueller Zähler: 1  
Aktueller Zähler: 2  
Aktueller Zähler: 3  
Aktueller Zähler: 4  
Aktueller Zähler: 5
```

Abbildung 3.5: Ausgabe mit While-Schleife

Universelle Schleifen mit Do...Loop

Der Test der Bedingung am Anfang hat einen wesentlichen Nachteil, wenn der Inhalt des Blocks für die weitere Programmfortsetzung unbedingt erforderlich ist. Es ist möglich, dass die Bedin-

gung so wirkt, dass der Inhalt nie durchlaufen wird. Um das jedoch sicherzustellen, kann die *Do...Loop*-Schleife verwendet werden. Der einzige Unterschied zu *While* besteht in der Art der Abarbeitung und den erweiterten Bedingungsschlüsselwörtern. Zuerst wird die Schleife einmal durchlaufen und am Ende wird die Abbruchbedingung getestet. Auch dann, wenn die Abbruchbedingung beim Schleifeneintritt *False* ist, wird der Block mindestens einmal ausgeführt.

```
<%
Dim counter As Integer = 0
Dim test As Integer = 6
Do
    Response.Write ("Aktueller Zähler: " & counter.ToString() & "<br/>")
    counter += 1
Loop While test > counter
%>
```

Listing 3.9: Einfache Anwendung der *do*-Schleife (*VbKeywordDoLoop.aspx*)

Die Bedingung kann bei *Do...Loop* sowohl am Schleifenanfang als auch am Schleifenende stehen. Außerdem stehen als Bedingungsvarianten sowohl *While* als auch *Until* zur Verfügung. Während *While* die Schleife solange durchlaufen lässt, wie die Bedingung erfüllt ist (*True*), bricht *Until* ab, wenn die Bedingung erfüllt ist und lässt die Schleife laufen, solange sie unerfüllt ist.

Verlassen von Schleifen mit Exit

Die Problematik der Abbruchbedingung kann oft umgangen werden, indem zusätzlich ein Notausstieg eingebaut wird. Das folgende Listing zeigt eine fehlerhaft programmierte Schleife – die Abbruchbedingung wird regulär nie erfüllt. Der Notausstieg verwendet die schon bekannte *Exit*-Anweisung, die die Ausführung an die nächsthöhere Programmebene zurückgibt. Hinter *Exit* folgt der Name der Block-Anweisung, die verlassen werden soll. Neben *Do* und *While* kann dies auch *Function*, *Sub*, *For* usw. sein.

Exit

```
<%
Dim counter As Integer = 30
Dim test As Integer = 6
While counter > test
    Response.Write("Aktueller Zähler: " & counter.ToString() & "<br/>")
    Exit While
End While
```

```

        counter += 1
    If counter = 50 Then Exit While
End While
Response.Write("Schleifenende erreicht bei: " &
counter.ToString())
%>

```

Listing 3.10: Schleife mit Notausstieg mittels Exit While (VbKeywordExitWhile.aspx)

Wie es funktioniert

Die Schleife entspricht der Vorhergehenden. Wenn jedoch wegen falscher Startwerte die Variable *counter* bis auf 50 erhöht wird, unterbricht *Exit While* die Ausführung der Schleife. Anschließend setzt das Programm nach dem Ende *While*-Anweisung fort. Der Begriff Notausstieg sollte hier nicht überbewertet werden. Die Anwendung der *Exit*-Anweisung ist eine reguläre Programmier Technik. Als Anfänger sollten Sie sich aber vielleicht die eine oder andere *Exit*-Anweisung einbauen, um sicher durch alle Schleifen zu kommen. Profis werden besser einschätzen können, ob die gewählten Schleifenbedingungen allen Programmsituationen genügen werden. Der Einsatz von *Exit* ist generell nur im Zusammenhang mit *If...Then* sinnvoll.

Die Anwendung ist in gleicher Form auch für *Do* möglich – dann als *Exit Do*. Ebenso lassen sich die nachfolgend beschriebenen *For*- und *For Each*-Schleifen damit steuern. Allerdings ist der Einsatz in Zählschleifen wegen der komplexeren Steuerung nicht anzuraten – ein Stören des internen Zählers kann zu Programmierfehlern führen, die äußerst schwer nachvollziehbar sind.

Abzählbare Schleifen mit For...Next

Die vorangegangenen Beispiele dienten vor allem der Erläuterung der Syntax der Befehle; die feste Vorgabe von unteren und oberen Grenzen ist keine typische Anwendung der *While*-Schleifen. In solchen Fällen setzen Sie besser *For...Next*-Schleifen ein. Die Abbruchbedingung ist allerdings auch hier ein normaler logischer Ausdruck. Zusätzlich kann eine numerische Variable mitgeführt werden – die Zählvariable.

For...Next

Alle Parameter dieser Anweisung sind optional. Bei vollständiger Angabe ist die *For*-Schleife jedoch komplexer als die bisher behandelten Schleifentypen:

```
For laufvariable = startwert To zielwert Step Schrittweite
```

Dies ist die einfachste Form der Anwendung. Das folgende Listing zeigt Schrift in verschiedenen Größen an:

```
<%  
Dim i As Integer  
For i = 10 To 24 Step 2  
    Response.Write ("<div style=""font-size:" &  
                    & i.ToString() & "pt""/>")  
    Response.Write ("For...Next-Schleife</div>")  
Next  
%>
```

Listing 3.11: Klassische For...Next-Schleife (VbKeywordFor.aspx)

Die Schleife arbeitet mit der Laufvariablen *i*. Der Startwert ist 10. Die Schleife wird solange durchlaufen, wie *i* kleiner oder maximal gleich 24 ist. Nach jedem Durchlauf wird die Zählvariable *i* um zwei erhöht.

Schlüsselwort - For...Next

```
For...Next-Schleife  
For...Next-Schleife  
For...Next-Schleife  
For...Next-Schleife  
For...Next-Schleife  
For...Next-Schleife  
For...Next-Schleife  
For...Next-Schleife
```

Abbildung 3.6: Ausgabe von formatiertem HTML-Code mit For...Next

Die Angabe der Schrittweite mit *Step* kann entfallen, wenn mit der Standarderhöhung von Eins (1) gezählt werden soll. Der Wert kann im Bedarfsfall auch negativ sein, um rückwärts zu zählen. Ein vorzeitiges Verlassen der Schleife ist mit *Exit For* möglich.

Aufzählungen und Arrays sequenziell durchlaufen

In VB.NET und im .NET-Framework wird sehr oft mit Arrays, Kollektionen oder Aufzählungen gearbeitet. Beispiele finden Sie in den Abschnitten zu Arrays und Aufzählungen. Die grundlegende Syntax lautet:

```
For Each element_variable As Typ In complextyp ' BefehleNext
```

Danach folgt meist ein Block. Das Schlüsselwort *In* gehört zu der Anweisung. *For Each...Next* durchläuft jedes Element eines Arrays oder einer Aufzählung und weist dieses jeweils der Variable links im Ausdruck zu. Wenn diese vorher nicht bereits deklariert wurde, kann der Datentyp auch direkt in der Anweisung angegeben werden (As Typ).

3.4.8 Namensräume und Klassen

Das gesamte Framework besteht aus Klassen, sortiert in Namensräume. ASP.NET erzeugt aus dem Code einer *aspx*-Seite immer eine Klasse, die selbst von der eingebauten Klasse *Page* erbt, um der Seite die gewünschte Funktionalität zur Verfügung zu stellen. Es ist nahe liegend, dass die praktische Programmierung des Frameworks nicht nur Kenntnisse über Objekte verlangt, sondern den Einsatz in eigenen Projekten geradezu herausfordert.

Eine allgemeine Einführung in Objekte finden Sie in Abschnitt 1.4.3, »Die Welt der Objekte« ab Seite 64.



VB.NET unterstützt wie der Vorgänger VB 6 das Konzept der Module. Damit wurde in der klassischen prozeduralen Programmierung eine feinere Teilung des Codes erreicht, um die Codeblöcke nicht unbeherrschbar anwachsen zu lassen. Mit der vollständigen Unterstützung objektorientierter Konzepte sind Module eigentlich obsolet, denn Namensräume und Klassen »modularisieren« weit besser. In diesem Buch vertreten die Autoren die Ansicht, dass man nur dann erfolgreich im Framework programmieren kann, wenn das Prinzip der objektorientierten Programmierung akzeptiert und angewendet wird. VB-Module werden deshalb nicht behandelt.

Die objektorientierte Programmierung mit VB.NET

Einfache Klassen

Klassen sind Bauanleitungen für Objekte. Sie werden einmal deklariert und dann durch Instanziierung von Objekten verwendet. Allerdings gibt es den Spezialfall der statischen Klasse, aus der nur ein einziges Objekt entsteht, womit die explizite Instanziierung entfällt.

Zuerst jedoch eine einfache Klassendefinition:

```
Class KlassenName
    ' Deklaration der Mitglieder
End Class
```

Es ist ein Grundprinzip der objektorientierten Programmierung, dass aus einer Basisklasse andere abgeleitet werden können. Damit es kein heilloses Durcheinander von Eigenschaften, Methoden und Ereignissen gibt, kann der Entwickler einer Klasse festlegen, wo die Mitglieder der Klasse sichtbar sind. Dasselbe gilt für die Klasse selbst. Die Steuerung des Sichtbereiches wird durch so genannte »Zugriffsmodifizierer« erreicht. Durch die Beschränkung von Variablen auf eine Klasse werden diese quasi versteckt. Man bezeichnet das allgemein als »Kapselung«.

Allzu komplizierte Abhängigkeiten von Klassen oder gar der Aufbau einer Hierarchie, wie sie das Framework darstellt, ist in eigenen Projekten extrem selten notwendig. Nur wenn Bibliotheken (Sammlungen von Klassen) entwickelt werden, die fremde Entwickler nutzen sollen, ist eine feinere Abstufung sinnvoll.



Der Sichtbereich kann nicht nur für eine Klasse, sondern auch für jedes Mitglied festgelegt werden. In VB.NET werden dazu folgende Schlüsselwörter verwendet:

Mitglieder

Schlüsselwort	Bedeutung
Public	Öffentlich, überall sichtbar
Protected Friend	Kombiniert Friend und Protected
Friend	Sichtbar im selben Programm
Protected	Öffentlich, sichtbar in direkt erbenden Klassen und allen dort vorhandenen Mitgliedern
Private	Nur innerhalb der Klasse und aller ihrer Mitglieder

Tabelle 3.12: Zugriffsmodifizierer in der Reihenfolge der Sichtbarkeit (abnehmend)

Generell sollten Mitglieder immer den kleinstmöglichen Sichtbereich haben, damit Seiteneffekte ausgeschlossen werden. Deklarieren Sie deshalb vorzugsweise mit `Private` und steigern Sie dann bis `Public`, wenn die Logik des Programms dies in Einzelfällen erfordert.

Die Klasse selbst kann nur mit `Private`, `Friend` oder `Public` deklariert werden.

Klassen



In der ASP.NET-Praxis kann man es etwas einfacher handhaben. Der Zugriff auf die »Mutterklasse« Page und deren Mitglieder (meist Steuerelemente) erfolgt mit `Protected`, eigene Mitglieder sind `Private` und öffentliche Ereignisbehandlungsmethoden `Public`.

Ereignisse im Kontext von VB.NET

Die objektorientierte Programmierung unterliegt bestimmten Paradigmen (Denkmustern), die Sie verstehen müssen, um .NET erfolgreich verwenden zu können. Auf den Begriff der Klasse wurde bereits eingegangen. Aus Klassen werden Objekte instanziiert, auch dies ist bereits erwähnt worden. Objekte existieren jedoch nicht allein und warten darauf, benutzt zu werden, sondern können gewissermaßen ein Eigenleben entwickeln. Diese Fähigkeit zur Kommunikation basiert auf Ereignissen. Ereignisse können an bestimmte externe Aktionen geknüpft sein, beispielsweise ein Klick auf eine Schaltfläche oder die Änderung von Daten in einer Datenbanktabelle. Das zuständige Objekt verfügt über eine so genannte Ereignisbehandlungsmethode. Die Verknüpfung zwischen dem Ereignis und der Methode kann explizit über Schlüsselwörter oder implizit über Namen erfolgen.

Ereignisse behandeln

Wenn Sie im Designer von Visual Studio .NET eine Schaltfläche (Button-Steuerelement) anlegen, können Sie das Standardereignis »Mausklick« sehr einfach damit verknüpfen – ein Doppelklick genügt. In der Code-Ansicht wird dann die passende Ereignisbehandlungsmethode erstellt. Wenn Sie andere Ereignisse benötigen, gehen Sie folgendermaßen vor:

1. Wechseln Sie mit **[F7]** in die Code-Ansicht.
2. Wählen Sie aus der linken DropDown-Liste das Schaltflächen-Objekt aus.
3. In der rechten Liste der Deklarationen finden sie nun alle Ereignisse (jeweils mit einem gelben Blitz davor).
4. Wählen Sie die benötigte Methode aus. Wenn eine Definition bereits erfolgte, ist der Eintrag fett, der Cursor wird dann auf den vorhandenen Code gesetzt. Die Auswahl eines noch nicht definierten Ereignisses führt zum Erzeugen des Rumpfes der Ereignisbehandlungsmethode.

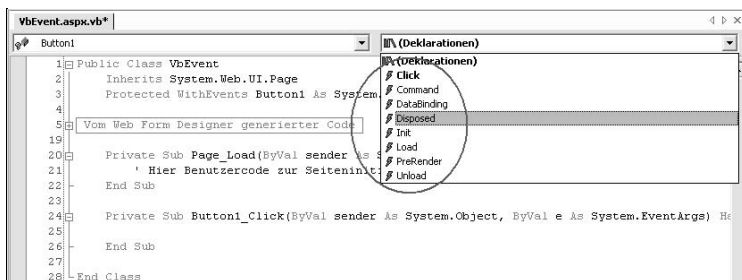


Abbildung 3.7: Erzeugen von Ereignisbehandlungsmethoden

Nach der Definition der Ereignisbehandlungsmethode kann nun deren Nutzung definiert werden. Das folgende Beispiel »meldet« den Klick durch Setzen des Inhalts eines Label-Steuerelements.

```
Public Class VbEvent
    Inherits System.Web.UI.Page
    Protected WithEvents 1Click As Label
    Protected WithEvents Button1 As Button

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        ' Hier Benutzercode zur Seiteninitialisierung einfügen
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles Button1.Click
        Dim b As Button = CType(sender, Button)
        1Click.Text = "Auf dem Button steht: " & b.Text
    End Sub
End Class
```

Listing 3.12: Ereignisbehandlung einer Schaltfläche (VbEvent.aspx.vb zu VBEvent.aspx)

Um die Abwicklung des Ereignisses sicherzustellen, wirken hier zwei Schlüsselwörter zusammen. Zuerst wird die Übernahme aller Ereignisse mit **WithEvents** sichergestellt. Da .NET Standardroutinen bereit stellt, die alle »übrig gebliebenen« Ereignisse abfangen, müssen Sie nicht alle denkbaren Fälle behandeln, sondern nur die, die wirklich relevant sind. Im Beispiel wird nur das Klickereignis ausgewertet. Abgesehen von der Erstellungshilfe im Code-Fenster können Sie sich bei der Auswahl auch auf Intellisense verlassen. Die Zuordnung der Ereignisbehandlungsmethode

WithEvents
Handles

de zu einem Ereignis erfolgt mit dem Schlüsselwort `Handles`. Sie müssen danach die Ereignisquelle (hier: `Button1`) und dann das Ereignis (`Click`) schreiben.

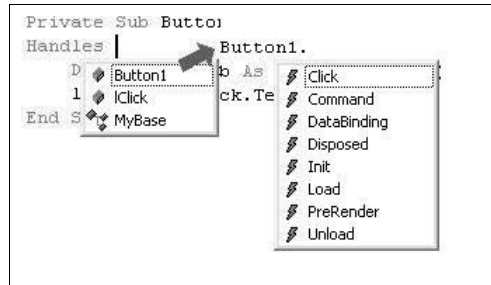


Abbildung 3.8: Intellisense hilft bei der Ereignisauswahl.

Dynamische Verknüpfung der Ereignisse

Neben dieser Form der statischen Festlegung von Ereignissen kann die Zuordnung auch dynamisch, zur Laufzeit erfolgen. Auch hier werden besondere Schlüsselwörter verwendet. Mit `AddHandler` wird ein Ereignis ausgewählt. Der zweite Parameter weist dann die Adresse der Methode zu. Dazu dient der Operator `AddressOf`. Das folgende Beispiel zeigt, wie die Ereignisbehandlungsmethode auf Grundlage eines `CheckBox`-Steuerelements zugewiesen wird. Zuerst die HTML-Vorlage:

```
<body>
  <form id="Form1" method="post" runat="server">
    <H1>Dynamisch Ereignisse zuweisen</H1>
    <P>
      <asp:Button id="Button1" runat="server"
        Text="Klick mich!" />
      <asp:CheckBox id="cbAuswahl" runat="server"
        Text="Kein Standardereignis" /></P>
    <P>
      <asp:Label id="lAusgabe" runat="server">
        Noch nicht geklickt
      </asp:Label>
    </P>
  </form>
</body>
```

Listing 3.13: Testprogramm für Ereignisbehandlung (`VBEventsDynamic.aspx`)

Die Zuweisung der Standardmethode *Button1_Click* erfolgt implizit bei der Übersetzung:

**AddHandler
AddressOf**

```
Public Class VbEventsDynamic
    Inherits System.Web.UI.Page
    Protected WithEvents Button1 As Button
    Protected WithEvents cbAuswahl As CheckBox
    Protected WithEvents lAusgabe As Label

    Private Sub Page_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If cbAuswahl.Checked Then
            AddHandler Button1.Click, AddressOf Button1_Special
        End If
    End Sub

    Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles Button1.Click
        lAusgabe.Text = "Standardhandler geklickt"
    End Sub

    Private Sub Button1_Special(ByVal sender As Object, _
        ByVal e As EventArgs) _
        Handles Button1_Special
        lAusgabe.Text = "Spezial-Behandlungsroutine angeklickt"
    End Sub
End Class
```

Listing 3.14: Dynamische Zuweisung von Ereignisbehandlungsmethoden zu Ereignissen (VbEventsDynamic.aspx.vb)

Die Methode der alternativen Ereignisbehandlungsmethode wird erst zur Laufzeit zugewiesen:

```
AddHandler Button1.Click, AddressOf Button1_Special
```

Eine gesonderte *Handles*-Anweisung ist nicht erforderlich. Die Ausführung in dieser Reihenfolge funktioniert, weil im Abarbeitungszyklus der Seite zuerst die *Page_Load*-Methode verarbeitet wird und erst danach anstehende Ereignisse der Steuerelemente behandelt werden.



Abbildung 3.9: Steuerung der Ereigniswege

Entfernen mit RemoveHandler

Neben der Zuweisung einer Ereignisbehandlungsmethode kann auch ein Entfernen erforderlich sein. In solchen Fällen hilft die Anweisung `RemoveHandler` weiter. Syntax und Verwendung sind mit `AddHandler` identisch:

```
RemoveHandler Button1.Click, AddressOf Button1_Special
```

Neben Intellisense liefert die Online-Hilfe Informationen darüber, welche Ereignisse ein spezifisches Steuerelement auslösen kann. Beachten Sie in allen Fällen, in denen serverseitige Ereignisse verarbeitet werden sollen, dass die Informationen erst vom Browser zum Server gelangen müssen. Dazu muss – direkt über die Sendeschaltfläche oder indirekt über JavaScript – ein Formular abgesendet oder ein Link angeklickt werden. Die Reaktion erfolgt deshalb für den Benutzer unter Umständen leicht

Der Lebenszyklus eines Objekts

Jedes Objekt durchläuft einen Lebenszyklus, der folgende Schritte umfasst:

1. Anlegen einer Referenz auf die Klasse (Referenzieren):

```
Dim object As Klasse
```

2. Erzeugen einer Instanz der Klasse (Instanzieren):

```
object = New Klasse()
```

Die Schritte 1 und 2 können syntaktisch in einer Anweisung zusammengefasst werden:

```
Dim object As Klasse = New Klasse()
```

3. Dem Objekt können nun Starteigenschaften zugewiesen werden (Initialisieren):

Dieser Schritt ist optional. Es ist jedoch typisch, dass bestimmte Grundeigenschaften beim Erzeugen des Objekts gesetzt werden. Dazu dient im Allgemeinen der Konstruktor. Die Übergabe der Startparameter sieht folgendermaßen aus:

```
object = New Klasse(Parameter, Parameter, ...)
```

Wie die Parameter gestaltet werden, hängt vom Konstruktor ab. Abschnitt »Aufbau und Zerstörung: Von Konstruktoren und Destrukturen« ab Seite 193 zeigt, wie es geht.

4. Das Objekt ist nun fertig und kann verwendet werden. Dies geschieht durch Aufruf von Methoden sowie Schreiben und Lesen von Eigenschaften.
5. Am Ende des Lebenszyklus wird das Objekt deinitialisiert. Dazu dient ein so genannter Destruktor. Der Destruktor ist ein allgemeiner Begriff für eine Methode, die zum Beenden der Lebensphase aufgerufen wird. In .NET werden die Destrukturen als Finalizer bezeichnet. Die Verwendung ist optional. Die Standardmethode für den Aufruf heißt `Dispose`:

```
object.Dispose()
```

Der genaue Zeitpunkt des Destruktor-Aufrufs kann nicht sicher bestimmt werden, weil die Zerstörung im Speicher durch den Garbage Collector erfolgt, der nur dann startet, wenn keine höher priorisierten Tasks laufen. Es ist zwar sicher, dass der Aufruf erfolgt, nicht jedoch der Zeitpunkt.

6. Nach dem Deinitialisieren erfolgt die Dereferenzierung, bei der das Objekt physisch aus dem Speicher entfernt wird. Auch dieser Vorgang kann gegebenenfalls verzögert ablaufen, da er unter der Kontrolle des Garbage Collectors steht:

```
object = Nothing
```

Die Schritte 5 und 6 werden automatisch abgewickelt, wenn ein Programm oder eine Website beendet wird. Die explizite Vernichtung des Objekts ist nur angebracht, wenn kurzfristig Speicher benötigt wird. Dies ist beispielsweise der Fall, wenn sehr große Datenbankobjekte mit hohem Speicherbedarf instanziiert werden.

Funktionen und Prozeduren sind Methoden in Klassen

Der Begriff Funktion oder Prozedur hat in der objektorientierten Welt ausgedient. Mitglieder von Klassen, die »etwas tun«, werden als Methoden bezeichnet. Die Syntax von VB.NET verlangt lediglich in Bezug auf den Rückgabewert eine Unterscheidung. Gibt eine Methode etwas zurück, wird das Schlüsselwort `Function` verwendet, ansonsten `Sub`. Nichtsdestotrotz handelt es sich in jedem Fall um eine Methode und im weiteren wird hier nicht mehr unterschieden, auch wenn je nach Anwendungsfall das eine oder andere Schlüsselwort zum Einsatz kommt.

Function	Grundsätzlich wird folgende Syntax für eine Methode mit Rückgabewert verwendet: <pre> Function Name (Parameter As Typ) As Typ ' Code If Bedingung Then Exit Function ' Code Return Rückgabewert End Function </pre>
Sub	Ohne Rückgabewert wird das Schlüsselwort <code>Sub</code> verwendet: <pre> Sub Name (Parameter As Typ) ' Code If Bedingung Then Exit Sub ' Code End Sub </pre>
Exit	Zusätzlich kann den Methoden als Mitglied einer Klasse ein Zugriffsmodifizierer wie <code>Public</code> oder <code>Private</code> vorangestellt werden. Die <code>Exit</code>-Anweisung führt zum vorzeitigen Verlassen der Methode und muss deshalb in einer <code>If</code>-Anweisung stehen.
Parameter	Die Parameter sind in beiden Fällen optional, die runden Klammern müssen jedoch immer geschrieben werden. Wenn Parameter verwendet werden, gilt allgemein der gezeigte Aufbau <i>Parameter-Name As Typ</i>. Werden Arrays übergeben, sind hinter den Parameternamen runde Klammern zu setzen: <i>ParameterArray() As Typ</i>.
ByVal ByRef	Wenn Sie Parameter in dieser Form in Visual Studio .NET eintippen, werden Sie feststellen, dass vor dem Parameternamen das Schlüsselwort <code>ByVal</code> hinzugefügt wird. Dies sieht dann folgendermaßen aus: <pre> Sub Name (ByVal Parameter As Typ) </pre> Es gibt zwei Arten der Parameterübergabe. Da Variablen nur einen begrenzten Gültigkeitsbereich besitzen, erfolgt normalerweise nur die Übergabe einer Kopie des aufrufenden Wertes. Betrachten Sie den folgenden Code: <pre> Dim a As String Klasse.Name(a) </pre> Ist in der Methode <i>Name</i> der Klasse <i>Klasse</i> der Parameter mit <code>ByVal</code> gekennzeichnet, wird nicht das Objekt <i>a</i>, sondern eine Kopie übergeben – quasi nur der Wert (<code>ByVal</code> stammt von »by value«, zu deutsch »als Wert«). Es kann notwendig sein, dass eine Methode

Zugriff auf das Original erhalten soll. Dann können Sie das Schlüsselwort `ByRef` (»by reference«, als Verweis) verwenden.

Änderungen innerhalb der Methode wirken sich nun auf das aufrufende Objekt aus. Das setzt natürlich voraus, dass das zu ändernde Objekt änderbar ist – Konstanten sind hier nicht erlaubt.

Das folgende Beispiel zeigt eine einfache Anwendung: Text wird durch Hinzufügen entsprechender Tags verändert:

```
Public Class VbMethodReference
    Inherits System.Web.UI.Page
    Protected WithEvents lAusgabe As Label

    Private Sub fett(ByRef text As String)
        text = String.Format("<b>{0}</b>", text)
    End Sub

    Private Sub Page_Load(ByVal sender As System.Object, 
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim s As String
        s = "Ein einfacher Text"
        Me.fett(s)
        Me.lAusgabe.Text = s
    End Sub
End Class
```

Listing 3.15: Übergabe von Parametern als Referenz
(Code-Datei `VbMethodReference.aspx.vb` zu `VbMethodReference.aspx`)

Die Ausgabe erfolgt an ein `Label`-Steuerelement, das hier nur der Demonstration dient. Dass der Parameter innerhalb der Methode einen anderen Namen hat, spielt keine Rolle.

Neben der Änderung von Parametern können Sie auch leere Objekte übergeben und durch eine Methode füllen lassen. Funktionen haben nur einen Rückgabewert. Werden mehrere benötigt, helfen Methodenaufrufe mit referenzierten Parametern weiter.

Anwendung

Bis auf bestimmte Situationen ist die Vorgabe `ByVal` grundsätzlich richtig. Die Entkopplung der Adressräume, wie sie standardmäßig stattfindet, verringert das Fehlerrisiko und führt zu lesbareren Programmen. Überlegen Sie sorgfältig, ob `ByRef` wirklich notwendig ist und setzen Sie es nur dann ein, wenn es keine andere Form der Parametermanipulation gibt. Oft werden Eigenschaften zum Wertaustausch besser geeignet sein. Dazu finden Sie mehr im folgenden Abschnitt.



Optionale Parameter

Manchmal kann es notwendig sein, erst zur Laufzeit und beim Aufruf einer Methode zu entscheiden, ob ein Parameter übergeben wird oder nicht. Sie können dann das Schlüsselwort `Optional` benutzen, um anzuzeigen, dass der Parameter entfallen kann. Damit der fehlende Wert nicht zu Störungen im Ablauf führt, geben Sie im Methodenkopf einen Standardwert an. Diese Angabe ist zwingend erforderlich. Die Syntax sieht folgendermaßen aus:

```
Function Calc(ByVal Wert As Decimal, 
               Optional ByVal Kurs As Decimal = 1.95583D) As
    Decimal
        ' Implementierung
End Function
```

Der Aufruf kann nun in zwei Varianten erfolgen:

```
var = Calc(100)

var = Calc(100, 2.32)
```

Optionale Parameter müssen zwingend am Ende der Parameterkette stehen, sonst kann die Platzierung nicht ermittelt werden. Falls Sie damit nicht zurecht kommen, bleibt die Möglichkeit der Überladung. Bei diesem Verfahren werden gleichnamige Methoden mit verschiedenen Parameterlisten (so genannte Signaturen) geschrieben.

Überladung

Bei der Überladung schreiben Sie einfach mehrere gleichnamige Methoden in eine Klasse. Der Compiler erkennt anhand der Signaturen die passende Zuordnung und löst den Aufruf eindeutig auf. Das Verfahren ist vorteilhaft, wenn die Logik des Programms oder der Klasse verschieden benannte Methoden nicht unterstützt oder die Lesbarkeit deutlich verbessert wird. Überdies unterstützt Intellisense die Nutzung.

Das Beispiel mit dem Umrechnungskurs ließe sich statt mit einem optionalen Parameter auch mit Überladung lösen:

```
Function Calc(ByVal Wert As Decimal, 
               ByVal Kurs As Decimal) As Decimal
    ' Implementierung
End Function

Function Calc(ByVal Wert As Decimal) As Decimal
    Dim Kurs As Decimal = 1.95583D
End Function
```



Abbildung 3.10: Überladungen im Visual Studio .NET

Aus der Überladung und der Vererbung ergeben sich komplizierte Gestaltungsformen von Klassen, die sich durch so genannte Vielgestaltigkeit (Polymorphie) auszeichnen. Lesen Sie mehr zur Vererbung im Abschnitt »Das Prinzip der Vererbung« ab Seite 194.

Zugriff auf Mitglieder und spezielle Deklaration von Eigenschaften

Der Zugriff auf Mitglieder von Objekten erfolgt in der bereits mehrfach gezeigten Punktschreibweise, wie sie seit langem in VB benutzt wird:

```
var = object.Eigenschaft
```

```
var = object.Methode()
```

Damit das funktioniert, muss eine Eigenschaft mit dem Zugriffsmodifizierer `Public` gekennzeichnet werden. Ein solche Eigenschaft wird folgendermaßen erstellt⁴:

```
Public Class Point
    Public x, y As Integer
End Class
```

```
Dim p As Point
p.x = 0
p.y = 100
```

Neben dieser Form einfacher Eigenschaften kennt VB.NET auch differenzierte Konstruktionen von Eigenschaften, die mit speziellen Schlüsselwörtern eingerichtet werden können. Der allgemeine Aufbau einer Eigenschaft gehorcht folgendem Muster:

Property

⁴ Das Beispiel ist etwas konstruiert; in einem so einfachen Konstrukt wäre eine Struktur besser geeignet als eine Klasse. Mehr Informationen dazu finden Sie in Abschnitt 3.4.9, »Strukturen und Aufzählungen« ab Seite 204.

```

Public Property Name()
    Set (ByVal Value)
        ' Verarbeitung der Zuweisung
    End Set
    Get
        Return "Rueckgabe"
    End Get
End Property

```

Get, Set ReadOnly WriteOnly

Das in älteren Basic-Versionen mögliche Schlüsselwort *Let* gibt es nicht mehr, weil ohnehin nur Objekte zugewiesen werden können (Sie erinnern sich: alles ist ein Objekt). Der Vorteil der Nutzung der Get-Set-Syntax liegt zum einen in der Einführung einer weiteren Abstraktionsebene. Der Entwickler der Klasse kann seine privat genutzten Variablen wirkungsvoll vor dem Zugriff schützen und bei der Annahme von Werten gleich eine Prüfung erzwingen. Ebenso kann die Ausgabe so aufbereitet werden, dass die Werte leicht weiterverarbeitet werden können, unabhängig von der Speicherform im Inneren der Klasse. Ein weiterer Effekt liegt in der unterschiedlichen Behandlung der Ein- und Ausgabewege. Werden Variablen benutzt, sind diese entweder unsichtbar (*Private*) oder vollständig nutzbar (*Public*). Lediglich die Verwendung in abgeleiteten Klassen ist differenzierter steuerbar. Mit Eigenschaften kann noch eine Steuerung des Schreib- und Leseweges erfolgen. Dazu entfällt entweder Set- oder der Get-Zweig. Zur besseren Lesbarkeit wird außerdem das Schlüsselwort *ReadOnly* oder *WriteOnly* hinzugefügt. Soll eine Eigenschaft nur lesbar sein, entfällt der Set-Zweig, eine nur schreibbare Eigenschaft verzichtet auf Get.

Das folgende Beispiel zeigt die Deklaration einer Klasse mit drei Eigenschaften und deren Verwendung. Die Ausgabe erfolgt an ein Label-Steuerelement, das hier nicht explizit gezeigt wird.

```

Public Class Kunde
    Private _vn As String
    Private _nn As String
    Public WriteOnly Property NachName()
        Set(ByVal Value)
            _nn = Value
        End Set
    End Property
    Public WriteOnly Property VorName()
        Set(ByVal Value)
            _vn = Value
        End Set
    End Property

```



```

    Public ReadOnly Property Name()
        Get
            Return String.Format("{0} {1}", _vn, _nn)
        End Get
    End Property

End Class

Public Class VbProperties
    Inherits System.Web.UI.Page

    Protected Namen As Label

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim k As Kunde = New Kunde()
        k.VorName = "Jörg"
        k.NachName = "Krause"
        Namen.Text = k.Name
    End Sub

End Class

```

Listing 3.16: Definition von Eigenschaften (VbProperties.aspx.vb)

Die Zuweisung des Verhaltens der Eigenschaften erfolgt durch die bereits erwähnten Schlüsselwörter `ReadOnly` und `WriteOnly`. Entfällt diese Angabe, sind beide Zweige erforderlich und die Eigenschaft kann gelesen oder geschrieben werden. Als alternative Zugriffsmodifizierer sind neben `Public` auch `Protected` und `Friend` erlaubt. Der Unterschied wird nur dann wichtig, wenn später von der Klasse geerbt wird.

Bei der Übergabe eines Wertes an eine Eigenschaft verhält sich diese syntaktisch wie eine Variablenzuweisung:

```
k.NachName = "Krause"
```

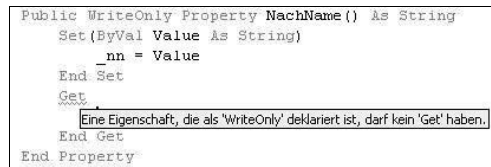
Die Zeichenkette »Krause« wird in diesem Fall über den Parameter `Value` an `Set` übergeben:

```
Set(ByVal Value)
```

Der Typ ergibt sich aus einer expliziten Angabe oder – wie im Beispiel – durch Annahme des universellen Typs `Object`. Voraussetzung dafür ist natürlich, dass `Option Strict` als `Off` deklariert wurde. Andernfalls müssen alle Deklarationen den passenden Datentyp erhalten, was im Beispiel folgendermaßen aussieht:

```
Public WriteOnly Property NachName() As String
    Set(ByVal Value As String)
        _nn = Value
    End Set
End Property
```

Wenn Sie die Schlüsselwörter `ReadOnly` bzw. `WriteOnly` verwenden, achtet auch der Editor im Visual Studio .NET darauf, dass der nicht zutreffende Zweig abgelehnt wird.



```
Public WriteOnly Property NachName() As String
    Set(ByVal Value As String)
        _nn = Value
    End Set
    Get
        'IntelliSense error: Eine Eigenschaft, die als 'WriteOnly' deklariert ist, darf kein 'Get' haben.'
    End Get
End Property
```

Abbildung 3.11: Intellisense erkennt, welche Zweige einer Eigenschaft zulässig sind.

Default VB.NET erlaubt außerdem die Festlegung *genau einer* Eigenschaft einer Klasse als so genannte Standardeigenschaft. Dies erfolgt durch Voranstellen des Schlüsselwortes `Default`. Die Standardeigenschaft darf nicht statisch sein (siehe nächster Abschnitt). Ebenso ist der Modifizierer `Private` ausgeschlossen, weil Standards nur sinnvoll sind, wenn sie extern benutzt werden. Darüber hinaus ist die Anwendung nur möglich, wenn die Eigenschaft Parameter verwendet. Folgende Deklaration nutzt Parameter:

```
Public Class Namen
    Private _vn As String
    Private _nn As String
    Default WriteOnly Property NachName(ByVal nn As String) As String
        Set(ByVal Value As String)
            _nn = Value
            _vn = nn
        End Set
    End Property
    ReadOnly Property Name() As String
        Get
            Return _vn & " " & _nn
        End Get
    End Property
End Class
```

Der Vorteil der Standardeigenschaft besteht lediglich in der Einsparung von Tipparbeit. Ein normaler Eigenschaftsaufruf mit Parametern sieht folgendermaßen aus:

```
n.NachName("Krause") = "Jörg"
```

Da *NachName* die Standardeigenschaft ist, kann verkürzend auch Folgendes geschrieben werden:

```
n("Krause") = "Jörg"
```

Eine Methode kann nicht als »Standard« deklariert werden. Allerdings spricht nichts dagegen in einer Eigenschaft Aktionen zu programmieren, wie es auch bei einer Methode möglich ist.



Statische Mitglieder für den Austausch zwischen Objekten

Statische Mitglieder von Klassen werden verwendet, um Informationen zu speichern, die für alle Objekte identisch sind. Darüber hinaus können die Instanzen miteinander kommunizieren. In VB.NET werden statische Mitglieder mit dem Schlüsselwort **Shared** deklariert.

Die Anwendung hat noch einen zweiten Effekt. Wenn das statische Mitglied nicht Teil eines Objekts ist, müsste es auch schon vor der Instanziierung zur Verfügung stehen. Genau das ist auch der Fall. Das folgende Beispiel rechnet Euro in DM um (falls Sie sich immer noch nicht daran gewöhnt haben, in Euro zu rechnen). Für solche Umrechnungsvorgänge wäre eine Instanziierung sinnlos. Mit einer statischen Methode ist die Benutzung der Klasse einfacher. Zuerst der HTML-Teil auf einen Blick:

Shared

```
<form id="Form1" method="post" runat="server">
  <H1>Statische Mitglieder</H1>
  <P>
    <asp:TextBox id="TextBox1" runat="server" Width="59px" />
    sind
    <asp:Label id="Label1" runat="server" Font-Bold="True" />
    &nbsp;  DM
  </P>
  <P>
    <asp:Button id="Button1" runat="server" Text="Berechnen" />
  </P>
</form>
```

Listing 3.17: Einfacher Euro-Rechner (VbStaticMember.aspx)

Die Klasse dient nun dazu, den Inhalt des `TextBox`-Steuerelements anzunehmen, zu prüfen und dann den zu der erfassten Zahl passenden DM-Wert auszugeben.

```
Public Class EuroDM
    Private Const _kurs As Double = 1.95583
    Public Shared Function Euro2DM(ByVal Euro As Object)
        If Euro.Equals(String.Empty) Then
            Return 0
        Else
            Return Cdbl(Euro) * _kurs
        End If
    End Function
End Class

Public Class VbStaticMember
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As Label
    Protected WithEvents TextBox1 As TextBox
    Protected WithEvents Button1 As Button

    Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles Button1.Click
        Me.Label1.Text = String.Format("{0:N2}", _
            EuroDM.Euro2DM(Me.TextBox1.Text))
    End Sub
End Class
```

Listing 3.18: Die Umrechnungsklasse des Eurorechners (VbStaticMember.aspx.vb)

Wie es funktioniert

Dem Aufruf der Methode *Euro2DM* der Klasse *EuroDM* geht hier keine Initialisierung voraus. Statische Mitglieder können sofort benutzt werden:

```
EuroDM.Euro2DM(Me.TextBox1.Text)
```

Wenn Sie statische und normale Mitglieder mischen, bleiben die statischen direkt aufrufbar, während für normale Mitglieder weiterhin eine Instanziierung erforderlich ist. Solche Klassen sind dann in beiden Formen parallel nutzbar. Allerdings können Sie eine statische Methode nicht über ein instanziiertes Objekt aufrufen. Intellisense erkennt dies und bietet nur die jeweils zutreffenden Mitglieder an.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Me.Label1.Text = EuroDM.
End Sub
Class
```



Abbildung 3.12: Intellisense erkennt statische Mitglieder.

Aufbau und Zerstörung: Von Konstruktoren und Destruktoren

Bei der Instanziierung von Objekten kann ein Anfangszustand festgelegt werden. Dazu dienen so genannte Konstruktoren. Der Konstruktor einer Klasse ist eine Methode, die den Namen `New` trägt und bei Anwendung des Schlüsselwortes `New` aufgerufen wird. Die möglichen Parameter können Sie flexibel festlegen. Ebenso können Sie mehrere `New`-Methoden schreiben, die sich durch die Parameter unterscheiden. Beim Aufruf entscheidet die Runtime selbstständig, welcher Konstruktor zu verwenden ist.

Das Beispiel aus Listing 3.16 kann gut um einige Konstruktoren erweitert werden:

```
Sub New(ByVal Name As String)
    Dim aName As String()
    Dim sep As Char() = {"c", ", "c, ";"c}
    aName = Name.Split(sep, 2)
End Sub
Sub New(ByVal VorName As String, ByVal NachName As String)
    _vn = VorName
    _nn = NachName
End Sub
```

Listing 3.19: Mehrere Konstruktoren in einer Klasse (Ausschnitt aus `VbClassConstructor.aspx.vb`, der Rest des Programms entspricht Listing 3.16)

Die Übergabe der Werte erfolgt dann in der Phase der Instanziierung:

```
Dim k As Kunde2 = New Kunde2("Uwe Bünning")
```

Selbstverständlich werden Sie auch hier von Intellisense unterstützt und über die möglichen Parameter informiert:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim k As Kunde2 = New Kunde2("Uwe Bünning")
    Namen.Text = k.Name
End Sub
```



Abbildung 3.13: Anzeige mehrerer Konstruktoren mit Intellisense

Das Prinzip der Vererbung

Neu in VB.NET ist die vollständige Implementierung aller OOP-Funktionen. Dazu gehört unter anderem die Vererbung über beliebig viele Stufen. Auch wenn dies in kleineren Projekten selten benötigt wird, sollten Sie sich mit dem Prinzip auseinander setzen, denn das gesamte Framework besteht aus einer Struktur einander beerbender Klassen.



Dieser Abschnitt ist leicht vereinfacht, um den Rahmen des Buches nicht zu sprengen. Konsultieren Sie spezielle Literatur zu VB.NET, um sich mit allen Möglichkeiten vertraut zu machen. Die Darstellungen hier sind jedoch ausreichend, um alle Beispiele dieses Buches verstehen und verwenden zu können.

Inherits

Eine Vererbung finden Sie in jeder Klasse, die der Verarbeitung einer Seite dient: Die Ableitung der Seitenklasse von der Basisklasse Page. Das Schlüsselwort zur Steuerung der Vererbung ist `Inherits`. Eine Klasse kann immer nur von genau einer Basisklasse erben. Der Vorgang führt im ersten Schritt dazu, dass die erbende Klasse alle Mitglieder der vererbenden Klasse übernimmt. An dieser Stelle kommen übrigens die Zugriffsmodifizierer ins Spiel, die bislang nur eine untergeordnete Rolle hatten.

Zur Erinnerung: Mitglieder, die als `Public` gekennzeichnet werden, sind überall im Programm sichtbar. Wird dagegen `Private` verwendet, sind sie außerhalb der Klasse unsichtbar. Beim Einsatz der Vererbung ist das Verhalten gegenüber der erbenden Klasse zu definieren. Verwenden Sie `Protected`, um den Sichtbereich ausschließlich in erbende Klassen auszudehnen. Schreiben Sie größere Projekte, die sich über mehrere Anwendungen hinweg ausdehnen, kann `Friend` interessant sein. Hierbei können andere Klassen derselben Anwendung auf das Mitglied zugreifen, andere Programmteile dagegen nicht. Dieselbe Einschränkung mit Ausnahme erbender Klassen definiert die Kombination `Protected Friend`.

Die grundlegende Syntax der Vererbung zeigt folgender Code:

```
Class Punkt
    Protected x As Integer
    Protected y As Integer
End Class
```

```

Class PunktVektor
  Inherits Punkt
  Public Vektor As Double
End Class

```

Die Klasse *PunktVektor* verfügt nun über drei Eigenschaften, die als Variablen deklariert sind. Öffentlich sichtbar ist nur das Feld *Vektor*, *x* und *y* sind dagegen nur innerhalb der Klasse verwendbar, außerhalb kann kein Zugriff erfolgen. Kombiniert mit Eigenschaftsdefinitionen, wie sie im Abschnitt »Zugriff auf Mitglieder und spezielle Deklaration von Eigenschaften« ab Seite 187 gezeigt wurden, kann so der Inhalt einer Klasse gut gekapselt werden. Dies wird immer dann enorm wichtig, wenn Sie planen, Klassen an andere Entwickler weiterzugeben. Solange dies innerhalb einer Applikation und damit im Quelltext passiert, kann auf `Friend` oft verzichtet werden. Geben Sie jedoch nur die Assembly weiter, womit die Klasse Teil einer eigenständigen Anwendung wird, sind differenziertere Regelungen der Zugriffe wichtig. Sie schützen Ihre Klassen durch Fehlbehandlung damit vor unkontrolliertem Verhalten. Letztlich sichert es die Stabilität und Zuverlässigkeit Ihrer Software.

Einige Schlüsselwörter stehen in direktem Zusammenhang mit der Vererbung:

MyBase
Me

- ▶ **MyBase** dient dem Aufruf der Basisklasse, von der geerbt wurde.
- ▶ **Me** erlaubt den Zugriff auf andere Mitglieder der eigenen Klasse. Der Aufruf in dieser Form ist nur notwendig, wenn andernfalls Namenskonflikte auftreten.

Das folgende Beispiel nutzt die Vererbung, um aus einer Basis-klasse zwei simple Arbeitsklassen zu erstellen, die aufeinander aufbauen.

```

Public Class Punkt
  Protected _x As Integer
  Protected _y As Integer
End Class

Public Class PunktVektor
  Inherits Punkt
  Private _vektor As Double
  Private Function SetVektor() As Double
    Return Math.Sqrt(Math.Pow(MyBase._x, 2) _
      + Math.Pow(MyBase._y, 2))
  End Function

```

```

Sub New(ByVal x As Integer, ByVal y As Integer)
    MyBase._x = x
    MyBase._y = y
    _vektor = SetVektor()
End Sub
Public Property X() As Integer
    Get
        Return MyBase._x
    End Get
    Set(ByVal Value As Integer)
        MyBase._y = Value
        _vektor = SetVektor()
    End Set
End Property
Public Property Y() As Integer
    Get
        Return MyBase._y
    End Get
    Set(ByVal Value As Integer)
        MyBase._x = Value
        _vektor = SetVektor()
    End Set
End Property
Public ReadOnly Property GetVektor() As Double
    Get
        Return _vektor
    End Get
End Property
End Class
Public Class PunktName
    Inherits PunktVektor
    Private _name As String = String.Empty
    Sub New(ByVal x As Integer, ByVal y As Integer, ByVal Name As String)
        MyBase.New(x, y)
        _name = Name
    End Sub
    Public ReadOnly Property GetName() As String
        Get
            Return _name
        End Get
    End Property
End Class
Public Class VbClassInherits
    Inherits System.Web.UI.Page
    Protected WithEvents lVektor As Label
    Protected WithEvents lName As Label

```



```

Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) _
Handles MyBase.Load
    Dim p1 As PunktVektor = New PunktVektor(13, 99)
    Dim p2 As PunktName = New PunktName(23, 77, "Startpunkt")
    lVektor.Text = "Vektor: " & p1.GetVektor.ToString()
    lName.Text = "Vektor: " & p2.GetVektor.ToString() & _
        ", Name: " & p2.GetName
End Sub

End Class

```

Listing 3.20: Vererbungskette – Erweiterung einer Basisklasse durch Vererbung
(VbClassInherits.aspx.vb)

Die erste Klasse *Punkt* definiert lediglich zwei Felder zur Speicherung der beiden Punktwerte. Diese sind als `Protected` gekennzeichnet. Damit sind die Variablen `_x` und `_y` nur in den ererbenden Klassen sichtbar. Die zweite Klasse *PunktVektor* erbt von der Basisklasse:

Wie es funktioniert

```

Public Class PunktVektor
    Inherits Punkt

```

Beachten Sie, dass die Syntax tatsächlich verlangt, die `Inherits`-Anweisung auf eine neue Zeile zu schreiben. Wenn Sie das stört, verlängern Sie die Zeile mit dem Doppelpunkt:

Vererbung

```

Public Class PunktVektor : Inherits Punkt

```

Die Klasse *PunktVektor* fügt einen Konstruktor sowie eine Funktion zur Berechnung des Vektors hinzu. Als Vektor wird hier der Abstand zum absoluten Nullpunkt des virtuellen Koordinatensystems bezeichnet. Gespeichert wird dieser Wert in einer privaten Variablen:

```

Private _vektor As Double

```

Damit der Zugriff später erfolgt, wird eine Nur-Lese-Eigenschaft definiert. So wird das Berechnungsergebnis geschützt und der Zugriff in der Applikation offen gelegt:

Nur-Lese-Eigenschaft

```

Public ReadOnly Property GetVektor() As Double
    Get
        Return _vektor
    End Get
End Property

```

Der Zugriff auf die von der Basisklasse ererbten Felder kann direkt erfolgen. Im Fall von Namenskonflikten oder zur deutlicheren Lesbarkeit kann das Schlüsselwort `MyBase` benutzt werden:

```
MyBase._x = x
```

Eigenschaften Zwei weitere Eigenschaften erlauben den Zugriff auf die einzelnen Punktwerte. Beachten Sie hier, dass der Vektor nach jeder Änderung eines Punkts neu berechnet werden muss. Genau dies leisten die Eigenschaften im `Set`-Zweig:

```
Set(ByVal Value As Integer)
    MyBase._y = Value
    _vektor = SetVektor()
End Set
```

Einfache Variablenzugriffe würden dagegen die Synchronisation zwischen Punktwert und Vektor aufheben. Der Schutz durch Zugriffsmodifizierer verhindert dies prinzipiell.

Konstruktor Die dritte Klasse *PunktName* erweitert nun die bereits vorgestellte Klasse *PunktVektor* erneut. Zusätzlich wird dem Punkt noch ein Name hinzugefügt. Dies geschieht durch die Erweiterung des Konstruktors und die Einführung einer weiteren Eigenschaft. Der Konstruktor nutzt den bereits vorhandenen aus *PunktVektor*. Um den Aufruf durchzuleiten, erfolgt der Aufruf der Basisklasse. Dies ist aus der Sicht von *PunktName* die Klasse *PunktVektor*, nicht *Punkt*:

```
Sub New(ByVal x As Integer, ByVal y As Integer, ByVal Name As String)
    MyBase.New(x, y)
    _name = Name
End Sub
```

Der Aufruf des Konstruktors der Basisklasse ist im Beispiel zwingend. Denn mit der Instanziierung des Objekts werden alle Konstruktoren der Vererbungskette gestartet. Da der Konstruktor der Klasse *PunktVektor* Parameter erwartet, müssen diese übergeben werden. Deshalb ist der Zugriff über `MyBase` notwendig. Würde es einen parameterlosen Konstruktor geben, könnte der Aufruf entfallen. Ein solcher Konstruktor könnte folgendermaßen aussehen:

```
Sub New()
    _x = 0
    _y = 0
End Sub
```

Nun kann sogar bei der Instanziierung entschieden werden, welche Aufrufform benötigt wird. Intellisense ist auch hier wieder ein unersetzliches Hilfsmittel:

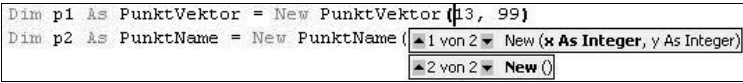


Abbildung 3.14: Bei mehreren Konstruktoren einer Vererbungskette zeigt Intellisense alle Formen an.

Neben der hier gezeigten Erweiterung der Klassen auf späteren Ebenen der Vererbung ist es auch möglich, das Verhalten einzelner Mitglieder zu verändern. Sie können Methoden so kennzeichnen, dass diese in erbenden Klassen überschrieben werden. Dabei bleiben Namen und Signaturen (Parametertypen) erhalten. Wenn Sie damit rechnen, dass ein Überschreiben einer Methode erfolgen wird, können Sie diese als überschreibbar kennzeichnen:

```
Public Overridable Sub Entwurf()
```

Nun wird es vorkommen, dass erbende Klassen die Methode überschreiben, also anders implementieren, als in der Basisklasse. Wird die Methode jedoch auch in der Basisklasse selbst benutzt, wäre ein Überschreiben fatal, denn der Entwickler der Basisklasse kann nicht wissen, wie die Überladung später erfolgt. Deshalb gibt es das Schlüsselwort `MyClass`, mit dem immer auf die originale Definition zugegriffen wird. Es entspricht damit `Me`, mit dem einen Unterschied, das `Me` der Vererbung folgt und gegebenenfalls die überschriebene Methode aufruft.

Nur informativ seien an dieser Stelle auch die übrigen Schlüsselwörter genannt, die rund um das Überschreiben von Methoden in erbenden Klassen zum Einsatz kommen können:

Schlüsselwort	Bedeutung
Overridable	Kennzeichnet eine Methode als Überschreibbar
NotOverridable	Verhindert, dass eine Methode überschrieben werden kann. Dies ist bei öffentlichen Methoden der Standardfall.
MustOverride	Zeigt an, dass die ableitende Klasse diese Methode überschreiben muss. Zugleich wird damit festgelegt, dass die Methode nicht implementiert ist.

Tabelle 3.13: Schlüsselwörter zur Steuerung der Vererbung auf einen Blick

Schlüsselwort	Bedeutung
Overrides	Zeigt an, dass diese Methode eine andere überschreibt. Der Einsatz erfolgt in der erbenden Klasse.
Inherits	Zeigt die Vererbung einer Klasse an
MustInherit	Zeigt an, dass die Klasse vererbt werden muss, sie wird damit zwingend zur Basisklasse
NotInheritable	Verhindert, dass von der Klasse geerbt werden kann
MyBase	Zugriff auf Mitglieder der Basisklasse
MyClass	Zugriff auf Mitglieder der Klasse selbst. Überschriebene Mitglieder werden nicht verwendet, stattdessen immer das Original.
Me	Zugriff auf Mitglieder der Klasse selbst. Überschriebene Mitglieder werden verwendet.

Tabelle 3.13: Schlüsselwörter zur Steuerung der Vererbung auf einen Blick (Forts.)

Aus diesen Möglichkeiten geht hervor, dass es offensichtlich möglich ist, praktisch funktionslose Klassen zu schreiben, die erst später durch Vererbung mit Arbeitsfunktionen gefüllt werden. Damit ist es an der Zeit, sich mit abstrakten Klassen zu beschäftigen.

Gerüste für Klassen: Abstrakte Klassen und Schnittstellen

Dieser Abschnitt ist nur informativ. Eine zwingende Notwendigkeit, die hier gezeigten Techniken in der Praxis der ASP.NET-Programmierung zu verwenden, besteht nur sehr selten. Generell ist es möglich, funktionslose Klassen zu schreiben. Durch das Schlüsselwort `MustOverride` wird angezeigt, dass die betroffene Methode in der erbenden Klasse implementiert werden muss. Mit `MustInherit` wird gleich eine ganze Klasse als implementierungspflichtig deklariert. Der Sinn dahinter besteht in der Definition der Signatur, also der Parameterfolge. Architekten von Klassenhierarchien können damit ein einheitliches Erscheinungsbild durchsetzen. Die eigentliche Implementierung wird jedoch dem Anwendungsentwickler überlassen, sodass hier eine größtmögliche Freiheit herrscht.

Klassen können solche überschreibungspflichtigen Methoden mit fest implementierten mischen. Diese Freiheit haben Schnittstellen nicht. Diese sind ähnlich wie Klassen aufgebaut, enthalten jedoch generell keinen ablauffähigen Code. Es gibt einen wesentlichen Unterschied zu Klassen: Während immer nur von einer Klasse geerbt werden kann, ist das Erben von mehreren Schnittstellen möglich.

Damit sind Schnittstellen als Bauanleitung für Klassen flexibler. Allerdings ist dann die Implementierung aller Mitglieder zwingend erforderlich. Dadurch ist der Umsetzungsaufwand höher. Schnittstellen lohnen sich nur, wenn komplexe Klassenhierarchien von größeren Teams verwendet oder entworfen werden und eine einheitliche Benennung bestimmter Methoden erforderlich ist, die »perfekte« Kommunikation aber nicht immer gewährleistet ist.

Überschreibungs- bzw. implementierungspflichtige Mitglieder werden im OOP-Sprachgebrauch als »abstrakt« bezeichnet. Wenn sie dann implementiert wurden, sind sie »konkret«.



Im .NET-Framework werden Schnittstellen vielfältig verwendet. Dabei geht es vor allem um die Absicherung der Kompatibilität selbst implementierter Klassen mit vorhandenen, was zwingend ist, um flexibel alle eingebauten Leistungen zu nutzen.

Die Deklaration einer Schnittstelle erfolgt mit dem Schlüsselwort `Interface`:

Interface

```
Interface Name
    ' Methoden/Eigenschaften
End Interface
```

Da konkrete Variablen nicht erlaubt sind, müssen Sie Eigenschaften verwenden. »Einleitende« Schlüsselwörter sind `Property`, `Function`, `Sub` und `Event`.

Wird eine Schnittstelle verwendet, findet das Schlüsselwort `Implements` Verwendung. Dies wird auch mit eingebauten .NET-Schnittstellen benutzt. Sie werden es an einigen Stellen im Buch wiederfinden:

```
Class Name
    Implements Schnittstelle, Schnittstelle2
```

Zusätzlich muss das Schlüsselwort `Implements` hinter jeden Methodenkopf geschrieben werden, welcher der Implementierung konkret dient. Dies hat den Vorteil, dass der Name geändert werden kann⁵.

Implements

⁵ C# verhält sich hier übrigens anders. Bei der Implementierung kann dort der Name nicht geändert werden und aus rein theoretischer Sicht ist dies auch keine gute Idee, denn es erschwert die Lesbarkeit des Codes unter Umständen enorm.

Implements folgt dem Kopf der Methode direkt, nicht auf der folgenden Zeile. Erscheint Ihnen das aus Sicht der Lesbarkeit unglücklich, denken Sie an das Zeilenumbruchzeichen »_«.

Prinzip der Namensräume in Visual Studio .NET und in VB.NET-Projekten

Die Klassen des Frameworks sind in Namensräumen organisiert. Diese helfen dabei, gleichnamige Klassen zu unterscheiden. Wenn Sie kleinere Projekte schreiben, ist die Organisation in einer vergleichbaren Namensraumhierarchie nicht sinnvoll. Sie sollten allerdings wissen, dass Visual Studio .NET ihre Seiten in einen automatisch erstellten Namensraum verpackt. Dieser folgt standardmäßig dem Namen des Projekts.

Namensraum-hierarchie

In diesem Buch wird für die mitgelieferten Programme eine Namensraumhierarchie verwendet, weil es sich nicht um ein großes Programm, sondern um Hunderte einzelner Klassen handelt.

Namensräume werden folgendermaßen deklariert:

```
Namespace Oberster.Naechster
    ' Viele Klassen ...
End Namespace
```

Visual Studio .NET

Dabei sind die Punkte zur Trennung mehr oder weniger willkürlich, es kommt darauf an, wie sie die Klassen organisieren. In Visual Studio .NET gehen Sie etwas anders vor. Im Projekt wird ein Standardnamensraum eingerichtet, dem dann alle weiteren Deklarationen untergeordnet sind. Wenn Sie nun den Namensraum Oberster deklarieren und dann im Code erneut das Schlüsselwort `Namespace` einsetzen, entsteht sofort eine weitere Stufe der Hierarchie⁶.

Normalerweise sollten Sie also mit Visual Studio .NET nicht in die Verlegenheit kommen, weitere Namensräume anzulegen. Wenn Sie die Namensräume eines Projekts überwachen möchten, öffnen Sie die Klassenansicht mit `Strg-Umschalt-C`. Die Namensräume werden durch geschweifte Klammern angedeutet (siehe Abbildung 3.16).

⁶ Dies ist insofern bemerkenswert, als dass sich C# auch hier wieder anders verhält.

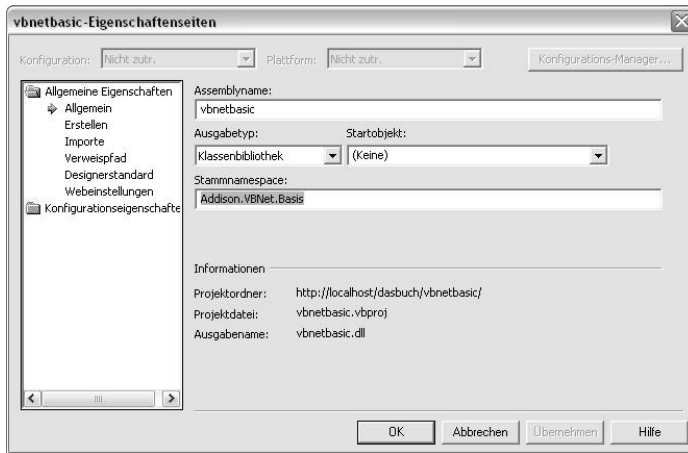


Abbildung 3.15: Festlegen des Standardnamensraumes für ein Projekt

Die Aufteilung der Programme auf Dateien hat auf die Verwendung durch den Compiler und die durch Namensräume und Klassen erzeugten Struktur keinen Einfluss. Wählen Sie eine solche Dateistruktur, dass Sie sich gut darin zurecht finden.

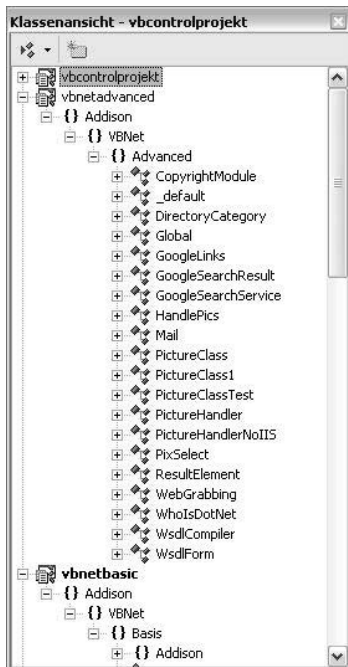


Abbildung 3.16: Ansicht eigener Namensräume

3.4.9 Strukturen und Aufzählungen

In den vorangegangenen Abschnitten wurden bereits eigene Klassen deklariert. Objekte, die daraus abgeleitet werden, haben den Typ der Klasse. Es gibt zwei Spezialfälle, die zwar durch Klassen abgedeckt werden, deren Verwendung aber durch besondere Sprachkonstrukte erleichtert wird: Strukturen und Aufzählungen.

Strukturen zur Definition eigener Datentypen

Man kann Klassen konstruieren, die keine umfassende Funktionalität aufweisen, sondern lediglich der Deklaration eigener Typen dienen. Tatsächlich ist dies nicht nur möglich, sondern wird durch ein weiteres Schlüsselwort direkt unterstützt: `Structure`.

Referenztyp Trotz fast identischer Syntax und vergleichbarem Einsatz gibt es einen wesentlichen Unterschied zwischen Klassen und Strukturen. Während Klassen immer Referenztypen sind und bei der Übergabe davon abgeleiteter Objekte nur Zeiger darauf weitergereicht werden, sind Strukturen Werttypen. Die Objekte einer Struktur werden direkt übergeben. Damit eignen sich Strukturen eher für kleinere Datenmengen. Die Hauptanwendung ist deshalb die Definition eigener Typen.

Einfacher als Klassen Auch im Hinblick auf die Verwendung sind Strukturen einfacher als Klassen. So können sie nicht vererbt werden. Strukturen verbleiben, wie andere Werttypen auch, bis zum Ende des Programms im Speicher, ein Destruktor ist deshalb unnötig und wird nie aufgerufen. Der Einsatz von Ereignissen ist nicht möglich. Mitglieder können nicht als `Protected` gekennzeichnet werden. Die Anwendung des Schlüsselwortes `New` zur Initialisierung ist unnötig, es wird immer sofort eine Instanz erzeugt.

Das folgende Beispiel zeigt die Deklaration und Verwendung einer sehr einfachen Struktur. Sie fasst zwei Punkte vom Typ `Integer` und eine Zeichenkette zusammen:

```
Public Class VbStructure
    Inherits System.Web.UI.Page

    Public ausgabe As Label

    Public Structure MyPoint
        Public x As Integer
        Public y As Integer
```



```

        Public s As String
    End Structure

    Private Sub Page_Load(ByVal sender As System.Object,   

        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim p1 As MyPoint
        p1.x = 17
        p1.y = 23
        p1.s = "Erster Punkt"
        ausgabe.Text = p1.s & " hat folgende Werte:<br />"
        ausgabe.Text += "X = " & p1.x.ToString() & "<br />"
        ausgabe.Text += "Y = " & p1.y.ToString() & "<br />"
    End Sub

End Class

```

Listing 3.21: Erzeugen und Nutzen einer Struktur (VbStructure.aspx.vb)

Die Ausgabe erfolgt in ein Label-Steuerelement mit dem Namen *ausgabe*.



Abbildung 3.17: Nutzung eines eigenen Datentyps (VbStructure.aspx)

Im Gegensatz zum fehlenden Destruktor ist die Anwendung eines Konstruktors bei einer Struktur möglich, vor allem um einen definierten Anfangszustand der Mitgliedsvariablen zu erreichen.

Aufzählungen

In Programmen kommt es oft vor, dass Variablen nur eine fest umrissene Anzahl Werte aufnehmen. Denken Sie beispielsweise an Wochentage oder Monatsnamen. In beiden Fällen wäre der Datentyp *String* zwar verwendbar, aber nicht optimal. Das Schlüsselwort, mit dem Aufzählungen erzeugt werden, heißt *Enum* (vom englischen Begriff *enumeration*):

Enum

```

Enum weekday
    Montag,
    Dienstag,

```

```

        Mittwoch,
        Donnerstag,
        Freitag,
        Samstag,
        Sonntag
    End Enum

```

Sie verfügen damit noch nicht über eine Variable, sondern nur über eine sehr individuelle Typdefinition. Von `weekday` abgeleitete Variablen dürfen einen der angegebenen Werte enthalten. Definieren Sie eine Variable wie üblich, um sie mit einem der Aufzählungswerte zu belegen:

```
Dim tag As weekday
```

Der Variablen `tag` können Sie nun einen der Werte zuweisen, indem Sie direkt auf die Definition zugreifen:

```
tag = eWeekday.Montag
```

Das folgende Beispiel *EnumWeekday.aspx* zeigt die Anwendung. Bei der Ausgabe wird der erkannte Wert als Zeichenkette ausgegeben. Intern werden jedoch Zahlen zur Indizierung verwendet – daher der Name »Aufzählung«. Die Zählung beginnt mit 0 und setzt jeweils um 1 erhöht fort. Sie können aber andere Werte erzwingen:

```

<script language="vb" runat="server">
Enum weekday
    Montag,
    Dienstag,
    Mittwoch,
    Donnerstag,
    Freitag,
    Samstag,
    Sonntag
End Enum
</script>
<%
Dim tag As weekday
tag = weekday.Montag
Response.Write ("Tag Nr. " & tag & "<br/>")
If tag = weekday.Montag Then
    Response.Write("Montag")
End If
%>

```

Listing 3.22: Aufzählung mit vorgegebenen Indexwerten (VbEnumWeekdayIndex.aspx)



Abbildung 3.18: Ausgabe der Aufzählungswerte

In dieser Form sind Aufzählungen vor allem hilfreich, zusammengehörende Konstanten in lesbarer Form zu halten. Außerdem sollten Sie einen eigenen Datentyp immer so wählen, dass er den für die Erfüllung der Problemstellung kleinstmöglichen Wertebereich umfasst. Aufzählungen sind ein probates Mittel dafür.

Anwendungstipps

Der Zugriff ist wie vorher beschrieben über die Namen möglich. Außerdem gibt es eine spezielle Syntax für die Nutzung der Indizes. Statt eines konstanten Werts sind auch Variablen zulässig. Zusätzlich kann durch die Formatierung festgelegt werden, was zurückgegeben wird:

Zugriff auf Aufzählungen

```
<%  
Dim i As Integer  
For i = 0 To 6  
    Response.Write <img alt="arrow icon" data-bbox="308 458 328 468"/>  
        (System.Enum.Format(GetType(weekday), i, "g") & "<br/>")  
Next  
%>
```

Listing 3.23: Zugriff auf eine Aufzählung mit variablem Index (VbEnumWeekday-For.aspx, die Definition der Aufzählung entspricht der in Listing 3.22)

Die möglichen Formatierungen für die Aufzählungsmitglieder können Sie der folgenden Tabelle entnehmen:

Formatierungen

Formatsymbol	Bedeutung
g oder G	Der Name der Konstanten als Zeichenkette
x oder X	Der Wert des Elements als Hexadezimalzahl
d oder D	Der Wert des Elements als Dezimalzahl

Tabelle 3.14: Formatierungen mit `Enum.Format` für den Aufzählungs-Datentypen

Aus der Darstellung folgt auch, dass die durch das Schlüsselwort `Enum` bereitgestellt Funktion keine Spracherweiterung von VB.NET ist, sondern auf Klassen des Frameworks basiert. Starten Sie für weitere Recherchen bei `System.Enum`.

Aufzählungen - for-Schleife

```
Sonntag  
Montag  
Dienstag  
Mittwoch  
Donnerstag  
Freitag  
Samstag
```

Abbildung 3.19: Ausgabe aller Elemente einer Aufzählung

Zugriffs- modifizierer

Abschließend noch ein Hinweis zur Sichtbarkeit. Aufzählungen sind ähnlich wie Klassen zu verstehen. Es sind allerdings Wertetypen, entsprechen also mehr einer Struktur. Die Definition kann ebenso wie diese innerhalb von Namensräumen, Modulen und auf Dateiebene erfolgen, sowie innerhalb von Klassen. In Prozeduren oder Funktionen ist dagegen nur die Verwendung, nicht die Definition erlaubt. Zulässige Zugriffsmodifizierer sind `Public`, `Private` und `Friend`.



Im Framework werden Aufzählungen sehr häufig verwendet. Diese sind natürlich bereits fertig definiert und müssen nur eingesetzt werden.

3.4.10 Ausnahmebehandlung

Try...Catch... Finally

Die Anweisung, die hierzu verwendet wird, besteht aus mindestens zwei Schlüsselwörtern: *Try* und *Catch*. Der erste Teil wird von *Try...End Try* umschlossen. Hier wird die Ausführung von beliebigem Code versucht. Tritt ein Laufzeitfehler auf, wird – in der .NET-Sprechweise – eine Ausnahme »geworfen«. Ist keine Behandlung dafür definiert, hilft sich die CLR selbst und zeigt den Fehler an. Sie können die Ausnahme aber auch »fangen« (engl. *to catch*), was mit dem Schlüsselwort *Catch* erfolgt. Falls Sie Code schreiben, der immer ausgeführt werden soll, egal ob es einen Laufzeitfehler gab oder nicht, ist ein weiterer durch *Finally* eingeleiteter Block notwendig.

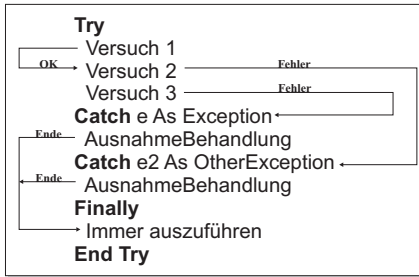


Abbildung 3.20: Ablauf eines Programms mit Try-Catch-Finally-Blöcken

Das folgende Beispiel zeigt, wie die Anwendung in der Praxis erfolgt. Abgefangen wird eine Ausnahme, die bei Berechnungen häufiger auftritt: Division durch Null.

```

<html lang="de">
  <head>
    <title>Try Catch Finally</title>
  </head>
  <body>
    <h1>Try Catch Finally</h1>
    <p id="ausgabe" runat="server"/>
    <p id="fehler" runat="server"/>
  </body>
</html>

```

Listing 3.24: Abfangen einer Ausnahme (VbTryCatch.aspx)

```

Class Ausnahmen
  Public Function rechne(ByVal x As Integer,   

    ByVal y As Integer,   

    ByVal opcode As Char) As String
    Dim result As Integer
    Try
      Select Case opcode
        Case "*"c
          result = (x * y)
          Exit Function
        Case "/"c
          result = (x \ y)
          Exit Function
        Case "-"c
          result = (x - y)
          Exit Function
        Case Else
          result = (x + y)
          Exit Function
      End Select
    End Try
  End Function
End Class

```

```

        Catch e As Exception
            Return ("<b>Fehler:</b> " + e.ToString())
        End Try
        Return result.ToString()
    End Function
End Class

Public Class VbTryCatch
    Inherits System.Web.UI.Page
    Protected WithEvents ausgabe As Label

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim a As Integer = 16
        Dim b As Integer = 0
        Dim r As String
        Dim rechner As Ausnahmen = New Ausnahmen()
        r = rechner.rechne(a, b, "/"c)
        ausgabe.Text = r
    End Sub
End Class

```

Listing 3.25: Code-Datei zur Ausnahmebehandlung mit einem Try-Catch-Block (VbTryCatch.aspx.vb)

Wie es funktioniert

In diesem Beispiel wird eine zusätzliche Klasse definiert, die neben der Ausführung der Aufgaben auch Fehler auswertet. Die Berechnungen finden in einem *Try*-Block statt. Läuft dieser Block korrekt ab, werden nachfolgende *Catch*-Blöcke übersprungen. Damit wird die letzte *Return*-Anweisung erreicht:

```
Return result.ToString()
```

Tritt dagegen eine Ausnahme auf, wird der nächste dazu passende *Catch*-Block ausgeführt. Verwendet wird hier der globale Typ einer Ausnahme, *Exception*. Diese Art trifft für alle Ausnahmen zu. Abgeleitet werden diese Ausnahmen in dem in ASP.NET benötigten Umfang von der Klasse *System.SystemException*. Die eigentlichen Klassendefinitionen sind weit im Framework verstreut, da fast überall Laufzeitfehler auftreten können. Der häufig auftretende Fehler »Division durch Null« wird in *System.ArithmeticException* definiert.

Try Catch Finally

Fehler: System.DivideByZeroException: Attempted to divide by zero. at ASP.Ausnahmen.rechne(Int32 x, Int32 y, Char opcode) in C:\inetpub\wwwroot\dotnet\trycatch.aspx:line 28

Abbildung 3.21: Ausgabe eines Laufzeitfehlers mit eigenem Code

An einigen Stellen in Ihrem Programm werden Sie vielleicht Fehlerbedingungen selbst feststellen. Statt nun eine erneute Fehlerbehandlungsmethode zu schreiben, nutzen Sie eine aus einem vorhandenen *Try-Catch*-Block. Dazu »werfen« (engl. *to throw*) Sie selbst eine Ausnahme. Das Schlüsselwort heißt nahe liegenderweise *Throw*. Der Umgang damit ist nicht ganz einfach, weil Sie erst eine Klasse definieren müssen, die die Fehlerbehandlung übernimmt, beispielsweise eine Textausgabe. Dann ist es notwendig, eine Instanz dieser Klasse beim Auftreten der Fehlerbedingungen zu »werfen«:

Throw

```
If zahl = 0 Throw New AusnahmeKlasse("Fehlermeldung")
```

Wie dies am konkreten Beispiel aussieht, wird an vielen Stellen im Buch gezeigt.

4 Die Basisklassen des Frameworks

Die Programmiersprache hat im .NET-Framework nur eine geringe Bedeutung, denn alle für die praktische Programmierung benötigten Funktionen stammen aus dem Framework. Dies hat den großen Vorteil, dass bei einer Anwendung wie ASP.NET der gesamte Leistungsumfang von .NET zur Verfügung steht.

4.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

4.1.1 Über dieses Kapitel

Bei der bisherigen Vorstellung haben Ihnen im Vergleich zu VBScript oder auch Visual Basic vielleicht die vielen Funktionen zur Verarbeitung von Arrays oder Zeichenketten gefehlt. Tatsächlich sind derartige Funktionen in VB.NET kaum enthalten. Sie müssen dazu auf Klassen des .NET-Frameworks zurückgreifen. Solche elementaren Klassen sind im Namensraum `System` definiert, den ASP.NET automatisch einbindet. Wenn Sie mit hinterlegtem Code (Code Behind) arbeiten, VB.NET also in einer eigenen Datei ablegen, ist dieser Namensraum explizit anzugeben. Informationen dazu finden Sie in Abschnitt 1.5.2, »Code Behind« ab Seite 79. Dies ist eine speziell für ASP.NET entwickelte Technik.



Weiterhin werden in diesem Kapitel bereits so genannte Steuerelemente verwendet. Lassen Sie sich davon nicht irritieren. Wenn Sie neugierig sind: Diese Tags werden ausführlich in den Kapiteln 7 und 8 vorgestellt.

Standardklassen aus dem Namensraum System

Wichtige Klassen

Hier werden die Klassen aus `System` vorgestellt, die die in vielen anderen Sprachen vorhandenen Funktionen ersetzen:

► `Array`

Diese Klasse enthält Eigenschaften und Methoden zur Verwendung mit Arrays. Eine Beschreibung finden Sie im Abschnitt 4.2, »Verarbeitung von Arrays« ab Seite 217.

► `CharEnumerator`

Mit dieser Klasse wird auf einzelne Zeichen einer Zeichenkette verwiesen.

► `String`

Hier sind alle Eigenschaften und Methoden zur Verarbeitung von Zeichenketten zu finden.

► `Hashtable`, `ArrayList`, `SortedList`, `NameValueCollection` u. a.

Diese Klassen sind spezialisierte Formen von Kollektionen und dienen der Speicherung von Werten oder Schlüssel-/Wertepaaren, kombiniert mit spezifischen Ein- und Ausgabemethoden. Abschnitt 4.3, »Aufzählungen und Kollektionen« ab Seite 222 befasst sich mit diesen Klassen.

► `Convert`

Durch das typstrenge Konzept müssen oft Datentypen umgewandelt werden, wozu `Convert` eingesetzt wird.

► `Enum`

Aufzählungen lassen sich mit dieser Klasse erstellen und nutzen.

► `Math`

Mathematische Berechnungen basieren auf Eigenschaften und Methoden dieser Klasse.

► `Random`

Diese Klasse dient der Erzeugung von Zufallszahlen.

Behandlung von Datentypen

Datentypen haben in einem typstrengen System wie .NET eine große Bedeutung. Konvertierungen und Formatierungen sind mit großer Sorgfalt durchzuführen. Für den professionellen Softwareentwickler ist die Kenntnis der wichtigsten Umwandlungsoptionen deshalb wichtig. Viele Methoden und Eigenschaften dienen dem Umgang mit Datentypen. Mehr Informationen dazu finden Sie in Abschnitt 4.8, »Eigenschaften und Methoden der Datentypen« ab Seite 282.

VB.NET erlaubt einen etwas laxeren Umgang mit Typen und Deklarationen. Sie können dennoch eine strenge Typbehandlung erzwingen, indem die Anweisung `Option Strict On` (keine implizite Typkonvertierung) und `Option Explicit On` (Deklaration explizit erforderlich) eingesetzt werden. Fast alle Beispiele im Buch – mit Ausnahme der Trivialcodes am Anfang – sind so aufgebaut.

Wie typstrenge ist VB.NET?

Datums- und Zeitoperationen, Lokalisierung

Datums- und Zeitoperationen sind nie trivial. Abschnitt 4.9, »Datum und Zeit« ab Seite 284 zeigt, wie Sie mit Daten rechnen sowie Datums- und Zeitwerte landesspezifisch formatieren können.

Zugriff auf das Dateisystem

Ebenso elementar sind Dateisystemzugriffe. Mit der Bedeutung von XML wächst auch die Anzahl der Situationen, in denen Textdateien gelesen oder geschrieben werden müssen. Ausführlich wird dieses Thema im Abschnitt 4.11, »Zugriff auf das Dateisystem« ab Seite 329 behandelt.

4.1.2 Wegweiser in die Referenz

Jede der vorgestellten Klassen verfügt über viele Eigenschaften und Methoden. Durch die Vererbungshierarchie im Framework werden einige davon aus höheren Klassen oder Schnittstellen abgeleitet. Für die Suche in der Online-Referenz ist die Kenntnis der Vererbungen hilfreich.

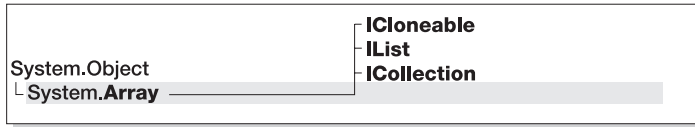


Abbildung 4.1: Ableitung der Klasse `System.Array` und deren Schnittstellen

Die Implementierung der Schnittstelle `IList` erfolgt nur teilweise, einige Methoden werden von `Array` nicht unterstützt.

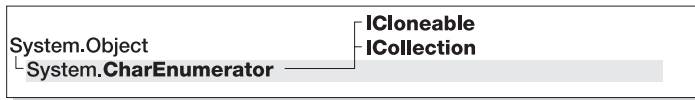


Abbildung 4.2: Ableitung der Klasse `System.CharEnumerator` und deren Schnittstellen

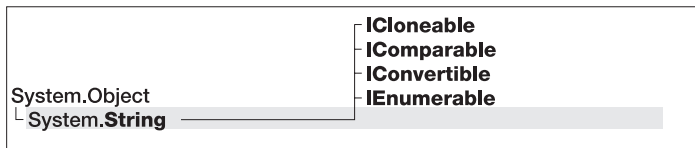


Abbildung 4.3: Ableitung der Klasse `System.String` und deren Schnittstellen

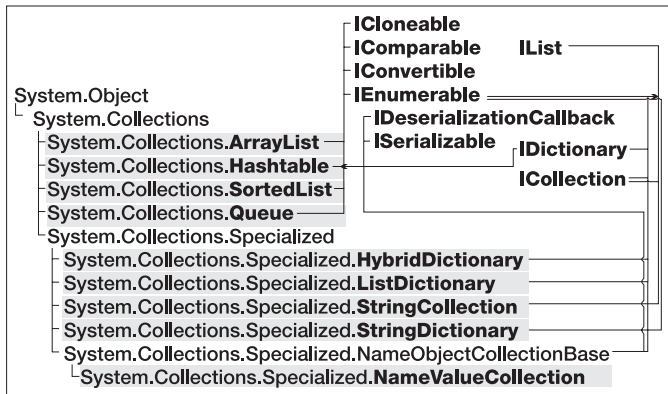


Abbildung 4.4: Ableitung der Klassen zur Verarbeitung von Kollektionen und die implementierten Schnittstellen

Die Liste der Klassen unterhalb `System.Collections` bzw. `System.Collections.Specialized` ist in der Abbildung nicht vollständig. Aufgeführt sind nur die wichtigsten in diesem Buch behandelten Klassen.

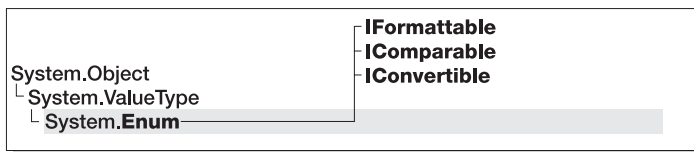


Abbildung 4.5: Ableitung der Klasse `System.Enum`

Die Klassen `Array`, `ArrayList`, `Hashtable` und `Enum` sind selbst Basis-klassen einer Vielzahl anderer Klassen des Frameworks, die wie-derum mit Aufzählungen arbeiten.

Die Klassen `Convert`, `Math` und `Random` befinden sich im Namens-raum `System`.

4.2 Verarbeitung von Arrays

Arrays kommt generell eine größere Bedeutung zu, denn im Ge-gensatz zu allen anderen Listenformen sind sie am universellsten und vergleichsweise gut mit Eigenschaften und Methoden aus-gestattet. Wenn Sie ein »Aufzählungsproblem« haben, versuchen Sie es zuerst mit Arrays zu lösen, danach erst mit den Wörter-büchern.

4.2.1 Verarbeiten von Arrays

Arrays wurden bereits aus Sicht der Programmiersprache VB.NET beschrieben. Eine ganze Palette von Eigenschaften und Methoden erweitern die Möglichkeiten erheblich. Alle Mitglieder stammen aus der Klasse `System.Array`. In ASP.NET wird `System` au-tomatisch eingebunden; dieser Namensraum enthält die Klasse. In externen VB.NET-Dateien deklarieren Sie ihn zur Nutzung von Zeichenketten folgendermaßen:

```
Imports System
```

Auch hier sei es nochmals angemerkt: Visual Studio .NET erledigt das automatisch für Sie. Nur wenn Sie mit anderen Entwicklungsumgebun-gen arbeiten, ist die explizite Deklaration in VB.NET erforderlich.



Typische Eigenschaften und Methoden

Größe feststellen Die Feststellung der aktuellen Größe eines Arrays wird mit der Eigenschaft `Length` festgestellt. Hat das Arrays mehr als eine Dimension, werden alle Elemente aller Dimensionen gezählt. Mit `Rank` kann die Anzahl der Dimensionen ermittelt werden. Die Methode `GetLength` zählt dagegen die Elemente einer bestimmten Dimension.

In manchen Situationen ist es besser, mit Aufzählungstypen statt Arrays zu arbeiten. Dann können Sie die Schnittstelle `IEnumerator` des Arrays mit `GetEnumerator` verwenden. Mehr Informationen darüber finden Sie in Abschnitt 4.3, »Aufzählungen und Kollektionen« ab Seite 222. Aufzählungen verfügen über Methoden, mit denen man sich programmtechnisch durch die Elemente bewegen kann, wie beispielsweise `MoveNext`. Solche Methoden kennt die Klasse `Array` nicht.

Elemente kopieren Ein komplettes Array können Sie an einer bestimmten Stelle in einem anderen einfügen, dazu wird `CopyTo` verwendet. Das funktioniert aber nur mit eindimensionalen Arrays. Ein Teil wird dagegen mit `Copy` eingefügt.

Der lesende Zugriff auf ein spezifisches Element erfolgt mit `GetValue`. Als Parameter werden numerische Indizes für jede Dimension erwartet, die das Array besitzt. Schreibend wird mit `SetValue` zugegriffen.

Ein neues Array mit einem bestimmten Datentyp der Elemente wird mit `CreateInstance` erzeugt:

```
Dim namen As Array = Array.CreateInstance(GetType(String), 5)
```

Dieses Array kann fünf Elemente vom Typ `String` aufnehmen. In VB.NET kann man dafür auch Folgendes schreiben:

```
Dim namen As String(5)
```

Elemente suchen Um ein bestimmtes Element in einem Array zu suchen, wird `BinarySearch` verwendet. Als Parameter wird das Array und das zu suchende Objekt erwartet, zurückgegeben wird der Index. Beachten Sie, dass ein konstanter Wert mit dem Boxing-Verfahren in ein Objekt verwandelt werden kann:

```
<script runat="server" language="vb">
Private Sub Page_Load()
    Dim namen() As String = {"Schmidt", "Krause", "↵"
```

```

        "Mueller", "Meier"}
    Dim name As Object = "Krause"
    Dim ni As Integer = Array.BinarySearch(namen,name)
    ausgabe.InnerHtml += "Gefundener Index: " + ni.ToString()
End Sub
</script>
<html>
    <head>
        <title>Arrays</title>
    </head>
    <body>
        <h1>Array-Methoden</h1>
        <p id="ausgabe" runat="server"/>
    </body>
</html>

```

Listing 4.1: Durchsuchen eines Arrays (ArrayMethods1.aspx)

Die Variable *ni* enthält nach der Ausführung den Wert »Schmidt«. Generell gilt, dass die Suche nur funktioniert, wenn das Suchwort mit dem Arrayinhalt völlig übereinstimmt.

Soll ein Array in umgekehrter Reihenfolge gelesen werden, bietet sich die Methode `Reverse` an. Sortiert werden kann dagegen mit `Sort`. Beide Methoden sind statisch. `Sort` arbeitet nur mit eindimensionalen Arrays. Allerdings gibt es mehrere Überladungen, sodass auch komplexere Sortiervorgänge abgewickelt werden können. So kann ein Array auf der Basis eines anderen sortiert werden. Die Methode `Sort` ändert immer das Original. Das folgende Beispiel zeigt die Verwendung; die HTML-Ausgabe entspricht dem letzten Listing und wird nicht wiederholt abgedruckt:

Sortieren

```

Private Sub Page_Load()
    Dim namen() As String = {"Schmidt", "Krause", "↵",
                             "Mueller", "Meier"}
    Dim daten() As String = {"1.12.1973", "13.4.1980", "↵",
                             "23.8.1960", "14.9.1977"}
    Array.Sort(namen, daten, New CaseInsensitiveComparer())
    Dim i As Integer
    For i = 0 To namen.Length - 1
        ausgabe.InnerHtml += "<br/>" + namen(i) + ", " + daten(i)
    Next
End Sub

```

Listing 4.2: Sortieren eines Arrays und davon abhängig eines zweiten (Ausschnitt aus ArrayMethods2.aspx)

Die Ausgabe zeigt, dass das alphabetisch aufsteigend sortierte erste Array *namen* das zweite mit den Geburtsdaten ebenfalls beeinflusst hat.

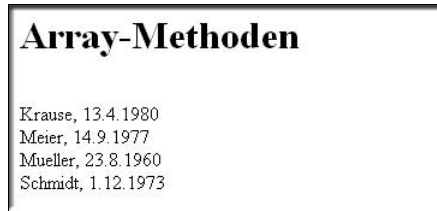


Abbildung 4.6: Ausgabe eines sortierten Arrays

Definition eigener Suchalgorithmen

Sortieren Eine einfache aufsteigende Sortierung wird zwar oft benötigt, mehr Sortieroptionen wären aber wünschenswert. Das .NET-Framework bietet dafür lediglich eine Implementierungsschnittstelle; Sie müssen andere Sortierkriterien selbst implementieren. Die Schnittstelle heißt *IComparer*. Sie wird implementiert, indem auf Basis der Schnittstelle eine Klasse entwickelt wird, die eine Methode *Compare* (Methode) enthält. Diese Methode erwartet zwei Objekte. Die Sortierung basiert auf dem Rückgabewert; bei 0 sind die Objekte gleich, bei einer negativen Ganzzahl ist das erste Objekt »kleiner«, bei einer positiven Zahl »größer«. Was Sie als kleiner oder größer betrachten, ist Ihnen überlassen. Das folgende Beispiel sortiert ein Array nach der Länge der Elemente, wobei angenommen wird, dass es sich um Zeichenketten handelt:

```
<script runat="server" language="vb">
Public Class MyComparer
Implements System.Collections.IComparer
    Public Function MyCompare(ByVal a As Object, ByVal b As Object) As Integer _
        Implements System.Collections.IComparer.Compare
        Dim sa As String = a.ToString()
        Dim sb As String = b.ToString()
        If sa.Length < sb.Length Then
            Return -1
        End If
        If sa.Length > sb.Length Then
            Return 1
        End If
        Return 0
    End Function
End Class
```



```

End Function
End Class
Private Sub Page_Load()
    Dim namen() As String = {"Schmidtchen", "Krause", "␣
                                "Mueller", "Meier"}
    Dim daten() As String = {"1.12.1973", "13.4.1980", "␣
                                "23.8.1960", "14.9.1977"}
    Array.Sort(namen, daten, New MyComparer())
    Dim i As Integer
    For i = 0 To namen.Length - 1 Step i + 1
        ausgabe.InnerHtml += "<br/> " + namen(i) + ", " +
    daten(i)
    Next
End Sub
</script>

```

Listing 4.3: Arrays mit eigenem Sortierkriterium (ArrayMethods3.aspx)

Das Geheimnis steckt hier in der Klasse `MyComparer`, die die Schnittstelle `IComparer` implementiert:

Wie es funktioniert

```
Public Class MyComparer Implements System.Collections.IComparer
```

Dort wird eine Methode definiert, die zwei Objekte vergleichen kann. Die Schnittstelle erwartet die Implementierung der Methode `Compare`, was hier als *MyCompare* umgesetzt wird:

```

Public Function MyCompare(ByVal a As Object, ␣
ByVal b As Object) As Integer _
    Implements System.Collections.IComparer.Compare

```

Für das konkrete Beispiel wird angenommen, dass es sich um Zeichenketten handelt, deshalb erfolgt – nicht fehlersicher und sehr direkt – die Umwandlung mit `ToString`:

```
Dim sa As String = a.ToString()
```

Dann folgt der Vergleich der Längen, wobei zuerst festgestellt wird, ob die erste Zeichenkette kleiner als die zweite ist:

```

If sa.Length < sb.Length Then
    Return -1

```

Auch der umgekehrter Fall wird analysiert:

```

If sa.Length > sb.Length Then
    Return 1

```

Trifft beides nicht zu, müssen die Zeichenketten gleich sein:

Return 0

Damit sind alle Pfade der Methode abgedeckt, es gibt keine unerwarteten Zustände mehr. Die Anwendung erfolgt durch Übergabe einer Instanz der Klasse an die `Sort`-Methode:

```
Array.Sort(namen, daten, new MyComparer())
```

Dieses Verfahren können Sie auch verwenden, wenn nur ein `Array` oder eine der anderen Implementierungen von `Sort` verwendet wird.

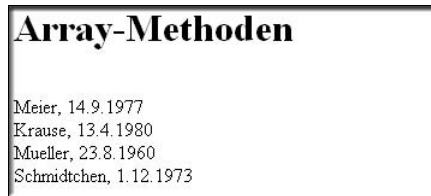


Abbildung 4.7: Sortierung eines Arrays mit eigener Sortiermethode



Wenn Sie tiefer in die Materie einsteigen möchten, müssen Sie die Technik der Schnittstellen und Implementierungen verstehen. Das gezeigte Beispiel funktioniert, ist aber auf das absolut nötige Minimum reduziert worden. Die Möglichkeiten, die das .NET-Framework hier bietet, sind enorm.

4.3 Aufzählungen und Kollektionen

Der Umgang mit Arrays wurde bereits in der Spracheinführung und im letzten Abschnitt gezeigt. Die Speicherung von zusammengehörenden Werten kann jedoch mit mehreren Verfahren erfolgen. Für alle denkbaren Variationen bietet das .NET-Framework die nötige Unterstützung. Dafür steht sogar ein eigener Namensraum mit dem Namen `System.Collections` zur Verfügung. Im weitesten Sinne handelt es sich um Klassen, mit denen Wertegruppen, Felder, Schlüssel-/Wertepaare und ähnliche Datensammlungen manipuliert werden können.

4.3.1 Einführung in die Welt der Kollektionen

Viele Programmiersprachen bieten neben einfachen Arrays auch so genannte Hashes. Dies sind Arrays mit nichtnumerischen Indizes. .NET bietet gleich fünf Varianten solcher Gebilde, mit denen sehr elegant programmiert werden kann.

Übersicht der Klassen

Nachfolgend finden Sie eine Liste der Basisklassen des Namensraumes `System.Collections`:

**System.
Collections**

► `ArrayList`

Eine Werte-Liste, die ähnlich wie ein Array verwendet werden kann. Die Indizes sind numerisch.

► `Hashtable`

Die Implementierung des klassischen Hashes; die Indizes (Schlüssel) sind immer Zeichenketten. Wenn Objekte verwendet werden, wandelt `Hashtable` diese intern in Zeichenketten um, wie es die Implementierung von `ToString` für diese Objekte verlangt.

► `SortedList`

Diese Klasse bietet beide Indexvarianten, Zahlen und Zeichenketten, realisiert also ein Array mit dem Verhalten eines Hashes.

► `Queue`

Arrays und Hashes erlauben den wahlfreien Zugriff auf die Elemente. Bei Objekten der Klasse `Queue` ist dies anders. Hier erfolgt der Zugriff nach dem FIFO-Prinzip (FIFO = First In, First Out). Elemente, die zuerst abgelegt wurden, müssen auch zuerst wieder entnommen werden.

► `Stack`

Wie bei der Klasse `Queue` ist der wahlfreie Zugriff nicht möglich, das Speicherprinzip ist hier allerdings FILO (FILO = First In, Last Out). Realisiert wird ein Stapel – das zuletzt abgelegte Element muss zuerst wieder entnommen werden.

► Spezielle Kollektionen, so genannte Wörterbücher: `ListDictionary`, `HybridDictionary`, `StringDictionary`

Diese Wörterbücher sind spezialisierte Formen der `Hashtable`-Klasse und erleichtern den Umgang mit Schlüssel-/Wertpaaren.

► NameValueCollection

Diese Kollektion wird intern eingesetzt, um Attribute und ihre Parameter von XML- oder HTML-Tags oder auch Styleanweisungen zu speichern. Verarbeiten Sie ähnliche Daten, ist der Einsatz auch in eigenen Programmen sinnvoll.

Allen Objekten ist gemeinsam, dass die Einträge mit *For Each* sequenziell ausgelesen werden können.

Zugriff auf den Namensraum

Namensraum aktivieren

Der Zugriff auf den Namensraum ist in ASP.NET standardmäßig nicht aktiviert. Sie müssen deshalb Ihren Programmen folgende Direktive voranstellen:

```
<%@Import Namespace="System.Collections" %>
```

Wenn Sie mit hinterlegtem Code arbeiten, ist diese Anweisung im Kopf des Programms zu ergänzen:

```
Imports System.Collections
```

Mit hinterlegtem Code arbeiten

Viele der folgenden Beispiele dieses Abschnitts verwenden hinterlegten Code. Der Abdruck der Listings beschränkt sich deshalb auf die *vb*-Dateien. Die Ausgabe erfolgt in diesen Fällen mit einer universellen *aspx*-Datei:

```
<% @Page Language="vb" Inherits="Addison.VBNet.Basis.class"
      src="class.vb"%>
<html>
  <head>
    <title>Kollektionen</title>
  </head>
  <body>
    <asp:label runat="server" id="ausgabe"/>
  </body>
</html>
```

Listing 4.4: Ausgabe der in den Beispielen erzeugten Daten

Die Attribute `Inherits` und `src` ändern sich jeweils, die korrekte Angabe finden Sie bei den Listingunterschriften.

Besonderheiten beim Einsatz von Visual Studio .NET

Wenn Sie Visual Studio .NET verwenden, wird der Designer beim Anlegen eine etwas andere – funktional gleichwertige – Syntax verwenden. Lassen Sie sich davon nicht irritieren. Die meisten Programme auf der Buch-CD sind mit Visual Studio .NET erstellt. Die @Page-Direktive am Anfang der Seite sieht hier etwa folgendermaßen aus:

```
<%@ Page language="vb"
    Codebehind="class.aspx.vb"
    AutoEventWireup="false"
    Inherits="Addison.VBNet.Basis.class" %>
```

4.3.2 Einfache Listen mit ArrayList

Eine `ArrayList` ist eine Sammlung beliebiger Objekte. Die Ablage erfolgt mit numerischen Indizes. Die Objekte können anhand des Indizes oder des Objekts selbst entnommen werden. Zum Hinzufügen wird die Methode `Add` eingesetzt. Entfernt werden Objekte unter Angabe des Index mit `RemoveAt`, bei Angabe des Objekts mit `Remove`. An einer bestimmten Stelle – mit Indexnummer – kann ein Element mit `Insert` eingefügt werden. `Clear` entfernt alle Elemente.

Methoden

Der Zugriff auf die Elemente erfolgt in VB.NET über einen Array-Schreibweise, was folgendermaßen aussieht:

Eigenschaften

```
element(3)
```

Das funktioniert, weil der Elementabruf die Standardeigenschaft des Objekts ist. Die Anzahl kann mit `Count` ermittelt werden. Der aktuell reservierte Platz wird mit `Capacity` festgelegt. Der Standardwert ist 16; werden mehr Elemente hinzugefügt, wird das Array bei Überschreiten der Grenze verdoppelt. `Capacity` kann nie kleiner als `Count` sein.

Ein Beispiel zeigt die Verwendung:

```
Public Class CollectionArrayList
    Inherits System.Web.UI.Page
    Protected WithEvents ausgabe As
    System.Web.UI.WebControls.Label

    Private Sub Page_Load(ByVal sender As Object,
        ByVal e As EventArgs) _
```

```

Handles MyBase.Load
    ausgabe.Text = "<h3>Aus ArrayList erzeugt.</h3>"
    Dim album As ArrayList = New ArrayList()
    ausgabe.Text += "Erlaubt sind derzeit " <img alt="mouse cursor icon" data-bbox="695 105 715 115"/>
        + album.Capacity.ToString() + " Werte"
    album.Add("The Wall")
    album.Add("Animals")
    album.Add("Ummelgumme1") ' Fehler
    album.Add("Atom Heart Mother")
    album.Add("Meddle")
    album.Add("Wish You Were Here")
    album.Add("The Final Cut")
    album.Add("The Division Bell")
    album.Add("The Dark Side Of The Moon")
    album.Remove("Ummelgumme1")
    album.Insert(2, "Ummagumma")

    Dim titel As String
    For Each titel In album
        ausgabe.Text += "<br/>" + titel
    Next
    ausgabe.Text += "<br/>Es waren: " <img alt="mouse cursor icon" data-bbox="640 405 660 415"/>
        + album.Count.ToString() + " Werte"

End Sub
End Class

```

Listing 4.5: Verwendung von ArrayList (Aus CollectionArrayList.aspx.vb)

Wie es funktioniert

Zuerst wird hier ein Objekt vom Typ ArrayList erzeugt:

```
Dim album As ArrayList = New ArrayList()
```

Dann werden mehrere Werte hinzugefügt, darunter ein falscher (in der Zeile steht der Kommentartext 'Fehler):

```
album.Add("The Wall")
```

Ein falscher Wert wird wieder entfernt:

```
album.Remove("Ummelgumme1")
```

An seiner Stelle wird das korrekte Element eingefügt:

```
album.Insert(2, "Ummagumma")
```

Nullbasierter Index

Die Indizierung ist nullbasiert. Wenn ein Element mit dem Index 2 eingefügt wird, verschieben sich die alten Elemente ab Index 2; 2 wird also 3 usw.

Die Ausgabe mit *For Each* ist so einfach wie möglich gehalten. Falls Ihre Liste verschiedene Objekte enthält, müssen Sie hier natürlich eine entsprechende Sonderbehandlung einfügen.

```
Aus ArrayList erzeugt.  
Erlaubt sind derzeit 16 Werte  
The Wall  
Animals  
Ummagumma  
Atom Heart Mother  
Meddle  
Wish You Were Here  
The Final Cut  
The Division Bell  
The Dark Side Of The Moon  
Es waren: 9 Werte
```

Abbildung 4.8: Ausgabe von Daten aus einer *ArrayList*

Weitere Methoden, die Sie verwenden können, sind unter anderem:

Methoden

► Reverse

Hiermit wird die Reihenfolge der Elemente gedreht.

► Sort

Sortiert den Inhalt. Beachten Sie, dass Sie eigene Sortieralgorithmen mit der Implementierung der Schnittstelle *IComparer* entwickeln können. Wie das geht, wurde bereits im Abschnitt »Definition eigener Suchalgorithmen« ab Seite 220 gezeigt.

► Contains

Die Methode ermittelt, ob ein bestimmtes Element in der Liste ist.

► BinarySearch

Hiermit können Sie Objekte ebenfalls suchen. Für kompliziertere Gebilde kann die *IComparer*-Schnittstelle zur Implementierung eigener Suchmethoden genutzt werden.

► AddRange

Fügen Sie mit dieser Methode mehrere Elemente hinzu. Der Parameter soll vom Typ *ICollection* sein; eine Schnittstelle, die viele andere Datenlisten im .NET-Framework auch verwenden und die sehr gut die Kompatibilität sicherstellt, beispielsweise zu *DataGridView* (siehe Kapitel 9, »Datenbanken und ADO.NET« ab Seite 823).

4.3.3 Schlüssel/Werte-Paare mit Hashtable speichern

Falls Sie das alte ASP kennen, kann man diese Implementierung mit *Scripting.Dictionary* vergleichen. Allerdings bietet .NET – wie nicht anders zu erwarten war – mehr als dies.

Methoden Eine Hashtable ist eine Sammlung beliebiger Objekte. Die Ablage erfolgt mit eigenen, alphanumerischen Schlüsseln. Objekte können anhand des Schlüssels oder des Objekts selbst entnommen werden. Zum Hinzufügen wird die Methode `Add` eingesetzt. Entfernt werden Objekte unter Angabe des Index mit `RemoveAt`, bei Angabe des Objekts mit `Remove`. An einer bestimmten Stelle – mit Indexnummer – kann mit eingefügt werden. `Insert` entfernt alle Elemente.

Das folgende Beispiel zeigt den Umgang mit Hashtable. Gespeichert werden hier als Werte nicht Zeichenketten, sondern Strukturen:

```
Public Class CollectionBooks
    Inherits System.Web.UI.Page
    Public ausgabe As Label

    Private Structure bookdata
        Public autor As String
        Public preis As Double
        Public seiten As Integer
    End Structure

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        ausgabe.Text = "<h3>Aus Hashtable erzeugt.</h3>"
        Dim buchliste As Hashtable = New Hashtable()
        Dim buch As bookdata = New bookdata()
        buch.autor = "Jörg Krause"
        buch.preis = 24.95
        buch.seiten = 360
        buchliste.Add("ASP.NET lernen", buch)
        buch.autor = "Jörg Krause, Uwe Bünning"
        buch.preis = 59.95
        buch.seiten = 960
        buchliste.Add("ASP.NET mit VB.NET", buch)
        Dim abuch As ArrayList = New ArrayList()
        Dim atitel As ArrayList = New ArrayList()
        atitel.AddRange(buchliste.Keys)
        abuch.AddRange(buchliste.Values)
        Dim i As Integer
```



```

        For i = 0 To atitel.Count - 1 Step i + 1
            ausgabe.Text += "<br/><b>Titel: " + atitel(i) + "</b>"
            Dim bd As bookdata = CType(abuch(i), bookdata)
            ausgabe.Text += "<br/>Autor: " + bd.autor
            ausgabe.Text += "<br/>Preis: " + bd.preis.ToString()
            ausgabe.Text += "<br/>Seiten: " + bd.seiten.ToString()
        Next
    End Sub
End Class

End Class

```

Listing 4.6: Verwendung von Hashtable (Ausschnitt aus CollectionBooks.aspx.vb)

Das Hashtable-Objekt soll Bücher speichern. Dazu wird eine Struktur definiert, die die Daten aufnimmt:

Wie es funktioniert

```
Private Structure bookdata
```

Diese enthält drei Variablen, in denen der Autor, der Preis und die Anzahl Seiten gespeichert werden.

Im Hauptprogramm wird zuerst das Hashtable-Objekt erzeugt:

```
Dim buchliste As Hashtable = New Hashtable()
```

Dann wird eine Instanz der Struktur benötigt:

```
Dim buch As bookdata = New bookdata()
```

Dieser Struktur werden nun die Werte zugewiesen:

```

buch.autor = "Jörg Krause"
buch.preis = 24.95
buch.seiten = 360

```

Das fertige Objekt wird dann der Hashtable hinzugefügt:

```
buchliste.Add("ASP.NET lernen", buch)
```

Mit einem weiteren Objekt soll jetzt der Zugriff auf die Elemente erfolgen. Auf direktem Wege ist das leider nicht möglich, deswegen werden Schlüssel und Werte einzeln in Instanzen von `ArrayList` abgelegt. Lesen Sie in Abschnitt 4.3.4, »Zugriff auf Aufzählungen« ab Seite 231, wie dies mit Aufzählungen eleganter erfolgen kann. Hier zuerst die beiden `ArrayList`-Objekte:

```

Dim abuch As ArrayList = New ArrayList()

Dim atitel As ArrayList = New ArrayList()

```

Dann werden die Titel zugewiesen; dies sind die Schlüssel der Hashtable:

```
atitel.AddRange(buchliste.Keys)
```

Mit der Eigenschaft `Values` erfolgt der Zugriff auf die Strukturen:

```
abuch.AddRange(buchliste.Values)
```

Das Auslesen der Elemente erfolgt in einer *For*-Schleife, der Umfang des Arrays wird mit der Eigenschaft `Count` bestimmt:

```
For i = 0 To atitel.Count - 1 Step i + 1
```

Auf den Titel kann direkt über den Index des `Arraylist`-Objekts zugegriffen werden:

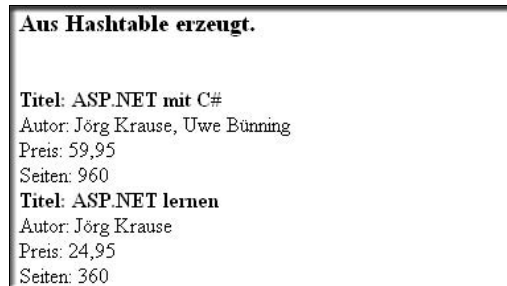
```
ausgabe.Text += "<br/><b>Titel: " + atitel(i) + "</b>"
```

Zum Auslesen der Struktur muss diese erst wieder hergestellt werden, es ist also eine explizite Typkonvertierung erforderlich:

```
Dim bd As bookdata = CType(abuch(i), bookdata)
```

Auf die Instanzen erfolgt nun der Zugriff in gewohnter Weise (die numerischen Werte werden zusätzlich noch mit `ToString` in Zeichenketten konvertiert):

```
ausgabe.Text += "<br/>Autor: " + bd.autor
```



```
Aus Hashtable erzeugt.  
  
Titel: ASP.NET mit C#  
Autor: Jörg Krause, Uwe Bünning  
Preis: 59,95  
Seiten: 960  
Titel: ASP.NET lernen  
Autor: Jörg Krause  
Preis: 24,95  
Seiten: 360
```

Abbildung 4.9: Ausgabe der in einer Hashtable gespeicherten Daten

Interner Aufbau der Klasse Hashtable

Der Ansatz, die Werte und Schlüssel über die `ICollection`-Schnittstelle zu entnehmen, ist zwar direkt und oft sinnvoll, kann aber vereinfacht werden. Intern werden nämlich diese Werte und Schlüssel nicht einzeln gespeichert, sondern in Objekten vom Typ

DictionaryEntry. Ein Durchlaufen der Liste kann auch mit *For Each* erfolgen, was folgendermaßen aussieht:

```
Dim dasbuch As DictionaryEntry
For Each dasbuch In buchliste
    ausgabe.Text += "<br/><b>Titel: " + dasbuch.Key + "</b>"
    Dim bd As bookdata = CType(dasbuch.Value, bookdata)
    ausgabe.Text += "<br/>Autor: " + bd.autor
    ausgabe.Text += "<br/>Preis: " + bd.preis.ToString()
    ausgabe.Text += "<br/>Seiten: " + bd.seiten.ToString()
Next
```

Listing 4.7: Vereinfachte Ausgabe (Collectionbooks2.aspx, die Code-Datei, aus der dieser Ausschnitt stammt, finden Sie unter dem Namen Collectionbooks2.aspx.vb)

Objekte vom Typ `DictionaryEntry` kennen zwei Eigenschaften: `Key` und `Value`, mit denen dann auf Schlüssel und Werte zugegriffen werden kann.

Eigenschaften

4.3.4 Zugriff auf Aufzählungen

Vielfach geben Eigenschaften Aufzählungen zurück oder Objekte können ihren Inhalt alternativ als Kollektion exportieren. Viele Klassen verfügen über eine Methode mit dem Namen `GetEnumerator`. Zurückgegeben wird eine Aufzählung, deren Definition auf der Schnittstelle `System.Collection.IEnumerator` basiert. Weit über 100 Klassen implementieren diese Schnittstelle. Der Umgang mit `IEnumerator` ist deshalb elementares Handwerkszeug.

GetEnumerator

Anwendungsbeispiel

Das folgende Beispiel zeigt den Einsatz – der Zugriff erfolgt auf eine Kollektion von Tabellenzellen einer HTML-Tabelle:

**Tabellen
manipulieren**

```
<form runat="server" ID="Form1">
    <asp:Table id="tabelle" runat="server" />
    Auswahl einer Reihe:
    <asp:DropDownList id="liste" runat="server" />
    <asp:Button id="senden" Text="Reihe ermitteln" ↵
        onclick="holeReihe" runat="server" />
    <hr noshade width="300" align="left">
    <asp:Label id="ausgabe" runat="server" />
</form>
```

Listing 4.8: Dynamisch erzeugte Tabelle und Auswahl einer Zellenreihe mit `GetEnumerator` (GetEnumerator.aspx)

Der Code dazu wurde in eine Code-Behind-Datei ausgelagert:

```
Public Class GetEnumerator
    Inherits System.Web.UI.Page

    Protected WithEvents tabelle As Table
    Protected WithEvents senden As Button
    Protected WithEvents ausgabe As Label
    Protected WithEvents liste As DropDownList

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim zahlReihen As Integer = 4
        Dim zahlZellen As Integer = 8
        Dim rnd As Random = New Random(100)
        Dim j As Integer
        For j = 0 To zahlReihen - 1 Step 1
            Dim reihe As TableRow = New TableRow()
            Dim i As Integer
            For i = 0 To zahlZellen - 1 Step 1
                Dim zelle As TableCell = New TableCell()
                zelle.Text = rnd.Next(1000, 9999).ToString()
                reihe.Cells.Add(zelle)
            Next
            tabelle.Rows.Add(reihe)
        Next
        If Not IsPostBack Then
            Dim arrayReihe As ArrayList = New ArrayList()
            Dim k As Integer
            For k = 1 To zahlReihen Step 1
                arrayReihe.Add(k.ToString())
            Next
            liste.DataSource = arrayReihe
            liste.DataBind()
        End If
    End Sub

    Public Sub holeReihe(ByVal sender As Object,
        ByVal e As EventArgs) _
        Handles senden.Click
        Dim reihe As Integer = liste.SelectedIndex
        Dim aktuelleZelle As TableCell
        Dim aufzaehlung As IEnumerator
            = tabelle.Rows(reihe).Cells.GetEnumerator()
        ausgabe.Text = "Folgende Reihe wurde ausgew&auml;hlt: "
            &lt;br/>"
        While aufzaehlung.MoveNext()
            aktuelleZelle = CType(aufzaehlung.Current, TableCell)
```

```

        ausgabe.Text += " " + aktuelleZelle.Text
    End While
End Sub

```

```
End Class
```

Listing 4.9: Dynamisches Erzeugen der Tabelle (Ausschnitt aus GetEnumerator.aspx.vb)

Bevor die Funktion im Detail behandelt wird, soll ein Blick auf die Ausgabe das Ergebnis zeigen:

Aufzählungen verarbeiten

9718	2432	7000	9121	4191	9537	7404	6494		
4142	2339	9683	5848	4399	5454	9813	5341		
9083	9956	4457	1315	1935	4435	9030	9703		
5504	8600	9193	6835	2512	6553	1943	8145		

Auswahl einer Reihe:

Folgende Reihe wurde ausgewählt:
5504 8600 9193 6835 2512 6553 1943 8145

Abbildung 4.10: Tabellarische Ausgabe und Zugriff auf Tabelleninhalte

Das Programm zeigt neben der Nutzung der Aufzählung noch zwei besondere Techniken, die anderswo im Buch besprochen werden:

Wie es funktioniert

- ▶ Das dynamische Erzeugen einer Tabelle wird an mehreren Stellen im Buch genauer gezeigt, meist im Zusammenhang mit anderen Funktionen, unter anderem im Abschnitt 9.3.5, »Aktualisieren einer Datenbank mit CommandBuilder« ab Seite 904.
- ▶ Das Binden einer Datenquelle an ein DropDownList-Steuerelement wird im Abschnitt 6.5.1, »Listen erzeugen und verwalten« ab Seite 556 behandelt.

Betrachten Sie zuerst den HTML-Teil. Dort wird eine Tabelle ganz allgemein definiert – noch ohne irgendwelchen Inhalt:

Tabelle aufbauen

```
<asp:table id="tabelle" runat="server"/>
```

Dann wird noch ein DropDownList-Steuerelement erzeugt, auch ohne Inhalt:

```
<asp:DropDownList id="liste" runat="server"/>
```

Das Formular wird komplettiert durch eine Sendeschaltfläche:

```
<asp:Button id="senden" Text="Reihe ermitteln" onclick="holeReihe"
runat="server" />
```

Damit eine Ausgabe zu sehen ist, folgt noch ein Label-Steuererelement:

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

Die eigentliche Arbeit wird wieder über ein Programm erledigt. Es beginnt mit `Page_Load`. Zuerst werden mehrere Variablen definiert, unter anderem auch ein Zufallsgenerator, dessen Startwert festgelegt ist:

```
Dim rnd As Random = New Random(100)
```

Es folgt eine *For*-Schleife, die die Reihen der Tabellen erzeugt:

```
For j = 0 To zahlReihen - 1 Step 1
```

Dazu wird jeweils ein neues Reihen-Objekt erzeugt:

```
Dim reihe As TableRow = New TableRow()
```

Innerhalb der Reihe sind nun die Zellen zu erzeugen:

```
For i = 0 To zahlZellen - 1 Step 1
```

Auch hier wird für jede Zelle ein entsprechendes Objekt erzeugt:

```
Dim zelle As TableCell = New TableCell()
```

Jede Zelle wird nun mit einer Zufallszahl gefüllt. Die Datenquelle kann natürlich auch anders gestaltet werden:

```
zelle.Text = rnd.Next(1000, 9999).ToString()
```

Dann wird die fertige Zelle der aktuellen Reihe hinzugefügt:

```
reihe.Cells.Add(zelle)
```

Ist eine Reihe fertig, wird auch diese an die Tabelle angefügt:

```
tabelle.Rows.Add(reihe)
```

Liste aufbauen

Jetzt ist noch das Formular auszuwerten. Der Vorgang besteht aus zwei Teilen. Zuerst muss noch die Liste aufgebaut werden, allerdings nur beim ersten Aufruf des Programms (`IsPostBack` gleich `False`). Dazu wird ein `ArrayList`-Objekt instanziiert:

```
Dim arrayReihe As ArrayList = New ArrayList()
```

Dieses wird mit so vielen Zahlen gefüllt, wie Reihen in der Tabelle existieren:

```
For k = 1 To zahlReihen Step 1
    arrayReihe.Add(k.ToString())
Next
```

Dieses Array wird nun als Datenquelle für die Liste verwendet:

```
liste.DataSource = arrayReihe
```

Mit der Bindung erscheinen die Listeneinträge auf der Seite:

```
liste.DataBind()
```

Die Aufzählung wird nun benötigt, um eine Zeile der Tabelle auszulesen. Dies passiert jedoch erst, wenn das Formular gesendet wurde, also in der Methode *holeReihe*, die das Ereignis `onClick` serverseitig verarbeitet, was durch `Handles senden.Click` erreicht wird:

**Zugriff auf die
Aufzählung**

```
Public Sub holeReihe(ByVal sender As Object, ↵
                    ByVal e As EventArgs) _
Handles senden.Click
```

Zuerst wird der Index ermittelt, der in der Liste ausgewählt wurde:

```
Dim reihe As Integer = liste.SelectedIndex
```

Dann wird ein Zellenobjekt deklariert:

```
Dim aktuelleZelle As TableCell
```

Nun kann auf die Reihe einer Tabelle zugegriffen werden. Die Methode `GetEnumerator` liefert die komplette Reihe als Objekt vom Typ `IEnumerator`:

```
Dim aufzaehlung As IEnumerator
    = tabelle.Rows(reihe).Cells.GetEnumerator()
```

Nach einer Textausgabe für den Benutzer wird diese Aufzählung nun durchlaufen. Dabei gibt die Methode `MoveNext` solange `True` zurück, wie noch Elemente vorhanden sind. Gleichzeitig wird mit jedem Aufruf ein interner Zeiger der Aufzählung ein Element weiter gesetzt:

```
While aufzaehlung.MoveNext()
```

Das aktuelle Element wird nun mit der Eigenschaft `Current` gelesen:

```
aktuelleZelle = CType(aufzaehlung.Current, TableCell)
```

Der Rest beschäftigt sich mit der Ausgabe, die zu dem bereits gezeigten Ergebnis führt. Beachten Sie, dass in der Schleife mehrere Zuweisungen erfolgen und deshalb der Inhalt des Steuerelementes mit dem Operator `+=` erweitert wird.

```
ausgabe.Text += " " + aktuelleZelle.Text
```

Die Schnittstelle `IEnumerator` im Überblick

Die Schnittstelle `IEnumerator` ist sehr einfach. Nachfolgend finden Sie eine Zusammenfassung der Eigenschaften und Methoden:

► `Current`

Die einzige Eigenschaft der Schnittstelle gibt das aktuelle Element zurück.

► `MoveNext`

Mit dieser Methode wird die Aufzählung durchlaufen. Jeder Aufruf schaltet einen internen Zeiger auf diese Elemente eine Position weiter. Sind keine Elemente mehr vorhanden, wird *False* zurückgegeben, sonst *True*.

► `Reset`

Mit dieser Methode wird der Zeiger wieder auf das erste Element zurück gesetzt.

Beide Methoden haben keinen Parameter.

4.3.5 Wörterbücher

Wörterbücher speichern Daten in indizierter Form. Das trifft zwar auch für `Hashtable` zu, einige spezialisierte Klassen bieten jedoch etwas mehr als dies.

Sortierte Listen mit `SortedList` verarbeiten

**Listen
automatisch
sortieren**

Sortierfunktionen werden sehr häufig benötigt. Andere Sprachen liefern teilweise beeindruckende Sortiermöglichkeiten in Form spezieller Funktionen, anwendbar auf Arrays oder Hashes. Im

.NET-Framework steht Ihnen eine etwas andere Form zur Verfügung: `SortedList`. Diese Liste arbeitet ähnlich wie `Hashtable`, speichert die Daten jedoch von vornherein sortiert. Der Sortiervorgang erfolgt also bereits beim Erzeugen der Elemente.

Standardmäßig werden alle Basistypen aufsteigend sortiert, so wie die Elemente im Zeichensatz definiert sind. Wenn Sie eigene Datentypen speichern, funktioniert diese Sortierung nicht, abgesehen davon, dass aufsteigende Sortierungen nicht immer gewünscht werden. Als Lösung kommt die bereits vorgestellte Schnittstelle `IComparable` zur Anwendung. Die Logik dahinter ist einfach. Über die sicherste Information, wie ein Typ sortiert werden kann, verfügt der Typ selbst. Also wird nicht die Klasse `SortedList` erweitert, sondern der darin abgelegte Typ. Abgesehen davon kann unter den Überladungen des Konstruktors der Klasse auch die Angabe einer direkten Implementierung der Schnittstelle festgelegt werden. Folgendermaßen erzeugen Sie eine Instanz, wenn die Schlüssel die Vergleichsimplementierung mitbringen:

```
Dim sl As SortedList = New SortedList()
```

Verfügen Sie dagegen über eine Implementierung der Schnittstelle `IComparer`, geben Sie diese folgendermaßen an.

```
Dim sl As SortedList = New SortedList(MyComparer)
```

Verwechseln Sie nicht die Schnittstellen `IComparer` und `IComparable`. Die Schnittstelle `IComparer` wird verwendet, um eine Klasse zu implementieren, die zur Steuerung der Sortierfunktion anderer Klassen dient. Außerdem findet Sie Anwendung, um die Eindeutigkeit von Schlüsseln zu ermitteln. `IComparer` verlangt die Implementierung der Methode `Compare`, die den zur Sortierung nötigen Vergleich ausführt.



Die Schnittstelle `IComparable` wird im Beispiel `SortedList` verwendet, um die Sortierung der Liste zu überlassen. Wenn Sie also einen eigenen Typ, wie im folgenden Beispiel `MyFile`, mit einer Sortiereigenschaft versehen möchten, fügen Sie eine Methode `CompareTo` aus der Schnittstelle `IComparable` hinzu.

Das folgende Beispiel liest Dateien aus einem Verzeichnis und legt die Liste in einer Instanz des Typs `SortedList` ab. Gespeichert werden Objekte vom Typ `FileInfo`. Die »sortierende« Klasse enthält nur diesen Typ und wird außerdem um zwei Methoden erweitert:

```

Public Class MyFile
    Implements IComparable
    Public _file As FileInfo

    Sub New(ByVal file As FileInfo)
        _file = file
    End Sub

    Public Overrides Function ToString() As String
        Return _file.Name
    End Function

    Public Function CompareTo(ByVal o As Object) As Integer _
    Implements IComparable.CompareTo
        Dim mo As MyFile = CType(o, MyFile)
        Dim mos As Long = mo._file.Length
        If (mos < _file.Length) Then Return -1
        If (mos > _file.Length) Then Return 1
        Return 0
    End Function
End Class

```

*Listing 4.10: Klasse MyFile, die ein eigenes Sortierkriterium mitbringt
(Ausschnitt aus SortedListFiles.aspx.vb)*

Wie es funktioniert

Die Standardmethode `ToString`, die alle Klassen kennen, wird hier überschrieben. Wenn Sie einer `SortedList`-Instanz ein Objekt als Schlüssel hinzufügen, wird intern `ToString` aufgerufen, um eine Zeichenkette als Schlüssel verwenden zu können. Standardmäßig gibt `ToString` den Namen der Klasse zurück, wenn es sich um eine eigene Definition handelt (hier: »MyFile«). Schlüssel müssen jedoch eindeutig sein, deshalb wird die Methode überschrieben. Als Rückgabewert dient der Dateiname:

```
Return _file.Name
```

Die zweite Erweiterung des Basistyps besteht in der Definition des Sortierkriteriums. Die Sortierung wird durch den Rückgabewert gesteuert. -1 wird zurückgegeben, wenn der neue Wert kleiner als vorhandene ist; 1, wenn er größer ist und 0, wenn die Werte gleich sind. Was Sie als »größer«, »kleiner« oder »gleich« ansehen, ist Ihnen überlassen. Im Beispiel wird die Dateigröße verwendet und absteigend sortiert.

Die Liste muss nun noch benutzt werden. Dies geschieht hier in der Page_Load-Methode:

```
Public Class SortedListFiles
    Inherits System.Web.UI.Page
    Protected Directory As TextBox
    Protected UnSorted As Label
    Protected Sorted As Label
    Protected SortedList As Repeater

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If (Page.IsPostBack) Then
            Dim dir As String = Directory.Text
            Dim di As DirectoryInfo ←
                = New DirectoryInfo(Server.MapPath(dir))
            Dim sl As SortedList = New SortedList()
            UnSorted.Text = String.Empty
            Dim fi As FileInfo
            For Each fi In di.GetFiles()
                If (fi.Extension = ".vb") Then
                    sl.Add(New MyFile(fi), fi.Name)
                    UnSorted.Text ←
                        += String.Format("{0} = {1} Byte<br/>", ←
                            fi.Length.ToString(),
                                fi.Name,
                                    fi.Length)
                End If
            Next
            SortedList.DataSource = sl
            SortedList.DataBind()
        End If
    End Sub
End Class
```

Listing 4.11: Nutzung der Klasse *SortedList* (Ausschnitt aus *SortedListFiles.aspx.vb*)

Da das Sortierkriterium vom Speicherobjekt selbst mitgebracht wird, kann eine Standardliste verwendet werden:

Wie es funktioniert

```
Dim sl As SortedList = New SortedList()
```

Diese wird nun mit Dateien des ausgewählten Verzeichnisses gefüllt (das vollständige Programm auf CD enthält ein kleines Formular in der *aspx*-Datei). Beachten Sie, dass hier die Klasse als Schlüssel übergeben wird. Tatsächlich wird der Schlüssel auch so gespeichert – als Objekt – aber als Unterscheidungskriterium wird der Zeichenkettenwert verwendet (so wie die überschriebene Methode *ToString* ihn erzeugt):

```
sl.Add (New MyFile(fi), fi.Name)
```

Die Ausgabe an das Label-Steuerelement `Unsorted` dient nur der Kontrolle der Reaktion. Die eigentliche Ausgabe erfolgt über ein `Repeater`-Steuerelement. Dies ist freilich nur eine von vielen Ausgabeformen, wenn auch eine recht elegante. Die Daten-Steuerelemente werden in Abschnitt 6.6, »Vorlagengebundene Daten-Steuerelemente« ab Seite 566 noch ausführlich behandelt. Die Anwendung hier soll ein Ausblick auf die Möglichkeiten bieten. Die Zuweisung der Daten erfolgt mit einer einzigen Zeile:

```
SortedList.DataSource = sl
```

Datenbindung Die Datenbindung führt dann zur Übertragung der Daten zur Laufzeit:

```
SortedList.DataBind()
```

Die Ausgabe kann nun in der *aspx*-Datei vielfältig beeinflusst werden. Das hat den Vorteil, dass zur Laufzeit die Ausgabe beeinflusst werden kann. Für die Ausgabe der sortierten Liste soll der Name und die Größe der Datei herangezogen werden. Der Name ist der Wert des Listenelements:

```
<%# Container.DataItem.Value %>
```

Die Datenbindungssyntax

Dies ist die Datenbindungssyntax von ASP.NET. `Container.DataItem` enthält jeweils ein Element der sukzessive durchlaufenen (wiederholten) Elemente der Liste. Obwohl es sich bei `SortedList` um Objekte des Typs `DictionaryEntry` handelt, muss eine entsprechende Typumwandlung nicht erfolgen. Standardmäßig wird auf der Seite keine strikte Typkontrolle verwendet (`Option Strict Off` bzw. `strict="false"`). Alternativ müsste die Bindungs-Methode der Datenbindung verwendet werden, was jedoch die Ausführung verlangsamt. Da die Daten nur ausgegeben und nicht weiter verarbeitet werden, ist die implizite Konvertierung unkritisch. Deshalb kann auf die Eigenschaft `Value` direkt zugegriffen werden, die den Dateinamen enthält¹.

Die Zeile zur Ausgabe der Dateigröße sieht komplizierter aus, enthält aber letztlich keine Besonderheiten:

```
(<%# Container.DataItem.Key._file.Length.ToString() %> Bytes)
```

¹ C# verhält sich hier übrigens anders, ein Austausch der Code-Datei reicht zum Sprachwechsel in solchen Fällen nicht aus.

Lesen Sie diesen Ausdruck von links nach rechts. Die ersten beiden Elemente umfasst den Zugriff auf das aktuelle Element des Containers: `Container.DataItem`. Von diesem wird wieder der Schlüssel entnommen: `.Key`. Nun steht das Objekt – wiederum implizit konvertiert – vom Typ `MyFile` zur Verfügung. Dann folgt ein direkter Zugriff auf das Feld `_file`, das ein `FileInfo`-Objekt enthält. Dessen Eigenschaft `Length` wird nun gelesen und in eine Zeichenkette umgewandelt, wozu wieder `ToString` zum Einsatz kommt.

**Datenbindungs-
ausdrücke
entwickeln**

Dieser Abschnitt zeigte mehrere Methoden zum Umgang mit Listen. Dies beinhaltet zum Einen die Möglichkeit, Objekten Sortierkriterien mitzugeben und sie damit automatisch sortieren zu lassen, was allein durch die Verwendung von `SortedList` passiert. Es ist wichtig den Vorteil darin zu erkennen, dass Sie im Code keine Methode `Sort()` einsetzen müssen. Ihr Programm wird universeller, denn es sortiert immer so, wie die gespeicherten Objekte es erfordern. Der zweite Vorteil besteht in der Liste selbst, die die Schnittstelle `IEnumerable` implementiert. Allein dadurch kann die Zuweisung an Daten-Steuerelemente erfolgen. Das Erzeugen, Füllen und Ausgeben besteht also letztlich aus nur vier Zeilen Code. Das ist, auch wenn es auf den ersten Blick kompliziert anmutet, an Einfachheit nicht zu schlagen.

Vorteile von Listen

SortedList

Anzeigen

Unsortiert

AssemblyInfo.cs = 2894 Byte
Global.asax.cs = 1285 Byte
indexer.aspx.cs = 1104 Byte
OwnException.aspx.cs = 2029 Byte
SimpleExceptions.aspx.cs = 1551 Byte
SimpleExceptions2.aspx.cs = 1538 Byte
SortedListFiles.aspx.cs = 2119 Byte

Sortiert

- AssemblyInfo.cs (2894 Bytes)
- SortedListFiles.aspx.cs (2119 Bytes)
- OwnException.aspx.cs (2029 Bytes)
- SimpleExceptions.aspx.cs (1551 Bytes)
- SimpleExceptions2.aspx.cs (1538 Bytes)
- Global.asax.cs (1285 Bytes)
- indexer.aspx.cs (1104 Bytes)

Abbildung 4.11: Das aktuelle Verzeichnis, unsortiert und sortiert als Liste (unten)

4.3.6 Spezialisierte Klassen für Wörterbücher

Die bisherigen Klassen für Wörterbücher waren sehr universell. Es gibt noch einige spezialisierte Klassen.

Namensraum Diese Klassen sind in folgendem Namensraum enthalten:

`System.Collections.Specialized`

Er muss explizit eingebunden werden:

`Imports System.Collections.Specialized`

Kleine Datensammlungen mit `ListDictionary` effizient verwalten

**Alternative I zu
Hashtable**

`Hashtable` ist eine Klasse, deren Instanzen große Datenmengen aufnehmen können. Wenn nur eine geringe Anzahl Elemente zu verwalten ist – als gering gilt weniger als zehn – ist `ListDictionary` bei ansonsten vergleichbaren Eigenschaften besser geeignet.

Die Klasse verfügt unter anderem über die üblichen Eigenschaften `Item` (Elementwert) `Values` (Kollektion der Werte), `Keys` (Kollektion der Schlüssel) und `Count` sowie die Methoden `Add`, `Clear`, `Contains` und `Remove`.

Schlüssel müssen auch hier eindeutig sein.

Universelle Datensammlung mit `HybridDictionary` verwalten

**Alternative II zu
Hashtable**

Wenn Sie nicht sicher festlegen können, ob `Hashtable` oder `ListDictionary` eingesetzt werden sollen, verwenden Sie `HybridDictionary`. Dieser Klasse kann bei der Instanziierung eine Anfangsgröße zugewiesen werden, nach der entschieden wird, ob intern `Hashtable` oder `ListDictionary` verwendet werden. Auch dann, wenn anfangs `ListDictionary` eingesetzt wird, ändert sich dieses, wenn die Anzahl der Elemente während der Laufzeit zunimmt. Freilich kostet auch das Umschalten Zeit, der Einsatz sollte deshalb nur dann in Erwägung gezogen werden, wenn die Datenmenge tatsächlich stark schwankt.

Sie können außerdem im Konstruktor festlegen, ob Groß- und Kleinschreibung beim Vergleich der Schlüssel ignoriert werden sollen.

Ein gutes Beispiel sind übrigens wieder Dateilisten. Verzeichnisse können leer sein, wenige oder extrem viele Elemente enthalten und es gibt kaum Vorhersagemöglichkeiten über den Umfang des Inhalts.

Beispiel

```
Public Class MyHybridDictionary
    Inherits System.Web.UI.Page
    Protected Directories As DropDownList
    Protected FileList As Repeater

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Page.IsPostBack Then
            Dim di As DirectoryInfo = _
                New DirectoryInfo(Server.MapPath("../"))
            Directories.DataSource = di.GetDirectories()
            Directories.DataBind()
        Else
            Dim di As DirectoryInfo = _
                New DirectoryInfo(Server.MapPath("../" & _
                    Directories.SelectedItem.Text))
            Dim hl As HybridDictionary = New HybridDictionary()
            Dim fi As FileInfo
            For Each fi In di.GetFiles()
                hl(fi.Name) = fi.Length.ToString()
            Next
            FileList.DataSource = hl.Keys
            FileList.DataBind()
        End If
    End Sub
End Class
```

Listing 4.12: Nutzung von *HybridDictionary* (*HybridDictionary.aspx.vb*)

Das Beispiel zeigt wieder die Datenbindungstechnik. Die Ausgabe der Verzeichnisse erfolgt an ein DropDownList-Steuerelement, das folgendermaßen definiert wurde:

Wie es funktioniert

```
<asp:DropDownList Runat="server" ID="Directories"/>
```

Beachten Sie, dass die Liste Dateinamen und Dateigrößen aufnimmt. *HybridList* kann für die Zuweisung neben der Methode *Add* die *Array*-Schreibweise verwenden, weil *Item* die Standardeigenschaft ist, die VB.NET einsetzt, wenn kein Eigenschaftsname angegeben wurde:

```
hl(fi.Name) = fi.Length.ToString()
```

Damit die Datenbindung besonders einfach ausfällt, werden für diese Ausgabe nur die Schlüssel übergeben:

```
FileList.DataSource = hl.Keys
```

Das Repeater-Steuerelement wird in der *aspx*-Datei *HybridDictionary.aspx* definiert:

```
<body MS_POSITIONING="GridLayout">
<h1>HybridDictionary</h1>
Wählen Sie Verzeichnis, dessen Inhalt angezeigt werden soll:
<form id="HybridDictionary" method="post" runat="server">
  <asp:DropDownList Runat="server" ID="Directories"/>
  <input type="submit" value="Anzeigen"/>
</form>
<asp:Repeater Runat="server" ID="FileList">
  <HeaderTemplate><ul></HeaderTemplate>
  <ItemTemplate>
    <li><%# Container.DataItem %></li>
  </ItemTemplate>
  <FooterTemplate></ul></FooterTemplate>
</asp:Repeater>
</body>
```

Listing 4.13: Auswahlliste und Ausgabeliste (Ausschnitt aus *HybridDictionary.aspx*)

Der Zugriff auf die Datenelemente der Liste ist hier extrem einfach, da die Auswahl eindeutiger Daten bereits zuvor erfolgte:

```
<%# Container.DataItem %>
```

HybridDictionary

Wählen Sie Verzeichnis, dessen Inhalt angezeigt werden soll:

data ▼ Anzeigen

- axischeck.xml
- getfiles.xslt
- getfiles2.xslt
- mainpage.xslt
- news.xml
- news.xsd
- news.xsx

Abbildung 4.12: Auswahl des Verzeichnisses und Ausgabe der enthaltenen Dateien

Für alle Listen gilt, dass die optimale Bindung an Steuerelemente mit Wiederholungseigenschaften eines der großen Vorteile von

ASP.NET ist. Zum einen verbleiben Gestaltungsoptionen im HTML-Teil. Zum anderen beschränkt sich die Zuweisung auf die Angabe der Datenquelle.

Zeichenketten mit StringDictionary verwalten

Wenn absehbar ist, dass nicht beliebige Objekte sondern »nur« Zeichenketten gespeichert werden sollen, ist `StringDictionary` die beste Wahl. Sowohl Schlüssel als auch Werte müssen Zeichenketten sein. Die Schlüssel weisen eine Besonderheit auf. Sie unterscheiden nicht zwischen Groß- und Kleinschreibung. Dies deutet auf einen besonderen Einsatzzweck hin, nämlich der Speicherung von Attribut/Parameter-Paaren, wie sie in HTML vorkommen. Bei Attributnamen ist Groß- und Kleinschreibung nicht definiert.

Zeichenkette

Das folgende Beispiel liest alle Attribute eines HTML-Elementes aus und zeigt sie als Liste an. Zuerst wird die *aspx*-Datei mit der Definition eines Bildes mit vielen Attributen und der Ausgabeliste gezeigt:

**HTML-Attribute
speichern**

```
<body MS_POSITIONING="GridLayout">
  <h1>StringDictionary</h1>
  
</p>
<asp:Repeater Runat="server" ID="Attributes">
  <HeaderTemplate><ul></HeaderTemplate>
  <ItemTemplate>
    <li>
      <%# Container.DataItem.Key %>
      =
      <%# Container.DataItem.Value %>
    </li>
  </ItemTemplate>
  <FooterTemplate></ul></FooterTemplate>
</asp:Repeater>
</body>
```

**Listing 4.14: Auslesen und Ausgaben von Attributen eines HTML-Elements
(StringDictionary.aspx)**

»Zwischengespeichert« werden sie in einem Objekt vom Typ `StringDictionary`:

```
Public Class MyStringDictionary
  Inherits System.Web.UI.Page
```

```

Protected MyPicture As HtmlImage
Protected Attributes As Repeater

Private Sub Page_Load(ByVal sender As Object,
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Dim sd As StringDictionary = New StringDictionary()
    Dim key As String
    For Each key In MyPicture.Attributes.Keys
        sd.Add(key, MyPicture.Attributes(key))
    Next
    Attributes.DataSource = sd
    Attributes.DataBind()
End Sub
End Class

```

Listing 4.15: Verwendung von *StringDictionary* zum Speichern von Attributen
(*StringDictionary.aspx.vb*)

Wie es funktioniert

Der Aufbau der Liste erfolgt in einer Zeile, wobei die Kollektion der Schlüssel durchlaufen wird:

```
sd.Add(key, MyPicture.Attributes(key))
```

Auch eine Instanz vom Typ *StringDictionary* kann als Datenquelle dienen:

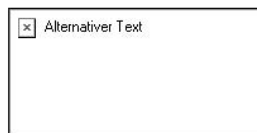
```
Attributes.DataSource = sd
```

Datenbindung

Da es sich um Schlüssel/Werte-Paare handelt, muss die Auswahl des gerade benötigten Datenteils mit der Datenbindungssyntax erfolgen, folgendermaßen beispielsweise für die Schlüssel:

```
<%# Container.DataItem.Key %>
```

StringDictionary



- width = 200
- alt = Alternativer Text
- src = data/bild.jpg
- height = 100
- border = 1
- class = Bild

Abbildung 4.13: Analyse eines HTML-Elementes

Eine Umwandlung des Typs in eine Zeichenkette mit `ToString` entfällt hier, denn `StringDictionary` kann ohnehin nur Zeichenketten enthalten.

Auch bei `StringDictionary` müssen die Schlüssel eindeutig sein. Durch die fehlende Unterscheidung von Groß- und Kleinschreibung ist die Anzahl überdies weiter begrenzt. In der Webserverprogrammierung treten aber weitere, spezielle Fälle von Wertesammlungen auf, die genau an dieser Vorgabe scheitern. Es ist möglich, per HTTP Daten zu empfangen, bei denen mehrere identische Schlüssel auftreten. Da dies ein reguläres Verhalten ist, muss eine spezielle Klasse für Verwaltung verwendet werden.

Besondere Datenpaare mit `NameValueCollection` verwalten

Angenommen, Sie definieren in einem Formular ein `<select>`-Element. Dann ist auch eine Mehrfachauswahl der Elemente möglich. Die folgende Abbildung zeigt dies:

Spezialfälle

Welche Systeme zur Webserverprogrammierung kennen Sie?

PHP
JSP
Perl/CGI
ASP

Anzeigen

__VIEWSTATE=dDwtNjYxODIzMTE2O3Q8O2w8aTwzPjs%
2bO2w8dDzwPHA8bDxUZXh0Oz47bDxfK1ZJRVdTVEFURT1kRHd0Tm
2bPjs%2bOz%2bOz4%2bOz7BBJbm%2b%2bO9RirlPzphNzBerYVwng%3d
&Selection=jsp&Selection=cgi&Selection=asp

Abbildung 4.14: In Formularen können Schlüssel bei Schlüssel/Werte-Paaren mehrfach auftreten.

Dasselbe gilt auch für die GET-Parameter eines URL. Wenn Sie diese Werte nun zum Zweck der Weiterverarbeitung in einem Wörterbuch speichern möchten, verwenden Sie `NameValueCollection`. Dieser Typ hat ein definiertes Verhalten für den Fall, dass Sie einen Schlüssel abrufen, der mehrfach auftritt. Sie erhalten in diesem Fall eine kommaseparierte Liste der Werte.

Die direkte Ausgabe der Werte mit `NameValueCollection` zeigt die folgende Abbildung:

```
Selection = PHP,ASP,JSP,CGL,DOTNET
```

Abbildung 4.15: Ausgabe der Werte nach der Speicherung im Wörterbuch unter einem Schlüssel

Da eine solche Liste nicht immer benötigt wird, bietet die Klasse einige spezielle Eigenschaften und Methoden, die über die Möglichkeiten der anderen Wörterbücher hinausgehen. Normalerweise steht immer eine Eigenschaft `Keys` zur Verfügung, die eine Aufzählung – meist von `ICollection` stammend – der Schlüssel zurückgibt. Damit kann beispielsweise *For Each* etwas anfangen. Dies wurde im Beispiel in Listing 5.14 gezeigt. Wenn Sie aber alle Werte eines Schlüssels einzeln benötigen, verwenden Sie `GetValues("Schlüsselname")`.

**Infos über
Request.Form
und Request.
QueryString
finden Sie in
Kapitel 8**

Die oben bereits angesprochenen Einsatzfälle sind POST- und GET-Parameter. Bevor Sie nun auf die Idee kommen, die Kollektionen der Parameter in Instanzen der Klasse `NameValueCollection` zu speichern, lohnt ein Blick auf die Liste der Eigenschaften und Methoden von `Request.Form` und `Request.QueryString`. Beide Eigenschaften geben Objekte zurück, die selbst von `NameValueCollection` abstammen. Insofern erübrigt sich der Einsatz für diese Fälle eigentlich, das Framework hat Ihnen hier einiges an Arbeit abgenommen. Allerdings können Sie eine direkte Zuweisung vornehmen, wie sie im folgenden Beispiel verwendet wird:

```
Public Class MyNameValueCollection
    Inherits System.Web.UI.Page
    Protected CheckForm As Label
    Protected FormValue As Repeater

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If MyBase.IsPostBack Then
            Dim nvc As NameValueCollection
            = New NameValueCollection()
            If Not Request.Form Is Nothing Then
                nvc = Request.Form
                CheckForm.Text = String.Empty
                Dim de As String
                For Each de In nvc.GetValues("Selection")
                    CheckForm.Text += String.Format("{0}<br/>", de)
                Next
            End If
        End If
    End Sub
End Class
```

```
End If
End Sub
End Class
```

*Listing 4.16: Übernahme von Formularwerten
(Ausschnitt aus NameValueCollection.aspx.vb)*

Die Zuweisung der `Form`-Kollektion ist der Schlüssel dieses Beispiels:

Wie es funktioniert

```
nvc = Request.Form
```

Auf die möglicherweise mehrfachen Werte eines Schlüssels wird dann mit `GetValues` zugegriffen:

```
For Each de in nvc.GetValues("Selection")
```

Korrespondierend dazu sieht die Liste im HTML-Teil folgendermaßen aus:

```
<select name="Selection" runat="server" ←
    id="Selection" multiple>
    <option value="php">PHP</option>
    <option value="jsp">JSP</option>
    <option value="cgi">Perl/CGI</option>
    <option value="asp">ASP</option>
    <option value="dotnet">ASP.NET</option>
</select>
```

*Listing 4.17: Definition der im Code verwendeten Liste
(Ausschnitt aus NameValueCollection.aspx)*

Je nach Auswahl erscheinen die gewählten Werte nun einzeln.

NameValueCollection

Welche Systeme zur Webserverprogrammierung kennen Sie?

PHP
JSP
Perl/CGI
ASP

Anzeigen

php
cgi
asp
dotnet

Abbildung 4.16: Getrennte Anzeige der Werte, die in der Liste gewählt wurden

4.3.7 Spezielle Kollektionen

Wörterbücher enthalten immer Paare – Schlüssel und Werte. Es gibt aber in der Programmierung ebenso oft Bedarf an einfacheren Listen. Meist werden eindimensionale Arrays eingesetzt. Skriptsprachen wie Perl oder PHP verfügen über eine lange Liste von Arrayfunktionen, die in der Webprogrammierung vielfältig Verwendung finden. So lassen sich die Elemente eines Array rotieren oder wie in einem Stapel das erste oder letzte Element entnehmen.

Im .NET-Framework gibt es noch mehr ausgereifte Möglichkeiten, denn für zwei Spezialfälle steht jeweils eine Klasse bereit.

Warteschlangenverarbeitung mit Queue

Die Klasse `Queue` realisiert so etwas wie eine Warteschlange. Neue Elemente werden am Ende angehängt, während der Abruf des nächsten Wertes das Element vom Anfang holt und dabei entfernt. Man nennt dieses Verarbeitungsprinzip »First-In-First-Out« (FIFO).

Stapelverarbeitung mit Stack

Auch die Stapelverarbeitung wird gelegentlich gebraucht. Die Klasse `Stack` realisiert dies. Das Prinzip wird hier als »Last-In-First-Out« (LIFO) bezeichnet. Das zuletzt auf den Stapel gelegte Element muss auch zuerst wieder entnommen werden.

4.4 Mathematische Operationen

Das Framework liefert alle mathematischen Methoden, konkret aus der Klasse `System.Math`. In ASP.NET wird `System` automatisch eingebunden – dieser Namensraum enthält die Klasse. Die alten VB.NET-Funktionen lassen sich weiter verwenden, sollten aber aus Gründen der Vorwärtskompatibilität vermieden werden. Sie werden hier nicht weiter berücksichtigt.

4.4.1 Felder und Methoden der Klasse `System.Math`

In der Klasse `System.Math` stehen Felder als Quellen von konstanten Werten und Methoden zur Verfügung.

Felder

Zwei statische Felder dienen als Quelle zweier Konstanten: `E` für Eulersche Zahl und `PI` für π .

Methoden

Alle Methoden der Klasse `Math` sind statisch. Sie können also nicht `-3.Abs` schreiben, um den absoluten Betrag zu bestimmen, sondern müssen folgende Notation wählen: `Math.Abs(-3)`. Die Anwendung entspricht dem in anderen Programmierumgebungen üblichen Aufruf von eingebauten Funktionen, von der vorangestellten Klasse `Math` abgesehen.

Abs

Die Verwendung bereitet sicher wenig Schwierigkeiten. In der Praxis sollten Sie sich gelegentlich an einige Methoden erinnern, die trickreich zur Problemlösung eingesetzt werden können. So werden bei der Ausgabe langer Listen manchmal Aufteilungen in Blöcke oder Seiten benötigt. Wenn Sie 21 Elemente haben und jede Liste 5 Elemente lang sein darf, besteht die Fragestellung darin, die Anzahl der Listen zu ermitteln. Die Antwort ist 5 – die ersten 4 Listen enthalten 5 Elemente und die letzte 1. Eine einfache Division führt nicht zum Ziel: $21/5 = 4.2$. Runden funktioniert also nicht. Sie benötigen eine Methode, die immer aufrundet: `Ceiling`. Manchmal wird auch ein generelles Abrunden benötigt: `Floor`.

**Ceiling
Floor**

Mathematisch korrekt runden Sie dagegen mit `Round`, ein zweiter Parameter gibt die Anzahl der Kommastellen an. Potenzen werden mit `Pow` berechnet. Das Vorzeichen wird mit `Sign` ermittelt, wobei die Zahlen -1 , 0 oder 1 zurückgegeben werden, nicht `True` oder `False`.

**Round
Pow
Sign**

Wenn Sie andere Programmiersprachen kennen, sollten Sie wissen, dass `Min` und `Max` nur zwei Parameter ermöglichen, nicht unbegrenzt viele.

**Min
Max**

4.5 Zufallszahlen

Zufallszahlen sind immer wieder ein Thema in der Programmierung. Im Namensraum `System` gibt es die Klasse `Random`, die das Erzeugen derartiger Nummern sehr komfortabel gestaltet.

4.5.1 Die Klasse `Random`

Zufallszahlen Die Mitglieder der Klasse `Random` sind nicht statisch, Sie müssen deshalb eine Instanz der Klasse erzeugen:

```
Random rnd = New Random()
```

`rnd` ist nun das Objekt, das Zufallszahlen aller Art liefern kann. Verwenden Sie folgende Methoden:

► `Next`

Hiermit wird eine Zufallszahl erzeugt. Ohne Parameter im gesamten Wertebereich `Int32`, mit einem Parameter zwischen 0 und dem angegebenen Wert, mit zwei Parametern zwischen diesen beiden.

► `NextDouble`

Diese Methode erzeugt eine Gleitkommazahl vom Typ `Double` größer 0.0 und kleiner als 1.0.

► `NextBytes`

Die Methode füllt ein beliebig großes Arrays aus Bytes mit Bytewerten (0 bis 255).

Die Verwendung aller Methoden zeigt das folgende Beispiel:

```
Private Sub Page_Load()
    Dim rnd As Random = New Random()
    Dim b() As Byte = New Byte(10) {}
    rnd.NextBytes(b)
    Dim i As Integer
    For i = 0 To 10 - 1 Step i + 1
        ausgabe.InnerHtml += i.ToString() + ":" + " "
        + b(i).ToString() + "<br/>"
    Next
    ausgabe.InnerHtml += "<p/>"
    ausgabe.InnerHtml += rnd.NextDouble().ToString()
    ausgabe.InnerHtml += "<p/>"
```



```

    ausgabe.InnerHtml += rnd.Next(1000, 99999).ToString()
    ausgabe.InnerHtml += "<p/>"
    ausgabe.InnerHtml += rnd.Next().ToString()
End Sub

```

Listing 4.18: Erzeugen von verschiedenen Zufallszahlen (Ausschnitt aus Randoms.aspx; der knappe Code wurde nicht in eine eigene Code-Behind-Datei ausgelagert)

Hier wird ein leeres Array für 10 Bytes erzeugt:

**Wie es
funktioniert**

```
Dim b() As Byte = New Byte(10) {}
```

Das zuvor erzeugte Random-Objekt *rnd* füllt dies nun mit einer zufälligen Folge:

```
rnd.NextBytes(b)
```

Der Rest des Programm besorgt die Ausgabe, die zeigt, wie die Methoden reagieren:



Abbildung 4.17: Ausgabe verschiedener Zufallszahlen

Wie üblich bei Zufallszahlen basiert der Algorithmus auf einem Startwert. Die Zuweisung erfolgt mit der Instanziierung. Der Konstruktor verwendet einen Zeitstempel, wenn kein Startwert angegeben wurde. Alternativ kann auch ein Startwert vorgegeben werden:

Startwerte

```
Dim rnd As Random = New Random(12345)
```

Dann beginnt die Zufallsfolge immer wieder an der gleichen Stelle.

4.6 Zeichenketten

Der Verarbeitung von Zeichenketten kommt generell eine große Bedeutung zu, denn die Datenströme zwischen Webserver und Browser sind letztlich Folgen von Zeichen.

4.6.1 Verarbeitung von Zeichenketten

Operationen mit Zeichenketten nutzen Eigenschaften und Methoden der Klasse `System.String`. In ASP.NET wird `System` automatisch eingebunden – dieser Namensraum enthält die Klasse. In externen Code-Dateien deklarieren Sie ihn folgendermaßen:

```
Imports System
```



Auch hier sei es nochmals angemerkt: Visual Studio .NET erledigt das automatisch für Sie. Nur wenn Sie mit anderen Entwicklungsumgebungen arbeiten, ist die explizite Deklaration in VB.NET erforderlich.

Eigenschaften

Die Zeichen einer Zeichenkette

Die Klasse `String` verfügt über zwei Eigenschaften, `Chars` und `Length`. Mit `Chars` erhalten Sie Zugriff auf einzelne Zeichen der Zeichenkette. Beachten Sie bei der Angabe des Indizes, dass dieser bei 0 beginnt. `Length` ermittelt die Länge der Zeichenkette.

```
<script runat="server" language="vb">
Private Sub Page_Load()
    Dim zk As String = "Ich bin eine Zeichenkette"
    ausgabe.InnerHtml = "Ausgabe: " + zk
    ausgabe.InnerHtml += "<br/>Zeichen 7 per Indexer: " &
        + zk.Chars(6)
    ausgabe.InnerHtml += "<br/>Länge: " + zk.Length.ToString()
End Sub</script>
<html>
<head>
    <title>Zeichenketten</title>
</head>
<body>
    <h1>Umgang mit Zeichen und Zeichenketten</h1>
    <p id="ausgabe" runat="server"/>
</body>
</html>
```

Listing 4.19: Zeichenketten-Eigenschaften `Chars` und `Lengths`
(*StringCharacterLength.aspx, ohne Code-Behind*)

Die Ausgabe wird hier in das HTML-Serversteuerelement *ausgabe* geschrieben. Der siebte Buchstabe wird folgendermaßen adressiert:

```
zk.Chars(6)
```

Die Länge wird durch Anwendung der Eigenschaft `Length` ermittelt und für die Ausgabe mit `ToString` in eine Zeichenkette konvertiert:

```
zk.Length.ToString()
```

Length

*Beachten Sie, dass die Eigenschaft `Chars` ein »Zeichen« zurückgibt. VB.NET unterscheidet bei *typstrenger Notation* (*Option Strict*) zwischen einem einzelnen Zeichen und einer Zeichenkette mit einem Zeichen.*



Der Unterschied fällt auf, wenn Sie aus einer Zeichenkette, die aus Ziffern besteht, ein Zeichen entnehmen und als Ziffer verwenden möchten. Nahe liegend wäre folgende Schreibweise:

```
Dim s As String = "123456"  
Dim i As Integer = Convert.ToInt32(s.Chars(1))
```

Erwartet wird eigentlich, dass in `i` jetzt die Zahl 2 ist. Tatsächlich steht dort 50 – der Unicode (und auch ASCII- bzw. ANSI-Code in diesem Fall) des Zeichens »2«.

Für eine derartige Umwandlung muss also sicher gestellt werden, dass eine Zeichenkette vorliegt:

```
Dim s As String = "123456"  
Dim i As Integer = Convert.ToInt32(s.Chars(1)).ToString()
```

Freilich stellt sich hier die Frage, ob man so umständlich auf Zeichen einer Zeichenkette zugreifen muss. Ein Blick auf die Zeichenketten-Methoden lohnt deshalb.

Methoden für Zeichenketten

Es gibt sehr viele Methoden, die wichtigsten werden hier vorgestellt. Häufig ist der Zugriff auf Teile einer Zeichenkette notwendig. Dann verwenden Sie `Substring`. Diese Methode verlangt zwei Parameter; den Startwert und die Anzahl der Zeichen, die herausgenommen werden sollen. Bei der Angabe der Indizes in `Substring` müssen Sie wieder darauf achten, dass die Zählung der

**Substring
IndexOf
LastIndexOf**

Zeichen intern mit 0 beginnt. Das erste Auftreten eines bestimmten Zeichens wird mit `IndexOf` ermittelt, `LastIndexOf` stellt das letzte Vorkommen des Zeichens fest.

ToCharArray Auf alle Zeichen kann auch mit `ToCharArray` zugegriffen werden – diese Methode erzeugt aus den Zeichen ein Array mit Elementen vom Typ *Char*. Die Definition erfolgt folgendermaßen:

```
Dim azk() As Char
```

Die Methode verlangt die Angabe des ersten und letzten Zeichens, das kopiert werden soll. Im Beispiel wird die gesamte Zeichenkette verwendet, das erste Zeichen hat den Index 0, das letzte wird mit `Length` bestimmt:

```
azk = zk.ToCharArray(0, zk.Length)
```

Die Ausgabe mit *For Each* dient nur der Demonstration:

```
<script runat="server" language="vb">
Private Sub Page_Load()
    Dim azk() As Char
    Dim zk As String = "Ich bin eine Zeichenkette"
    ausgabe.InnerHtml = "Ausgabe:"
    ausgabe.InnerHtml += "<br/>Teil mit 'Substring': " +
        zk.Substring(4, 3)
    ausgabe.InnerHtml += "<br/>Erstes 'n': " +
        zk.IndexOf("n")
    ausgabe.InnerHtml += "<br/>Letztes 'n': " +
        zk.LastIndexOf("n")
    ausgabe.InnerHtml += "<br/>"
    azk = zk.ToCharArray(0, zk.Length)
    Dim c As Char
    For Each c In azk
        ausgabe.InnerHtml += "&nbsp;" + c.ToString()
    Next
End Sub
</script>
<html>
<head>
<title>Zeichenketten</title>
</head>
<body>
<h1>Umgang mit Zeichen und Zeichenketten</h1>
<p id="ausgabe" runat="server"/>
</body>
</html>
```

Listing 4.20: Verwendung von Zeichenketten-Methoden
(*StringMethods1.aspx*, ohne *Code-Behind-Datei*)

Im Beispiel werden die zuvor beschriebenen Methoden verwendet. Die Ausgabe der Ergebnisse erfolgt in ein HTML Server-Steuerelement, dass mit `<p runat="server">` entsteht, durch verändern der Eigenschaft `InnerHTML`.

Umgang mit Zeichen und Zeichenketten

```
Ausgabe:
Teil mit 'Substring': bin
Erstes 'n': 6
Letztes 'n': 19
Ich bin eine Zeichenkette
```

Abbildung 4.18: Wirkungsweise der Zeichenketten-Methoden

Die Umwandlung in Arrays und die Rückumwandlung ist ebenfalls eine sehr oft benötigte Funktion. Nutzen Sie `Split`, um eine Zeichenkette an einem definierten Zeichen zu zerlegen. Die Fragmente werden in einem Array aus Zeichenketten abgelegt. Damit mehr als ein Trennzeichen verwendet werden kann, akzeptiert die Methode ein Array aus Zeichen. Ein zweiter Parameter begrenzt die Anzahl der Teile; dieser Parameter ist optional, ohne Angabe werden alle Teile zurückgegeben.

Split

Die Umkehrfunktion dazu ist `Join`. Als neues Trennzeichen kann nur eine Zeichenkette angegeben werden. Zwei optionale Parameter bestimmen den ersten Index und die Anzahl der Elemente des verwendeten Arrays.

Join

```
Private Sub Page_Load()
    Dim astr() As String
    Dim ctrenn() As Char = {"c", ",", "c"}

    Dim zk As String = "Ich bin eine Zeichenkette,mit Komma"
    ausgabe.InnerHtml = "Ausgabe:<br/>"
    astr = zk.Split(ctrenn, zk.Length)
    Dim str As String
    For Each str In astr
        ausgabe.InnerHtml += str + "<br/>"
    Next
    Dim strn As String = String.Join("|",astr)
    ausgabe.InnerHtml += strn
End Sub
```

Listing 4.21: Trennen und Zusammensetzen von Zeichenketten
(Ausschnitt aus `StringMethod2.aspx`)

Die Trennung erfolgt hier an Leerzeichen und Kommata. Die dafür notwendige Definition eines Zeichenarrays kann folgendermaßen aussehen (beachten Sie die Zeichen-Literale »c«):

```
Dim ctrenn() As Char = {" ", "c", ",", "c"}
```

Die Aufteilung der Zeichenkette passiert in dieser Zeile:

```
astr = zk.Split(ctrenn, zk.Length)
```

Das Zusammensetzen mit neuen Trennzeichen erledigt Join:

```
Dim strn As String = String.Join("|", astr)
```

Beachten Sie hier, dass Join eine statische Methode ist, die nicht auf eine Zeichenketteninstanz angewendet werden kann. Schreiben Sie stattdessen den Klassennamen davor: String.Join.



Abbildung 4.19: Teilen und Zusammenfügen von Zeichenketten

Leerzeichen entfernen

Bei der Annahme von Werten – beispielsweise aus Formularen – sind auch Umwandlungen nötig. Lästige Leerzeichen lassen sich mit Trim (am Anfang und am Ende), bzw. mit TrimStart und TrimEnd entfernen. Wenn Sie mehr als nur Leerzeichen entfernen möchten, geben Sie als Parameter ein Array aus Zeichen an, die entfernt werden sollen. Umwandlungen in Kleinbuchstaben werden mit ToLower vorgenommen, in Großbuchstaben mit ToUpper.

Formatierungen

Wichtig sind auch Formatierungen von Zahlen, die als Zeichenkette dargestellt werden sollen. Dazu dient in erster Linie die statische Methode Format. Als Parameter sind mindestens eine Formatzeichenkette und das zu formatierende Objekt anzugeben. Die Formatzeichenkette enthält Formatierungsanweisungen, die diesem Aufbau folgen:

```
{N[,M][:F[d]]}
```

Die Optionen `M` und `F` sind optional (durch die eckigen Klammern angedeutet, die nicht mit geschrieben werden. `N` ist eine Nummer, die einen aus der Liste der zu formatierenden Werte auswählt. `F` qualifiziert die Formatierung; die möglichen Werte sind der folgenden Tabelle zu entnehmen. `d` legt bei numerischen Werten die Anzahl der Nachkommastellen oder bei Ganzzahlen die der Zeichenzahl insgesamt fest. Auch diese Angabe ist optional. `M` bestimmt ein Füllzeichen, hier sollte bei Zahlen die »0« stehen.

Formatsymbol	Bedeutung
C	Währung
D	Dezimalzahl
E	Exponentialzahl
F	Gleitkommazahl (Float)
N	Numerisch
P	Prozentwert
R	Erzeugter Wert wird so dargestellt, dass eine Rückkonvertierung möglich ist
X	Hexadezimalzahl

Tabelle 4.1: Formatierungsoptionen von Zahlen als Zeichenketten (*F*)

Der Formatwert kann groß oder klein geschrieben werden. Nachkommastellen, die nicht dargestellt werden können, werden korrekt gerundet. Alle Zeichen der Formatierungszeichenkette, die nicht in geschweiften Klammern stehen, werden unverändert ausgegeben. Sollen dennoch Zeichenketten in den Parametern mit aufgenommen werden, schreiben Sie nur die Nummer des Parameters: `{3}`. Die Formatierung von Zahlen zeigt das folgende Beispiel:

```
Private Sub Page_Load()  
    Dim zk As String = "Jörg Krause, Berlin"  
    ausgabe.InnerHtml = "Ausgabe:<br/>"  
    ausgabe.InnerHtml += zk.ToUpper()  
    Dim zahl As Double = 12352.4565  
    Dim i As Double = 0.46  
    ausgabe.InnerHtml += "<br/>"  
    + String.Format("{0:N2}, {1:P0}", zahl, i)  
End Sub
```

Listing 4.22: Verwendung von Umwandlungs- und Formatierungsmethoden
(Ausschnitt aus *StringMethods3.aspx*)

Die Ausgabe zeigt die Wirkung auf die einzelnen Werte:

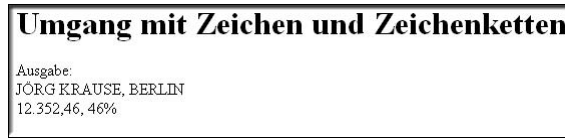


Abbildung 4.20: Formatierung von Zahlenwerten und Zeichenketten

Beispiel Die Zeichenfolge {0,0:D5} dient der Erstellung einer fünfstelligen Zahl, beispielsweise aus einem Integer-Wert. Die Zahl 17 wird dann zu »00017«. Die erste 0 weist darauf hin, dass der erste Parameter verwendet wird.



In diesem Abschnitt wird nur auf die Formatierung von Zahlen und Zeichenketten eingegangen. Wenn als Parameter andere Datentypen eingesetzt werden, haben die Formatierungszeichen eine grundlegend andere Bedeutung. In den entsprechenden Abschnitten wird darauf explizit hingewiesen.

Replace Ersetzungen nehmen Sie mit `Replace` vor:

```
Private Sub Page_Load()  
    Dim zk As String = "Jörg Krause, München"  
    ausgabe.InnerHtml = "Ausgabe:<br/>"  
    ausgabe.InnerHtml += zk.Replace("München", "Berlin")  
End Sub
```

Listing 4.23: Ersetzungen in Zeichenketten (Ausschnitt aus `StringMethods4.aspx`)

Die Methode ersetzt alle Vorkommen des ersten Parameters durch den zweiten. Als Datentypen sind `String` oder `Char` erlaubt, beide Parameter müssen aber vom gleichen Typ sein.

Weitere Methoden

Die Möglichkeiten, die die Zeichenkettenfunktionen bieten, sind sehr umfangreich. Eine detaillierte Darstellung sprengt den Rahmen dieses Buches. Nutzen Sie die folgende Liste als Stichwortquelle für die Online-Referenz:

► `StartsWith`, `EndsWith`

Diese Methoden geben `true` zurück, wenn eine Zeichenkette mit bestimmten Zeichen beginnt bzw. endet.

► `PadLeft`, `PadRight`

Mit diesen Methoden kann eine Zeichenkette bis zu einer bestimmten Anzahl Zeichen aufgefüllt werden.

► `IndexOfAny, LastIndexOfAny`

Analog zu den `Index`-Methoden wird das Auftreten bestimmter Zeichen festgestellt, als Eingabewert kann ein Array aus Zeichen genutzt werden.

► `CopyTo`

Diese Methode kopiert eine Anzahl Zeichen an eine bestimmte Position eines Arrays aus Zeichen.

► `Concat`

Dies ist eine statische Methode, die der Verknüpfung von Zeichenketten in Situationen dient, in denen der Verknüpfungsoperator `+` nicht angebracht ist.

Abschließend sei noch auf den statischen Feldwert `Empty` hingewiesen, der eine leere Zeichenkette erzeugt:

```
String leer = String.Empty
```

4.6.2 Zeichenketten als Ausgabeziel

Viele Klassen im .NET-Framework bieten Ausgabemethoden, die an Instanzen der abstrakten Klasse `TextWriter` gebunden sind. `TextWriter` gehört zum Namensraum `System.IO`. Sie finden hier auch Klassen zum Dateisystemzugriff. Für den Webprogrammierer sind mehrere Implementierungen von Bedeutung:

Implementierungen der Klasse `TextWriter`

► `System.Web.HttpWriter`

Diese Klasse dient der Ausgabe von Zeichen über `HttpResponse`. Alle Ausgaben zum Browser nehmen diesen Weg, meist jedoch intern, sodass der direkte Aufruf selten in Frage kommt.

► `System.Web.UI.HtmlTextWriter`

Diese Klasse dient der formatierten Ausgabe von HTML in einer `WebForm`. Steuerelemente können diese verwenden, um HTML zu schreiben.

► `System.IO.StreamWriter`

Mit dieser Klasse werden allgemeine Datenströme kontrolliert, beispielsweise solche zu einer Datei, einem Gerät usw. Der Ausgabedatenstrom erfolgt hier byteweise.

Das Schreiben und Lesen von Daten erfolgt mit verschiedenen Methoden, je nachdem, um welche Art von Daten es sich handelt. Wie beim `StreamWriter` angedeutet, kann es sich dabei um Bytes

handeln. Früher gab es bei der Behandlung von Daten keinen Unterschied zwischen Text und Bytes. Bytes stellten die Zeichen mit den Codes 0 bis 255 dar (1 Byte enthält 8 Bits, 8 gesetzte Binärstellen entsprechen dezimal 255). Text bestand meist aus ASCII, einer Codiertabelle, die den Zahlenwerten Zeichen zuordnet.

Im Zuge der Internationalisierung entstand Bedarf an mehr Zeichen, als die Darstellung mit 1 Byte ermöglichte. Eine anerkannte Norm für breitere Zeichendarstellungen ist Unicode. Wenn aber ein Textzeichen mehr als 1 Byte umfasst, taugt das byteweise Schreiben von Daten nicht mehr, denn es würde sehr schnell passieren, dass der Text »mitten im Zeichen« abbricht. Deshalb sind spezialisierte Klassen für die Zeichenkettenverarbeitung notwendig. In .NET erfolgt die Zeichenkettenverarbeitung mit der Klasse `StringWriter`.

Die Klassen `StringWriter` und `StringBuilder` im Detail

Namensraum `StringWriter` ist eine Implementierung der abstrakten Klasse `TextWriter`. Letztere können Sie nicht direkt verwenden; Sie müssten zuvor eine eigene Implementierung vornehmen. Für die meisten Fälle genügt jedoch `StringWriter`. Im Code müssen Sie den passenden Namensraum spezifizieren:

```
Imports System.IO
```

Ergänzend ist für den Zugriff auf die Codierungs-Aufzählungen noch `System.Text` zu empfehlen:

```
Imports System.Text
```



Diese Namensräume importiert auch Visual Studio .NET nicht automatisch für Sie. Die explizite Deklaration in VB.NET ist immer erforderlich.

StringBuilder In `System.Text` wird auch `StringBuilder` definiert. Die Instanzen der Klasse speichern die Zeichendaten nicht direkt, sondern in Objekten vom Typ `StringBuilder`. Daraus ergeben sich Grenzen der Verwendung. Die Standardkapazität beträgt 16 Zeichen, die maximale Anzahl Zeichen entspricht `Int32.MaxValue` (konkret: 2.147.483.647). Sie können also mit `StringWriter` nicht mehr als 2 GByte Daten pro Objekt speichern. Eine neues Zeichenkettenobjekt wird nun folgendermaßen erzeugt:

```
Dim sb As StringBuilder = New StringBuilder()
```

Optional können dem Konstruktor Parameter übergeben werden, um die Anfangsgröße von vornherein festzulegen, zu begrenzen oder einen Startwert zu erzeugen.

Wenn von einer Zeichenspezifischen Codierung die Rede ist, sind auch verschiedene Codierv Verfahren zu betrachten. Auch wenn Unicode eine zentrale Rolle auf Windows-Systemen spielt, sind aufgrund der notwendigen Interoperabilität Umwandlungen in andere Systeme notwendig. Die Klasse `StringWriter` stellt die Eigenschaft `Encoding` bereit, die Angaben zur aktuellen Codierung enthält.

Codierung

Die Welt der Zeichencodierung

Zeichen werden aus verschiedenen Wegen codiert. Das älteste System heißt ASCII (American Standardcode for Information Interchange). ASCII entstand 1968 als ANSI X3.4 Norm, 1972 wurde daraus der ISO-Standard ISO 646. Es definiert lediglich 127 Zeichen und ist 7 Bit breit. Ursprünglich wurde das achte Bit als Paritätsbit betrachtet, um Fehler bei der Übertragung zu erkennen. Das reichte jedoch nicht lange aus, denn viele Sprachen verwenden – im Gegensatz zum Englischen – eine ganze Reihe von Sonderzeichen. Im Deutschen sind dies beispielsweise die Umlaute. Deshalb wurde der ASCII-Zeichensatz erweitert auf 255 Zeichen, wobei das achte Bit nun der Codierung dient. Allerdings beschränkt sich auch diese Form auf Sprachen mit wenigen zusätzlichen Zeichen und lässt andere wie Kyrillisch, Griechisch oder asiatische Zeichen außen vor.

Aus der für internationale Zwecke völlig untauglichen 7-Bit-Version entstand die 8-Bit-Erweiterung, die als ISO-8859-1 weite Verbreitung fand und auch heute noch im Einsatz ist. Der Codename für ISO-8859-1 lautet Latin1. Definiert wird die Bedeutung der Zeichen 128 (`\xA0`) bis 255 (`\xFF`). Damit werden die Sonderzeichen der Alphabete von westeuropäischen Sprachen abgebildet, was unter anderem auch für das Deutsche zutrifft. Enthalten sind daneben auch Codierungen für Währungssymbole und einige Sonderzeichen.

Für osteuropäische Sprachen, die auf lateinischen Zeichen basieren, wird ISO-8859-2 (Latin2) angeboten, wo dieselben Zahlenwerte andere Zeichen codieren. Dies betrifft Sprachen wie Ungarisch, Rumänisch oder Slowakisch. Die deutschen Umlaute sind übrigens neben Latin1 auch in Latin2 bis Latin6 zu finden, sodass man bei geschickter Wahl der Zeichensätze mehrsprachig schreiben kann. Wer Deutsch und Polnisch schreibt, nutzt Latin2 statt Latin1. Weitere Sätze sind ISO-8859-3 (Latin3, Esperanto und Maltesisch), ISO-8859-4 (Latin4, Baltische Sprachen, Grönländisch, Lappisch, ISO-8859-5 (Cyrillic, Kyrillisch einschließlich aller Variationen), ISO-8859-6 (Arabic, Arabisch), ISO-8859-7 (Greek, Griechisch) und ISO-8859-8 (Hebrew, Hebräisch). Es gibt neue Varianten, die lateinische Sprachen modifizieren (ISO-8859-5 und ISO-8859-6) und Erweiterungen für asiatische Sprachen (ISO-8859-11 für Thai).

Es wird schnell klar, dass solche Systeme untauglich sind, mehrsprachige Dokumente zu codieren. Deshalb entstand Unicode. Unicode verwendet 16 Bit zur Zeichendarstellung und erlaubt damit 65.535 Zeichen. Davon werden 63.486 tatsächlich für Zeichen verwendet, während die übrigen mit weiteren 16-Bit-Werten Paare bilden, aus denen maximal 1.048.544 Zeichen geformt werden können. Das ist für alle Sprachen der Welt ausreichend, auch für Japanisch oder Chinesisch, wo allein Tausende Zeichen codiert werden müssen. Für exotische Ansprüche gibt es außerdem noch eine genormte 20-Bit-Erweiterung, die das Zeichenzahlproblem wohl endgültig beseitigt. Der offizielle Name für Unicode lautet Universal Character Set (UCS). Die Normungsnummer lautet ISO 10646. Das ist eine programmatische Nummer, 646 (ASCII, siehe oben) plus 10.000. Die Akzeptanz ist sehr weit reichend, so definiert RFC 2070 als Basiszeichensatz für HTML ISO 10646. Und spätestens hier wird klar, dass Unicode für Webprogrammierer eine Rolle spielt.

Unicode 1.0 entstand 1991. Es folgten mehrere Versionen, wobei 3.0 die aktuelle darstellt. Neue Versionen codierten vor allem neue Zeichen aus Sprachen, wie Khmer, Äthiopisch oder die Braille-Schrift für Blinde. Die Unicode-Tabellen enthalten neben dem Zeichenwert auch Namen der Zeichen, beispiels-

weise codiert U+0041 den lateinischen Buchstaben »A«, dessen Name LATIN CAPITAL LETTER A lautet. Etwas neuer ist EURO-CURRENCY SIGN, U+20A0 oder »€«. Unterschieden werden eindeutig zugeordnete und vereinheitlichte Zeichen. So kann das große A in vielen Sprachen auftreten, während andere Zeichen eindeutig genau einer Sprache zugeordnet sind. Unicode codiert jedoch Schriften, nicht Sprachen. Es gilt jedoch eine Eindeutigkeitsregel, die die Darstellungsform beachtet. So entspricht in der Schrift das große A dem großen Alpha im Griechischen. Dennoch wird Letzteres als Teil einer anderen Schrift anders codiert (GREEK CAPITAL LETTER ALPHA, U+0391). Diakritische Zeichen werden sowohl direkt (U+00FC, LATIN SMALL LETTER U WITH DIAERESIS für »ü«) also auch mit Kombinationszeichen codiert (U+0075, LATIN SMALL LETTER U gefolgt von U+0308 COMBINING DIAERESIS, führt ebenfalls zu »ü«). Circa 475 lateinische Buchstaben sind vorbereitet, das reicht aber für Sprachen wie Vietnamesisch nicht aus, weshalb die Kombinationsmöglichkeit bestehen bleibt.

Unicode enthält trotz seiner Mächtigkeit per Definition keine persönlichen Symbole und Logos. Allerdings ist der Dingbats-Zeichensatz enthalten (U+2700..U+27BF); dagegen fehlt beispielsweise das Apple-Symbol vom Macintosh.

Die flexible Breite von Unicode erleichtert nicht unbedingt die Verarbeitung. Nachwievor besteht ein hoher Bedarf an Zeichensätzen, die auf 8- oder 16-Bit-Zeichen beschränkt sind. Deshalb wurde eine Transformation in ein 8-Bit-Format entwickelt – UTF-8 (Unicode Transformation Format-8). Vor allem bei der Übertragung von Zeichen über Protokolle, die auf Byte-Formate zugeschnitten sind, hat UTF-8 eine große Bedeutung. Lässt die Umgebung eine Verarbeitung von 16-Bit-Werten zu, wird UTF-16 eingesetzt. Das UTF-System ist darüber hinaus generell selbstsynchronisierend, das heißt, die Byte-Folgen sind so kodiert, dass das lesende System erkennen kann, wenn es »mitten in einem Zeichen« beginnt und dann durch Rücksprünge in der Bytefolge den Anfang erkennen kann. Für Internet-Anwendungen ist noch UTF-7 wichtig, die Transformation von Unicode auf einen 7-Bit-Zeichensatz, der 7-Bit-ASCII entspricht. Von diesem Format machen

E-Mail-Programme häufig Gebrauch. Die einzige Abweichung betrifft das »+«-Zeichen, das in UTF-7 eine besondere Bedeutung hat.

Abschließend sei noch festgestellt, dass Unicode keine Informationen über Schriftarten, die Form der Darstellung oder Schreibrichtungen enthält.

Encoding Mit `Encoding` wird die Codierung eines Textes festgelegt oder geändert. Verfügbar sind folgende Einstellungen für `StringWriter`:

- ▶ `Encoding.ASCII`
- ▶ `Encoding.Unicode`
- ▶ `Encoding.UTF7`
- ▶ `Encoding.UTF8`

Der `StringBuilder` selbst arbeitet unabhängig von einer Codierung, wobei intern Unicode zum Einsatz kommt, damit keine Daten verloren gehen. Erst bei der Ausgabe wird der entsprechende Zeichensatz benutzt, um dem Zielformat zu entsprechen. Unabhängig davon ist jedoch `StringBuilder` ein effizienter Weg zum Umgang mit Zeichenfolgen.

Die Klasse `StringBuilder` effizient einsetzen

Ein neues Zeichenkettenobjekt wird mit `StringBuilder` folgendermaßen erzeugt:

```
Dim sb As StringBuilder = New StringBuilder()
```

Wenn Sie eine neue Zeichenkette an eine vorhandene anhängen möchten, gehen Sie folgendermaßen vor:

```
sb.Append("Zeichenkette")
```

Es gibt verschiedene Überladungen von `Append`, um beliebige Datentypen als Datenquelle verwenden zu können. So funktioniert auch Folgendes:

```
sb.Append(23)
```

Methoden Der Aufruf von `ToString` ist nicht notwendig. Wenn formatierte Werte übergeben werden sollen, wird `AppendFormat` eingesetzt.

Änderungen an der Zeichenkette werden mit den Methoden `Insert`, `Remove` und `Replace` vorgenommen. `Replace` ersetzt Zeichen oder Zeichenfolgen ab einem bestimmten Index und optional für eine bestimmte Anzahl von Vorkommen. `Insert` fügt Zeichen an einer bestimmten Position ein, `Remove` entfernt eine Anzahl Zeichen, ohne dass Lücken zurückbleiben.

Wenn Sie an irgendeiner Stelle eine `TextWriter`-Instanz benötigen, können Sie fast immer statt deren allgemeiner Erscheinungsform die Klasse `StringWriter` verwenden. Dann werden die Daten in dieses Objekt hineingeschrieben. Als Basis für die Daten kann ein `StringBuilder`-Objekt verwendet werden:

StringWriter

```
Dim sw As StringWriter = New StringWriter(sb)
```

Mit der Eigenschaft `Capacity` ermitteln Sie die aktuelle Kapazität bzw. legen diese fest. Die tatsächliche Anzahl Zeichen wird dagegen mit `Length` ermittelt.

Eigenschaften

Zeichenketten lesend verarbeiten

Wenn Sie Zeichen nicht selbst erzeugen, sondern von einem Gerät oder Datenstrom empfangen, müssen Sie diese irgendwo aufnehmen. Dann kommt das Gegenstück zum `StringWriter`, der `StringReader` zum Einsatz. `StringReader` kann jedoch nicht mit `StringBuilder` kombiniert werden, da er entweder Zeichencodes (Typ `int`) oder Arrays aus Zeichen (Typ `Char()`) zurückgibt.

4.7 Reguläre Ausdrücke

Reguläre Ausdrücke begegnen dem Programmierer immer wieder. Anfänger empfinden die kryptischen Schlangen voller Sonderzeichen als unlesbar und auch mancher Profi schreibt lieber schnell ein paar Schleifen, als sich Gedanken über ein solches Muster zu machen. Dabei sparen reguläre Ausdrücke viele Zeilen Code und lösen manches Problem auf verblüffend elegante Weise. Dieser Abschnitt führt Sie in die Welt der regulären Ausdrücke ein und zeigt, wie Sie sich ein unglaublich spannendes Kapitel der Programmierkunst erschließen.

Einführung

Um die Abhängigkeiten der nachfolgend beschriebenen Klassen verstehen zu können, machen Sie sich mit dem folgenden Diagramm vertraut:

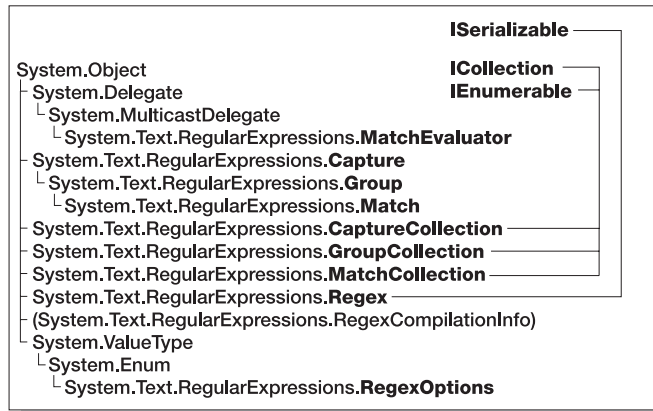


Abbildung 4.21: Abhängigkeiten der Klassen für reguläre Ausdrücke im Framework

4.7.1 Überall reguläre Ausdrücke

Namensraum

Das .NET-Framework verfügt auch über Klassen, die mit regulären Ausdrücken umgehen. Sie sind im Namensraum `System.Text.RegularExpressions` zusammengefasst.

Was sind reguläre Ausdrücke?

Reguläre Ausdrücke (aus dem Englischen: regular expressions) beschreiben Suchmuster. Man kann damit nach Zeichen in einer Zeichenkette oder einem Text suchen – nicht mehr, aber auch nicht weniger. Das geht freilich mit einfachen Suchfunktionen auch. Oft fehlt es aber an einer klaren Definition, was eigentlich gesucht werden soll. Wenn Sie in einem längeren Text nach Stellen suchen, die Währungsangaben beinhalten, dann hilft eine normale Suchfunktion nicht. Viele Suchfunktionen in Editoren, Textverarbeitungen und anderen Programmen, die Textstellen suchen (wie beispielsweise Visual Studio .NET), nutzen deshalb auch reguläre Ausdrücke. Diese erlauben auch die Suche nach allgemeineren Angaben. Die Möglichkeiten sind dabei aber so umfangreich, dass es sich schon wieder lohnt, ganze Bücher darüber zu schreiben. Die Vielfalt der Funktionen und die Mächtigkeit der Definitionen führt dann auch zu diesen unlesbaren, verwirrenden Konstrukten.

Das Programm in einer Zeile

In einem regulären Ausdruck steckt, wenn man es genau betrachtet, ein komplettes Programm. Wenn man dies akzeptiert und versteht, dann ist der Entwurf und die Nutzung nicht schwieriger als mit jedem anderen Stück Code auch. Sie müssen sich die Zeit nehmen, einen regulären Ausdruck sorgfältig zu entwerfen und zu testen. Niemand – auch Profis nicht – schreiben einen neuen Ausdruck, der vielleicht nur 10 oder 20 Zeichen umfasst, in ein paar Sekunden auf. Intern arbeitet auch nicht ein einfacher Mustervergleich, sondern die Regex-Maschine; ein Programm, das die Ausdrücke auflöst und ausführt.

Wie der Ausdruck intern arbeitet

Ein paar kleine Beispiele

Konkrete Beispiele sind der beste Weg, reguläre Ausdrücke zu verstehen.

Das folgende Muster sucht nach dem HTML-Tag `<h1>`, `<h2>` usw:

HTML-Tags suchen

```
<h\d>
```

Das ist nun wenig spektakulär, weil diese Aufgabe auch eine normale Suchfunktion erfüllen würde. So ein Suchmuster wird nun von links nach rechts abgearbeitet und »gleitet« dabei über den Text, der durchsucht werden soll. Wird eine Übereinstimmung gefunden, entsteht ein so genannter »Match« – ein Treffer. Im Beispiel stehen alle Zeichen für sich selbst, außer `\d` – dies ist ein Steuerzeichen, dass für ein Zahlzeichen steht. Dieser Ausdruck erkennt allerdings auch `<h9>`, was in HTML nicht vorkommt. Reguläre Ausdrücke arbeiten Zeichenweise. Jedes Zeichen kann aber unterschiedlich komplex definiert werden. Was hier benötigt wird, ist eine Zeichenklasse, die nur die Ziffern 1 bis 6 umfasst. Zeichenklassen werden in eckige Klammern gesetzt:

```
<h[123456]>
```

Die gesamte Klasse steht dennoch nur für ein Zeichen. Eleganter ist der folgende Ansatz:

```
<h[1-6]>
```

Innerhalb der Zeichenklasse können also Bereiche definiert werden. Mit Hilfe des `^`-Symbols kann der Inhalt der Zeichenklasse ausgeschlossen werden. Eine Alternative zur letzten Definition sähe dann so aus:

```
<h[^0789a-z]>
```

Es gibt bei regulären Ausdrücken oft viele Wege zum Ziel.

Häufig werden reguläre Ausdrücke eingesetzt, um Eingaben aus Formularen zu überprüfen. Die Prüfung einer E-Mail-Adresse ist dabei schon eine kleine Herausforderung. Mit einem regulären Ausdruck können Sie zumindest die grundsätzliche Form überprüfen.

E-Mail-Adresse Ein E-Mail-Name wird aus kleinen und großen Buchstaben aufgebaut. Als zusätzliche Zeichen sind das Minuszeichen »-«, der Unterstrich »_« und der Punkt ».« erlaubt. Groß- und Kleinschreibung spielt keine Rolle. Nach dem Namen folgt das @-Zeichen und die Domain. Hier dürfen keine Unterstriche auftreten, ansonsten entspricht es dem E-Mail-Namen. Es ist sinnvoll, den Ausdruck Schritt für Schritt zu entwickeln, wie es im Folgenden gezeigt wird:

► `^[_a-z0-9-]+`

Dieser Teil untersucht den E-Mail-Namen auf gültige Zeichen. Erlaubt sind neben Ziffern und Buchstaben noch Minuszeichen und Unterstriche. Außerdem soll der Ausdruck von Beginn an untersucht werden, was mit dem Symbol `^` erreicht wird.

► `^[_a-z0-9-]+(\.[_a-z0-9-]+)*`

Dieser Ausdruck akzeptiert auch den Punkt, allerdings nicht am Anfang oder Ende des Namens. Die Gruppierung erlaubt die Wiederholung des zweiten Teils, er kann aber auch entfallen. Damit sind Konstruktionen wie »joerg.krause« oder »joerg« oder »joerg_krause.vbnet« erlaubt.

Damit ist der Teil vor den @-Zeichen abgeschlossen.

► `([a-z0-9-]+\.[a-z]{2,4})$`

Hier wird nur der Domainname untersucht. Unterstriche sind nicht erlaubt, der Punkt ist zwingend und wird von zwei bis vier Zeichen gefolgt, die wiederum nur Buchstaben sein können. Hier stellt die Gruppierung sicher, dass der Domainname

aus mehreren Teilen bestehen darf. Damit sind Konstruktionen wie »domain.de«, aber auch »mail.rz.uni-magdeburg.de« erlaubt. Das abschließende `$`-Zeichen erzwingt, dass nach dem Domainnamen nichts mehr folgen darf – die zu untersuchende Zeichenkette muss hier enden.

Das Gebilde ist nun schon recht unübersichtlich. Fertig ist es in Kombination aus dem vorderen und dem hinteren Teil, verbunden durch das `@`-Zeichen:

```
^[_a-zA-Z0-9-]+(\\.[_a-zA-Z0-9-]+)*@
([a-zA-Z0-9-]+\\.){a-zA-Z}{2,4}$
```

Der Ausdruck ist keineswegs perfekt. Es gibt immer noch Fälle, die unzulässig sind und nicht erkannt werden. Aber er ist schnell und fängt die meisten Tippfehler oder bewusst unsinnige Angaben ab.

Programmierung regulärer Ausdrücke im Framework

Bevor eine tiefgehendere Betrachtung der regulären Ausdrücke folgt, soll eine praktische Nutzung im .NET-Framework gezeigt werden. Es gibt zwei Möglichkeiten, die Ausdrücke zu nutzen. Zum einen stehen die Klassen des Namensraumes `System.Text.RegularExpressions` bereit. Zum anderen gibt es ein Kontroll-Steuer-element mit dem Namen `RegularExpressionValidator`, das die Feldinhalte aus Formularen direkt prüft. Reguläre Ausdrücke erlauben nicht nur eine einfache Aussage der Form »passt« oder »passt nicht«. Innerhalb des Ausdrucks können Klammern zur Bildung von Gruppen genutzt werden. Die Untersuchung der E-Mail-Adresse liefert nicht nur eine Gültigkeitsprüfung, sondern auch gleich die Bestandteile Name, Domain und Toplevel. Dies ist mit dem Kontroll-Steuerelement nicht möglich.

```
<body>
<h1>Reguläre Ausdrücke</h1>
<form runat="server" ID="Form1">
    <asp:textbox runat="server" id="email"/>
    <asp:button runat="server" id="send" text="Prüfen"
        onclick="CheckEmail_OnClick"/>
</form>
<p id="ausgabe" runat="server"/>
</body>
```

Listing 4.24: Praktische Nutzung regulärer Ausdrücke (*RegexCheckEmail.aspx*)

Die Prüfung des Inhalts des Feldes *email* erfolgt mit einer in der Code-Datei hinterlegten Ereignisbehandlungsmethode, die folgendermaßen definiert wird:

```
Public Class RegexCheckMail
    Inherits System.Web.UI.Page
    Protected ausgabe As HtmlGenericControl
    Protected WithEvents send As System.Web.UI.WebControls.Button
    Protected email As TextBox

    Public Sub CheckEmail_OnClick(ByVal sender As Object,
                                   ByVal e As EventArgs) _
        Handles send.Click
        Dim rx As Regex
        Dim ma As Match
        Dim mc As CaptureCollection
        Dim rxmail As String = "^([_a-zA-Z0-9-]+)␣
                                   (\.[_a-zA-Z0-9-]+)*)(@)␣
                                   ((?>[_a-zA-Z0-9-]+\.)+␣
                                   ([a-zA-Z]{2,4}))$"

        rx = New Regex(rxmail, RegexOptions.IgnoreCase)
        ma = rx.Match(email.Text)
        If ma.Success Then
            ausgabe.InnerHtml = "<b>Die Adresse ist gültig</b><p/>"
            Dim i As Integer
            For i = 0 To ma.Groups.Count - 1 Step i + 1
                ausgabe.InnerHtml += "Treffer: <u>" + ␣
                                   ma.Groups(i).ToString()
                ausgabe.InnerHtml += "</u> an Position " + ␣
                                   ma.Groups(i).Index.ToString() + " = "
            mc = ma.Groups(i).Captures
            ausgabe.InnerHtml += mc.Count.ToString()
            If mc.Count > 0 Then
                Dim j As Integer
                For j = 0 To mc.Count - 1 Step j + 1
                    ausgabe.InnerHtml += "[" + mc(j).Value + "]"
                Next
            End If
            ausgabe.InnerHtml ␣
                                   += " (" + ma.Groups(i).Length.ToString() ␣
                                   + " Zeichen)<br />"
            Next
        Else
            ausgabe.InnerHtml = "<b>Die Adresse ist ungültig</b>"
        End If
    End Sub
End Class
```

Listing 4.25: Reguläre Ausdrücke verwenden (Ausschnitt aus *RegexCheckEmail.aspx.vb*)

Der Ausdruck in der Variablen `rxmail` ist hier gegenüber der vorgestellten Version noch um diverse Klammern erweitert worden. Runde Klammern dienen der Gruppierung, gehen also in die Analyse des Suchmusters selbst nicht ein, sondern steuern die Rückgabe von Teilen des Ausdrucks.

Reguläre Ausdrücke

Die Adresse ist gültig

Treffer: joerg@krause.net an Position 0 = 1[joerg@krause.net] (16 Zeichen)

Treffer: joerg an Position 0 = 1[joerg] (5 Zeichen)

Treffer: g an Position 4 = 5[j][o][e][r][g] (1 Zeichen)

Treffer: an Position 0 = 0 (0 Zeichen)

Treffer: @ an Position 5 = 1[@] (1 Zeichen)

Treffer: krause. an Position 6 = 1[krause.] (7 Zeichen)

Treffer: net an Position 13 = 1[net] (3 Zeichen)

Abbildung 4.22: Ergebnis einer erfolgreichen Prüfung mit regulärem Ausdruck

Die Analyse basiert auf einer Instanz der `Regex`-Klasse, die folgendermaßen erzeugt wird:

Wie es funktioniert

```
rx = New Regex(rxmail, RegexOptions.IgnoreCase)
```

Die Aufzählung `RegexOptions` kann mehrere Steuerungen des Ausdrucks vornehmen, `IgnoreCase` unterbindet die Unterscheidung von Groß- und Kleinschreibung. `rx` ist nun ein `Regex`-Objekt, das mit dem Ausdruck verbunden ist. »Gegen« diesen Ausdruck können nun Zeichenketten geprüft werden. Dazu dient die Methode `Match`, die ein Objekt der Klasse `Match` zurückgibt:

Regex

```
ma = rx.Match(email.Text)
```

In `ma` ist nun das `Match`-Objekt enthalten, der Ausdruck selbst wurde bereits abgearbeitet. `Match` ist sowohl eine Klasse, als auch eine Kollektion. Wenn das Suchmuster im durchsuchten Text mehrmals gefunden wird, enthält `Match` mehr als ein Element. Wenn die Suche allgemein erfolgreich war – also ein oder mehr Treffer vorhanden waren, ist die Eigenschaft `Success` gleich `True`. In diesem Fall wird der Ausdruck weiter untersucht:

Match

```
If ma.Success Then
```

Jedes Element der Kollektion `Match` enthält eine Eigenschaft `Groups`. Diese Eigenschaft gibt ein Objekt der Klasse `Group` zurück. Diese

Groups

Klasse erlaubt den Zugriff auf die durch runde Klammern gebildeten Gruppen. Eine wichtige Eigenschaft ist `Count` – die Anzahl der Gruppen.

For i = 0 To `ma.Groups.Count - 1` Step 1

Die Gruppen selbst bilden eine Kollektion. Der Zugriff erfolgt deshalb wieder in der üblichen Arrayschreibweise:

`ausgabe.InnerHtml += "Treffer: <u>" + ma.Groups(i).ToString()`

Jede Gruppe hat wiederum Eigenschaften. `Index` enthält die Position in der durchsuchten Zeichenkette, mit Null beginnend gezählt:

`ausgabe.InnerHtml += "</u> an Position " +
ma.Groups(i).Index.ToString()`

Die Eigenschaft `Length` enthält die Anzahl Zeichen, die in der Gruppe gefunden wurden:

`ausgabe.InnerHtml += " (" + ma.Groups(i).Length.ToString() +
" Zeichen)
"`

Alle Zugriffe auf die Ergebnisse regulärer Ausdrücke funktionieren auf Basis dieser Klassen.



Es gibt noch eine weitere Klasse, `Capture`, die ebenfalls Kollektionen bilden kann. Diese Klasse kann von `Group-Elementen` abgeleitet werden. Lesen Sie mehr dazu im Abschnitt 4.7.3, »Weitere Betrachtungen zu regulären Ausdrücken« ab Seite 278.

4.7.2 Grundlagen regulärer Ausdrücke

In der Praxis kommt es weniger auf die Programmierung, sondern auf die Entwicklung der benötigten Ausdrücke an. Nachfolgend werden reguläre Ausdrücke genauer vorgestellt. Zuerst die Zeichen zum Eingrenzen der Position des Suchmusters in der zu durchsuchenden Zeichenfolge:

Bedingung	Beschreibung
<code>^</code>	Beginn des Musters
<code>\$</code>	Ende des Musters
<code>\A</code>	Unbedingter Beginn auch bei mehrzeiligen Texten

Tabelle 4.2: Positionierungsbedingungen

Bedingung	Beschreibung
\Z	Unbedingtes Ende auch bei mehrzeiligen Texten oder vor dem Zeilenumbruchzeichen \n
\z	Unbedingtes Ende auch bei mehrzeiligen Texten

Tabelle 4.2: Positionierungsbedingungen (Forts.)

Es gibt Zeichen, die etwas anderes als sich selbst repräsentieren:

Zeichen	Bedeutung
\uNNNN	Unicode-Zeichen mit dem Code NNNN (Hexadezimale Angabe). ASCII-Zeichen nutzen das hintere Byte, ein Leerzeichen wird also als \u0020 geschrieben.
\n	Zeilenumbruch
\r	Wagenrücklauf
\0XXX	Oktale Angabe eines Zeichencodes mit ein bis drei Zahlen von 0 bis 7.
\xHH	Hexadezimale Angabe eines Zeichens
\\	Der Backslash selbst

Tabelle 4.3: Sonderzeichen in Zeichenklassen

Der Bildung von Zeichenklassen liegen fast alle regulären Ausdrücke zugrunde. Die Idee dahinter ist eine Beschreibung eines bestimmten Zeichens im Suchmuster. Wenn an einer Stelle eine Zahl zwischen 0 und 4 auftreten darf, schreibt man eine Zeichenklasse: `[0-4]` oder `[01234]`. In jedem Fall repräsentiert diese Klasse nur ein Zeichen.

Bildung von
Zeichenklassen

Zeichen	Bedeutung
.	Der Punkt repräsentiert jedes beliebige Zeichen
[abcd]	Eine Klasse wird durch eckige Klammern gebildet. Darin stehen die Zeichen, die die Klasse bilden oder die Sonderzeichen aus der Tabelle 4.3.
[^abcd]	Eine Klasse, in der die aufgeführten Zeichen nicht enthalten sind.
[a-z]	Mit Minuszeichen werden Bereiche von Zeichen definiert.
\p{name}	Vordefinierte Zeichenklasse <i>name</i> .
\P{name}	Alles außer dem Inhalt der vordefinierten Zeichenklasse <i>name</i> .
\w, \W	Wortzeichen bzw. kein Wortzeichen

Tabelle 4.4: Allgemeine und vordefinierte Zeichenklassen

Zeichen	Bedeutung
\s, \S	Whitespace bzw. kein Whitespace ²
\d, \D	Zahlzeichen bzw. keine Zahlzeichen
\b, \B	Wortgrenze bzw. keine Wortgrenze

Tabelle 4.4: Allgemeine und vordefinierte Zeichenklassen (Forts.)

Wiederholungsoperatoren

Wenn Sie mehr als nur ein Zeichen benötigen, müssen Sie einen der Wiederholungsoperatoren angeben.

Zeichen	Bedeutung
*	Kein oder beliebig viele Zeichen
+	Ein oder beliebig viele Zeichen
?	Kein oder genau ein Zeichen
{n}	Genau <i>n</i> Zeichen
{n,}	Mindestens <i>n</i> Zeichen
{n,m}	Mindestens <i>m</i> jedoch höchstens <i>n</i> Zeichen
*?	Das nachgestellte Fragezeichen macht den Ausdruck in diesem Bereich »ungierig«. Normalerweise sind reguläre Ausdrücke »gierig«, führen also die Prüfung solange fort, wie es geht, auch wenn die Erfüllung bereits möglich war. Dieses Verhalten kann mit dieser Notation unterdrückt werden.
+?	
??	

Tabelle 4.5: Wiederholungsoperatoren

Modifikatoren

Modifikatoren verändern das Verhalten des Ausdrucks innerhalb einer Gruppe oder eines Teils des Ausdrucks unabhängig von den globalen Optionen:

Konstrukt	Beschreibung
(?o)	Für <i>o</i> können Sie eine oder mehrere der folgenden Optionen einsetzen:
(?-o)	
	► <i>i</i> ignoriert Groß- und Kleinschreibung
	► <i>m</i> Mehrzeiliger Modus, ^ und \$ sind am Anfang und Ende jeder Zeile gültig

Tabelle 4.6: Modifikatoren und Kommentare in regulären Ausdrücken

2 Als Whitespace gelten Leerzeichen, Zeilenumbrüche, Tabulatoren usw.

Konstrukt	Beschreibung
	▶ <code>s</code> Einzeilenmodus, <code>.</code> erkennt auch das Zeilenendezeichen ▶ <code>x</code> Ignoriere Whitespaces, jedoch nicht in Zeichenklassen
<code>(?#)</code>	Kommentare. Die Zeichen vom <code>#</code> bis zum Ende der Klammer werden ignoriert.

Tabelle 4.6: Modifikatoren und Kommentare in regulären Ausdrücken (Forts.)

Direkter Zugriff auf Gruppen

Es besteht auch die Möglichkeit, auf die gefundenen Zeichen der Gruppen direkt im selben Ausdruck zuzugreifen. Der folgende Ausdruck sucht in einem Text Wörter, die doppelt hintereinander vorkommen:

```
\b([a-z]+)(\s)+(\1\b).
```

Der ersten Teil beginnt mit einer Wortgrenze `\b`. Danach wird als Wort erkannt, was nur aus Buchstaben besteht: `([a-z]+)`. Selbstverständlich würde man in der Praxis noch den `(?i`-Operator einfügen, um Groß- und Kleinschreibung zu gestatten. Dann folgen Leer- oder Trennzeichen; mindestens eins oder beliebig viele `(\s)+`. Der zweite Teil ist der eigentliche Clou an der Sache, hier wird nämlich mit dem speziellen Operator `\1` auf die erste gefundene Referenz verwiesen. Welche Ziffern man einsetzen muss, kann anhand der Anzahl der öffnenden Klammern ausgezählt werden. Danach folgt wieder eine Wortgrenze `\b`. Der Ausdruck findet in Sätzen wie: »Hier kommt kommt das Programm« die Doppelungen »kommt kommt« und gibt diese (und nur diese) zurück. Sind mehrere Worte doppelt, wird das ebenfalls erkannt. Sind in einem Text mehrere Wörter (hintereinander) doppelt, werden mehrere Gruppen zurückgegeben.

Gruppen
verwenden

Alternativen verwenden

Wenn ein Muster aus mehr als einem Basiswert besteht, sind Alternativen gefragt. Es gibt mehrere Möglichkeiten, alternative Bedingungen zu formulieren:

Konstrukt	Beschreibung
	Trennt zwei oder mehr Muster: aa ab bb
(?(ausdruck)y n)	Führt den Zweig y aus, wenn der <i>ausdruck</i> übereinstimmt, sonst n.
(?(name)y n)	Führt den Zweig y aus, wenn die benannte Gruppe <i>name</i> übereinstimmt, sonst n.

Tabelle 4.7: Formulierung von Alternativen

Der Umgang mit den letzten beiden Optionen ist keineswegs trivial, aber ausgesprochen flexibel. Man kann damit gut einige Zeilen konventionellen Codes sparen. Im Ja- bzw. Nein-Zweig steht ein weiteres Suchmuster, um weitere Teile des Ausdrucks zu prüfen.

4.7.3 Weitere Betrachtungen zu regulären Ausdrücken

Die enorme Bedeutung von regulären Ausdrücken mag Anfängern nicht offensichtlich sein. Wenn Sie diese oft und erfolgreich einsetzen, werden Sie sich nicht mehr vorstellen können, anders zu programmieren. Die Möglichkeiten gehen über die bereits beschriebenen Varianten weit hinaus. Reguläre Ausdrücke sind in ihrer ganze Breite buchfüllend. Dieser Abschnitt wird einige spezielle Aspekte zeigen, ohne dass ein Anspruch an Vollständigkeit besteht. Wenn Sie im Umgang mit regulären Ausdrücken nicht sicher sind, können Sie später hierher zurückkehren, für die ersten Schritte werden die Informationen nicht zwingend benötigt.

Verfeinerung des Konzepts der Gruppierungen

Zugriff auf Teile einer Zeichenkette

Beim Aufbau von Ausdrücken mit Gruppen gibt es zwei Vorgehensweisen:

- 1. ([Zeichenklasse])+
- 2. ([Zeichenklasse]+)

Im ersten Fall bildet das Zeichen eine Gruppe, die sich wiederholen darf. Im zweiten Fall werden die Zeichen wiederholt, und dann wird eine Gruppe gebildet. Beide erkennen dasselbe Suchmuster. Unterschieden wird durch die Schreibweise die Art der Repräsentation der Fundstellen im Ergebnis.

Im ersten Fall werden die Ergebnisse als Elemente der Groups-Kollektion abgelegt. Es handelt sich tatsächlich um typische Gruppen.

Im zweiten Falle wird jedoch der Inhalt der Gruppe aus Fundstellen gebildet. Wenn Sie auf die Fundstellen zugreifen möchten, hilft die `Capture`-Kollektion.

Das Beispiel *RegexCapture.aspx* enthält eine universelle Version eines Programms zur Prüfung regulärer Ausdrücke. Damit können Sie leicht Versuche vornehmen. Insbesondere zum Verständnis der Gruppierungen eignet sich dies hervorragend. Betrachten Sie die folgenden beiden Ausdrücke, die die gezeigten Vorgehensweisen bei der Gruppierung umsetzen, jeweils mit dem Resultat der erkannten Bestandteile:

Reguläre Ausdrücke

Gruppen und Capture-Kollektionen

Suchwort:

Ausdruck:

Der Ausdruck ist gültig

Treffer: 1234 an Position 0 = 1

Ausgabe der Captures-Kollektion: [1234] (4 Zeichen)

Treffer: 4 an Position 3 = 4

Ausgabe der Captures-Kollektion: [1][2][3][4] (1 Zeichen)

Abbildung 4.23: Ausdruck mit Wiederholung einer Gruppe

Wie an der ersten Zeile »Treffer« zu erkennen ist, wird der Ausdruck in beiden Fällen erkannt und auch als gleichartig bewertet. Diese Zeile entsteht durch Zugriff auf das erste Element der `Groups`-Kollektion:

```
ma.Groups(i)
```

Der zweite Treffer wertet dann alle durch Gruppen gebildeten Elemente aus. Im Beispiel gibt es nur eine Gruppe (ein Klammerpaar), sodass nur eine derartige Trefferzeile hinzukommt. Der Ausdruck in Abbildung 4.23 zeigt, dass als Gruppe nur ein Zeichen definiert wurde – die Zahlzeichenklasse `\d`. Die gesamte Gruppe wird dann wiederholt. Die Trefferzeile gibt nun jede Fundstelle entsprechend der Definition einzeln aus, angedeutet durch die eckigen Klammern. Die Kollektion wird folgendermaßen gebildet:

```
mc = ma.Groups(i).Captures
```

Reguläre Ausdrücke

Gruppen und Capture-Kollektionen

Suchwort:

Ausdruck:

Der Ausdruck ist gültig

Treffer: 1234 an Position 0 = 1
 Ausgabe der Captures-Kollektion: [1234] (4 Zeichen)

Treffer: 1234 an Position 0 = 1
 Ausgabe der Captures-Kollektion: [1234] (4 Zeichen)

Abbildung 4.24: Ausdruck mit Gruppierung eines wiederholten Zeichens

Die Variable `mc` enthält nun die Kollektion der inneren Gruppen. In einer Schleife erfolgt dann der Zugriff:

```
"[" + mc(j).Value + "]"
```

Der Ausdruck in Abbildung 4.24 gruppiert dagegen erst und wendet den Wiederholungsoperator `+` erst dann an. Dies führt dazu, dass der Ausdruck auch nur als Ganzes zur Verfügung steht.

```
Public Class RegexCapture
    Inherits System.Web.UI.Page
    Protected ausgabe As HtmlGenericControl
    Protected suchwort As TextBox
    Protected ausdruck As TextBox
    Protected WithEvents send As Button

    Public Sub CheckRegex_OnClick(ByVal sender As Object,
                                   ByVal e As EventArgs) _
        Handles send.Click
        Dim rx As Regex
        Dim ma As Match
        Dim mc As CaptureCollection
        Dim rxmail As String = ausdruck.Text
        rx = New Regex(rxmail, RegexOptions.IgnoreCase)
        ma = rx.Match(suchwort.Text)
        If (ma.Success) Then
            ausgabe.InnerHtml = "<b>Der Ausdruck ist gültig</b><p/>"
            Dim i As Integer
            For i = 0 To ma.Groups.Count - 1 Step 1
                ausgabe.InnerHtml += "Treffer: <u>" &
                    + ma.Groups(i).ToString()
                ausgabe.InnerHtml += "</u> an Position " &
                    + ma.Groups(i).Index.ToString() + " = "
```

```
mc = ma.Groups(i).Captures
ausgabe.InnerHtml += mc.Count.ToString()
ausgabe.InnerHtml += "<br/> Ausgabe der "
                        Captures-Kollektion: "

If (mc.Count > 0) Then
    Dim j As Integer
    For j = 0 To mc.Count - 1 Step 1
        ausgabe.InnerHtml += "[" + mc(j).Value + "]"
    Next
End If

Next

Else
    ausgabe.InnerHtml = "<b>Der Ausdruck ist ungültig</b>"
End If

End Sub

End Class
```

Listing 4.26: Das vollständige Programm zum Zugriff auf Capture-Gruppen (RegexCapture.aspx.vb)

Definition von Bedingungen

Es gibt verschiedene Konstrukte mit dem Gruppierungskonstrukt, die gleichzeitig Bedingungen prüfen. Die folgende Tabelle zeigt eine Zusammenfassung:

Bedingte
Teilausdrücke

Gruppierung	Beschreibung
()	Einfache Gruppe
(?<name>) (?'name')	Gruppe, deren Inhalt einem Element der Captures-Kollektion mit dem Index <i>name</i> zugewiesen wird
(?:)	Gruppe, die nicht in die Groups- oder Captures-Kollektion aufgenommen werden soll
(?=)	Vorausschauende zutreffende Bedingung
(?!)	Vorausschauende unzutreffende Bedingung
(?<=)	Zurückschauende zutreffende Bedingung
(?<!)	Zurückschauende unzutreffende Bedingung
(?>)	Unterdrückt die Gierigkeit eines Teilausdrucks

Tabelle 4.8: Gruppierungskonstrukte

Auf benannte Gruppen kann innerhalb des Ausdrucks mit \k<name> zugegriffen werden (die spitzen Klammern gehören dazu). Unbenannte Gruppen werden über die Nummerierung er-

Benannte
Gruppen

reicht – gezählt werden die öffnenden Klammern –, also beispielsweise \3 für die dritte Gruppe.

Bedingte Gruppen

Etwas schwieriger erscheint der Umgang mit Bedingungen. Hierbei wird ein Teil des Ausdrucks – vor oder nach dem betroffenen Teilausdruck – in die Prüfung mit einbezogen, nicht jedoch in die Selektion. Damit lassen sich Konstruktionen der Art »Prüfe, ob an der Stelle eine Zahl ist, aber nur dann, wenn dieser Zahl ein Buchstabe folgt«. Der Ausdruck wäre jedoch gültig, wenn der Zahl ein Sonderzeichen folgt.

Der folgende Ausdruck soll dieses Verhalten haben:

```
\d\d(?:![a-z])
```

Wenn Sie nun eine Zeichenfolge der Art »12cc« prüfen, ist dieser Ausdruck nicht gültig, »12_« dagegen wird akzeptiert, ebenso wie »1234«. In allen Fällen wird aber lediglich »12« als Ergebnis des ersten Elements der `Match`-Kollektion zurückgegeben, denn der bedingte Teil interessiert hier nicht.

Wenn Sie nun die Prüfung umkehren möchten, also vorausgehende Buchstaben zu erkennen sind, wird eine zurückschauende Bedingung benötigt:

```
(?<=[a-z])\d\d
```

Hier werden nun ebenfalls zwei Ziffern erkannt, die jedoch von einem vorausgehenden Buchstaben begleitet werden müssen. Das Suchwort »23c21x88« würde als Resultat »12« zurückgeben, weil die Ziffernfolge »12« als erste von einem Buchstaben angeführt wird.

4.8 Eigenschaften und Methoden der Datentypen

System.Type

Aus der Basisklasse `System.Type` des Namensraumes `System` werden die Datentypen abgeleitet. Der Umgang damit ist normalerweise trivial, weil VB.NET für wichtige Datentypen selbst Schlüsselwörter zum Zugriff bereitstellt. Da Datentypen aber auch Objekte sind, haben sie natürlich Eigenschaften und Methoden. Manchmal ist es erforderlich, darauf zuzugreifen, um beispiels-

weise einen Zustand zu ermitteln. In diesem Buch werden Sie oft die Methode `ToString` sehen. Sie steht praktisch jedem Datentyp zur Verfügung.

Die Datentypen selbst (`String`, `Byte` usw.) sind Aufzählungen aus der Klasse `Type`.

4.8.1 Eigenschaften der Datentypen

An erster Stelle steht oft die Analyse, ob eine Variable einen bestimmten Basisdatentyp hat. Außerdem kann festgestellt werden, wie die Variable deklariert wurde:

**Basisdatentyp
analysieren**

► *Basisdatentypen*

`isArray`, `IsClass`, `IsEnum`

► *Parametertypen*

`IsByRef`

► *Zugriffsebene*

`IsPrivate`, `IsPublic`, `IsSealed` usw.

► *Werttypen*

`IsValueType`

4.8.2 Methoden der Datentypen

`ToString` kennen Sie bereits, damit wird jeder Typ in eine Zeichenkette konvertiert. Andere wichtige Methoden sind `Equals` zur Feststellung der Gleichheit, `GetType` zur Ermittlung des Namens des Datentyps und `GetTypeArray` zur Ermittlung eines Arrays von Datentypen aus einem universellen Array.

```
<script runat="server" language="vb">
Private Sub Page_load()
    Dim aObject(2) As Object
    aObject(0) = 123
    aObject(1) = "Krause"
    aObject(2) = 13.04
    Dim aTypes() As Type = Type.GetTypeArray(aObject)
    ausgabe.InnerHtml = "Datentypen des Arrays:<br/>"
    Dim i As Integer
    For i = 0 To aTypes.Length - 1 Step i + 1
        ausgabe.InnerHtml += aTypes(i).FullName + "<br/>"
    Next
End Sub
```

```
End Sub
</script>
<html>
  <head>
    <title>Datentypen</title>
  </head>
  <body>
    <h1>Datentypen</h1>
    <p id="ausgabe" runat="server"/>
  </body>
</html>
```

Listing 4.27: Ermittlung von Datentypen (TypeGet.aspx, ohne Code-Behind-Datei)

Dieses Beispiel ist weniger spektakulär als es aussieht. Sie werden nur selten in die Verlegenheit kommen, wirklich Datentypen festzustellen. Wenn es erforderlich ist, können Sie aber die Methoden leicht auf jede Variable anwenden. Denken Sie daran, dass alle Variablen von Datentypen des Frameworks abstammen und damit Objekte sind und deshalb über diese Eigenschaften und Methoden verfügen – zusätzlich zu den kontextspezifischen.

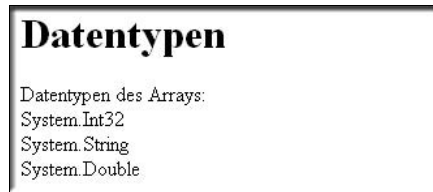


Abbildung 4.25: Ausgabe von Datentypen

4.9 Datum und Zeit

Datum und Zeit spielen in vielen Programmen eine große Rolle. Es gibt in jeder Programmiersprache und in jedem Betriebssystem andere Techniken, Datumswerte darzustellen. Vielleicht kennen Sie aus der Unix-Welt die Timestamps (Zeitmarken), die vom 1.1.1970 an zählen und ein 32-Bit-Wort nutzen. Dieser Zeitraum ist leider sehr begrenzt, das Geburtsdatum des Verfassers liegt früher und wäre nicht darstellbar.

Wenn im Folgenden von Datumswerten gesprochen wird, ist damit meist auch ein Zeitwert gemeint, andernfalls wird explizit darauf hingewiesen.



Im .NET-Framework ist die Struktur `DateTime` aus dem Namensraum `System` für den Umgang mit Datum und Zeit zuständig. Datumswerte können im Wertebereich vom 1.1.0001 0:00 Uhr (Mitternacht) bis zum 31.12.9999 23:59:59 Uhr verwendet werden. Die typischen Beschränkungen anderer Systeme sind damit endgültig passé. Die Genauigkeit basiert auf so genannten Ticks, dies sind Zeitsprünge von 100 Nanosekunden.

DateTime

Für Berechnungen mit Datumswerten dient außerdem die `TimeSpan`-Struktur. Objekte daraus enthalten Datumsdifferenzen statt absoluter Daten.

TimeSpan

4.9.1 Ein Datum ermitteln

Die Abfrage des Datums und der aktuellen Zeit gelingt mit der statischen Eigenschaft `Now`. Interessiert nur das Datum, ist `Today` die bessere Wahl. Die dritte statische Eigenschaft im Bunde ist `UtcNow` – die Darstellung im UTC-Format – früher bekannt als GMT (Greenwich Mean Time) und oft auch Weltzeit genannt. UTC steht für Universal Time Coordinated, ein 24-Stunden-Format, das international verwendet wird.

Aktuelles Datum

Die aktuelle Zeit bezieht sich immer auf den Server. Denken Sie daran, wenn Sie die Angabe zur Begrüßung verwenden, dass Benutzer in anderen Zeitzonen leben können.



Das folgende Programm verwendet drei Zeitdarstellungen:

```
<script runat="server" language="vb">
Private Sub Page_Load()
    ausgabe.InnerHtml = "Ausgabe von Datumswerten:"
    ausgabe.InnerHtml += "<br/>Es ist jetzt: " &
        + DateTime.Now
    ausgabe.InnerHtml += "<br/>Es ist heute: " &
        + DateTime.Today
    ausgabe.InnerHtml += "<br/>UTC: " &
        + DateTime.UtcNow
End Sub
</script>
<html>
<head>
```

```

        <title>Datum und Zeit</title>
    </head>
    <body>
        <h1>Datum und Zeit</h1>
        <p id="ausgabe" runat="server" />
    </body>
</html>

```

Listing 4.28: Ausgabemöglichkeiten für das aktuelle Datum (DateNow.aspx)

Da die drei Eigenschaften statisch sind, muss kein Objekt instanziiert werden.



Abbildung 4.26: Ausgabe des aktuellen Datums des Servers

Sollen Datum und Zeit getrennt werden, verwenden Sie die Eigenschaften `Date` bzw. `TimeOfDay`. Dazu muss jedoch eine Instanz gebildet werden. Die folgenden Beispiele nutzen denselben HTML-Teil wie das erste, er wird deshalb nicht erneut wiederholt.

```

Private Sub Page_Load()
    ausgabe.InnerHtml = "Ausgabe von Datumswerten:"
    ausgabe.InnerHtml += "<br/>Es ist jetzt: " + DateTime.Now
    Dim heute As DateTime = DateTime.Now
    ausgabe.InnerHtml += "<br/>Datum: " + heute.Date
    ausgabe.InnerHtml += "<br/>Uhrzeit: " + 
        + heute.TimeOfDay.ToString()
End Sub

```

Listing 4.29: Datum und Uhrzeit (Ausschnitt aus DateDatetime.aspx)

Hier wird zuerst eine Instanz des aktuellen Datums gebildet:

```
Dim heute As DateTime = DateTime.Now
```

Auf das Objekt *heute* werden die Eigenschaften dann angewendet. Die Abbildung zeigt, zu welcher Ausgabe das führt.

Ticks Die Zeit enthält den Anteil in der Genauigkeit, die die Ticks bieten (100 Nanosekunden). Das wird nicht oft benötigt, deshalb ist eine Abtrennung mit Zeichenketten-Methoden denkbar:

```
Dim zeit As String = heute.TimeOfDay.ToString()
zeit = zeit.Substring(0, zeit.IndexOf("."))
ausgabe.InnerHtml &= "<br/>Uhrzeit: " & zeit
```

Datum und Zeit

Ausgabe von Datumswerten:
 Es ist jetzt: 7.4.2002 23:01:59
 Datum: 7.4.2002 00:00:00
 Uhrzeit: 23:01:59.2677640

Abbildung 4.27: Ausgabe von Datum und Zeit

Die Trennung erfolgt am Punkt, der in der Zeitdarstellung nur ein Mal auftritt.

Um auf die Bestandteile von Datumswerten zugreifen zu können, **Datumswerte** gibt es natürlich auch die passenden Eigenschaften:

```
Private Sub Page_Load()
    ausgabe.InnerHtml = "Ausgabe von Datumswerten:"
    ausgabe.InnerHtml += "<br/>Es ist jetzt: " + DateTime.Now
    Dim heute As DateTime = DateTime.Now
    ausgabe.InnerHtml += "<br/>Datum: " + heute.Date
    ausgabe.InnerHtml += "<br/>Uhrzeit: " +
        + heute.TimeOfDay.ToString()
    Dim zeit As String = heute.TimeOfDay.ToString()
    zeit = zeit.Substring(0, zeit.IndexOf("."))
    ausgabe.InnerHtml += "<br/>Uhrzeit: " + zeit
End Sub
```

Listing 4.30: Zugriff auf Datumswerte (Ausschnitt aus DateDatetime2.aspx)

Die Eigenschaften dürften selbsterklärend sein. Sie geben alle – mit Ausnahme der Methode `DayOfWeek`, die den ausgeschriebenen englischen Wochentagsnamen ausgibt – numerische Werte zurück.

Datum und Zeit

Ausgabe von Datumswerten:
 Es ist jetzt: 7.4.2002 23:14:55
 Tag: 7
 Monat: 4
 Jahr: 2002
 Wochentag: Sunday
 Tag im Jahr: 97

Abbildung 4.28: Zerlegen eines Datums

- Schaltjahr** Um zu ermitteln, ob ein Jahr ein Schaltjahr ist, verwenden Sie die statische Methode `IsLeapYear`, die als Argument die Jahreszahl erwartet.
- Zeitwerte** Analog dazu werden für die Zeiten die Eigenschaften `Hour`, `Minute`, `Second` und `Millisecond` verwendet. Die Ticks werden mit der gleichnamigen Eigenschaft `Ticks` ermittelt. Der Datentyp, der zurückgegeben wird, ist `Long`.

4.9.2 Datumsberechnungen

Berechnungen mit Datumswerten sind im .NET-Framework sehr einfach. Dazu müssen Sie natürlich auch über Objekte verfügen, die nicht das aktuelle, sondern ein ganz spezifisches Datum enthalten. Dies ist möglich, indem Sie beim Instanziiieren des Objekts `DateTime` die entsprechenden Angaben machen:

```
Dim datum As DateTime = New DateTime(2002, 5, 26)
```

Stunde, Minute und Sekunden können analog als weitere Parameter angehängt werden. Berechnungen führen Sie nun aus, indem die Methoden `AddDays`, `AddMonth`, `AddYears` bzw. `AddHours`, `AddMinutes` und `AddSeconds` verwendet werden. Ist der übergebene Parameter negativ, erfolgt eine Subtraktion.

```
Private Sub Page_load()
    ausgabe.InnerHtml = "Berechnungen von Datumswerten:"
    Dim datum As DateTime = New DateTime(2002,5,26)
    ausgabe.InnerHtml += "<br/>Datum: " &
        + datum.ToShortDateString()
    ausgabe.InnerHtml += "<br/>Wochentag: " &
        + datum.DayOfWeek.ToString()
    Dim naechster As DateTime = datum.AddYears(1)
    ausgabe.InnerHtml += "<br/>Nächstes Jahr: " &
        + naechster.DayOfWeek.ToString()
End Sub
```

Listing 4.31: Berechnungen mit Datumswerten (Ausschnitt aus `DateAdd.aspx`)

Die Ausgabe zeigt die Wochentage des erzeugten Datums (26.5.2002) für das aktuelle Jahr (2002) und das folgende (2003):

Datum und Zeit

Berechnungen von Datumswerten:

Datum: 26.5.2002

Wochentag: Sunday

Nächstes Jahr: Monday

Abbildung 4.29: Ergebnis einer Datumsberechnung

Vergleiche zwischen Daten können mit den üblichen Operatoren (<, >, >=, ==, <=, !=) ausgeführt werden, die für diesen Zweck überladen wurden.

**Datums-
operatoren**

Reichen diese direkten Berechnungen nicht aus, können Sie mit Zeitspannen arbeiten. Diese werden aus der Struktur `TimeSpan` abgeleitet. Die Struktur kann als Datumswert in der folgenden Form dargestellt werden:

TimeSpan

`[-]d.hh:mm:ss:ff`

Dabei ist das Vorzeichen optional – bei Berechnungen mit negativen Werten wird eine Subtraktion ausgeführt. `d` bestimmt eine Anzahl Tage, `ff` den Teil einer Sekunde.

Der Konstruktor erwartet Ticks (ein Parameter), Zeiten (drei Parameter für Stunde, Minute und Sekunde) oder vier (Tage plus Zeit). Für Berechnungen sind auch ein paar Konstanten interessant – in der `TimeSpan`-Struktur als Felder definiert – die die Ticks pro Zeiteinheit enthalten: `TicksPerHour` für 1 Stunde, `TicksPerMinute` für eine Minute usw. bis hin zu `Zero` für 0. Instanzen von `TimeSpan` können direkt berechnet werden, dazu dienen die Methoden `Add` und `Subtract`.

**TicksPerHours
Add
Subtract**

Das folgende Beispiel berechnet einen Kursplan:

```
Private Sub Page_load()
    ausgabe.InnerHtml = "Eine Kurstafel:<br/>"
    Dim kurs As DateTime = New DateTime(2002,5,26,9,0,0)
    Dim pause As TimeSpan = New TimeSpan(0,30,0)
    Dim mittag As TimeSpan = New TimeSpan(1,15,0)
    Dim lektion As TimeSpan = New TimeSpan(0,90,0)
    ausgabe.InnerHtml += "<br/>1. Lektion: " &
        + kurs.ToShortTimeString()
    kurs = kurs.Add(lektion)
    ausgabe.InnerHtml += "<br/>Pause: " &
        + kurs.ToShortTimeString()
    kurs = kurs.Add(pause)
```

```

        ausgabe.InnerHtml += "<br/>2. Lektion: " + kurs.ToShortTimeString()
        kurs = kurs.Add(lecktion)
        ausgabe.InnerHtml += "<br/>Mittag: " + kurs.ToShortTimeString()
        kurs = kurs.Add(mittag)
        ausgabe.InnerHtml += "<br/>3. Lektion: " + kurs.ToShortTimeString()
        kurs = kurs.Add(lecktion)
        ausgabe.InnerHtml += "<br/>Pause: " + kurs.ToShortTimeString()
        kurs = kurs.Add(pause)
        ausgabe.InnerHtml += "<br/>4. Lektion: " + kurs.ToShortTimeString()
        kurs = kurs.Add(lecktion)
        ausgabe.InnerHtml += "<br/>Ende: " + kurs.ToShortTimeString()
    End Sub

```

Listing 4.32: Berechnungen auf Basis von TimeSpan (Ausschnitt aus DateTimeSpan.aspx)

Wie es funktioniert

Hier wird am Anfang das Startdatum festgelegt:

```
DateTime kurs = new DateTime(2002,5,26,9,0,0)
```

Dann werden die Zeitspannen programmiert, die zur Berechnung eingesetzt werden. Eine Pause soll 30 Minuten dauern:

```
TimeSpan pause = new TimeSpan(0,30,0)
```

Dann wird noch die Zeitspanne der Mittagspause festgelegt (eine Stunden und 15 Minuten):

```
TimeSpan mittag = new TimeSpan(1,15,0)
```

Zuletzt wird die Dauer einer Unterrichtseinheit auf 90 Minuten gesetzt:

```
TimeSpan lecktion = new TimeSpan(0,90,0)
```

Mit diesen Angaben erfolgt nun die Berechnung. Das Ende der ersten Lektion ist leicht zu ermitteln:

```
kurs = kurs.Add(lecktion)
```

Die folgenden Berechnungen laufen analog ab.

Datum und Zeit

Eine Kurstafel:

1. Lektion: 09:00
Pause: 10:30
2. Lektion: 11:00
Mittag: 12:30
3. Lektion: 13:45
Pause: 15:15
4. Lektion: 15:45
Ende: 17:15

Abbildung 4.30: Eine Kurstafel, mit *TimeSpan*-Objekten berechnet

Alternativ zur Verwendung von `Subtract` zum Abziehen von Zeiten können Sie das `TimeSpan`-Objekt auch mit der Methode `Negate` negieren und `Add` verwenden.

Negate

4.9.3 Datumswerte formatieren

Eine Formatierung wurde bereits verwendet: `ToShortDateString`. Diese Methode verkürzt das Datum auf eine Darstellung ohne den Zeitanteil. Entsprechend können Sie auch `ToShortTimeString` verwenden (Format »hh:mm«). Nebenbei findet in beiden Fällen eine Typkonvertierung nach `String` statt. Alternativ kann auch `ToLongDateString` und `ToLongTimeString` verwendet werden. Das ist meist nicht ausreichend, denn für gut gestaltete Webseiten werden sehr individuelle Datumsangaben verlangt. Dazu gehören auch sprachliche Versionen, also deutsche Wochentags- und Monatsnamen auf deutschen Seiten. Falls Ihre Seite mehrsprachig ist, müssen Sie darüberhinaus auch an andere Sprachen denken. Mit dem .NET-Framework sind Sie für viele Sprachen bestens gerüstet.

**Kurze und lange
Datumsformate**

Die Formatieranweisungen

Der einfachste Zugriff auf die Formatierung erfolgt mit der Methode `ToString`, die als Parameter eine Formatierungsanweisung bekommt:

**Individuelle
Datumsformate**

```
heute.ToString("dd.MM.yyyy")
```

Die Sache ist trickreich, wenn Sie komplizierte Formatierungen vorhaben. Es gibt nämlich zwei Formatierungsmöglichkeiten.

Zum einen können Sie vorgefertigte Kombinationen auswählen, beispielsweise stellt der Code `t` die Zeit in der Form »HH:mm« dar. Andererseits steht auch jede einzelne Zeitinformatio- zur Verfügung, um in der oben gezeigten Form das Ergebnis zusammenzusetzen. Dabei gelten folgende Regeln:

- ▶ Steht eine einzelne Formatanweisung allein, wird eine Kombination angenommen.
- ▶ Werden mehrere Formate angegeben, handelt es sich um eine Einzelformatierung
- ▶ Wollen Sie dennoch eine Einzelformatierung, setzen Sie ein %-Zeichen davor.
- ▶ Unbekannte Zeichen werden unverändert ausgegeben. Sollen andere Zeichen, die sonst eine Bedeutung haben, ausgegeben werden, schreiben Sie ein Backslash `\` davor.

Kombinations-
formate

Die folgende Tabelle zeigt eine Auswahl verfügbarer Kombina- tionsformate:

Format	Darstellung des folgenden Datums: 26.5.2002 9:00:00
D	Sonntag, 26. Mai 2002
f	Sonntag, 26. Mai 2002 09:00 Hinweis: Lokale Zeit
F	Sonntag, 26. Mai 2002 09:00:00 Hinweis: Lokale Zeit
m, M	26 Mai Hinweis: Hinter dem Tag steht kein Punkt
t	09:00
T	09:00:00
U	Sonntag, 26. Mai 2002 07:00:00 Hinweis: Die Zeit wird in UTC ausgegeben

Tabelle 4.9: Kombinationsformate (Auswahl)

Einzelformate

Die Liste der Einzelformate ist umfangreicher und erlaubt beliebige Kombinationen der einzelnen Bestandteile von Datum und Zeit.

Format	Beschreibung
d, dd	Tag ohne und mit führender Null
ddd	Kurzform des Wochentagsnamens (Fr)
dddd	Langform des Wochentagsnamens (Freitag)
M, MM	Monat ohne und mit führender Null
MM	Kurzform des Monatsnamens (Feb)
MMM	Langform des Monatsnamens (Februar)
yy	Jahr, zweistellig
yyyy	Jahr, vierstellig
H, HH h, hh	Stunden im 24-Stunden-Format ohne und mit führender Null, die kleinen Buchstaben erzeugen 12-Stunden-Format
m, mm	Minute ohne und mit führender Null
s, ss	Sekunde ohne und mit führender Null
tt	AM oder PM
:	Standardtrennzeichen für Zeitwert
/	Standardtrennzeichen für Datum, der Schrägstrich wird auf einem deutschen Windows als Punkt erscheinen.

Tabelle 4.10: Auswahl wichtiger Formatierungsanweisungen für Datum und Zeit

Das folgende Beispiel zeigt einige Formatierungen:

```
Private Sub Page_load()
    ausgabe.InnerHtml = "Kursinfo:<br />"
    Dim kurs As DateTime = New DateTime(2002,5,26,9,0,0)
    ausgabe.InnerHtml += kurs.ToString("dd.MM.yyyy \u\m H
    \U\h\r")
    ausgabe.InnerHtml += "<br/>"
    ausgabe.InnerHtml += kurs.ToString("dd/MM (ddd)")
    ausgabe.InnerHtml += "<br/>"
    ausgabe.InnerHtml += kurs.ToString("hh:mm")
End Sub
```

Listing 4.33: Datums- und Zeitformatierungen (Ausschnitt aus DateFormat.aspx)

Beachten Sie die Backslashes, mit denen freier Text in die Formatierung eingebaut wird:

```
ausgabe.InnerHtml += kurs.ToString("dd.MM.yyyy \u\m H \U\h\r")
```

Erwähnenswert ist auch die interne Umwandlung des /-Zeichens in einen Punkt bei der folgenden Formatierung:

```
ausgabe.InnerHtml += kurs.ToString("dd/MM (ddd)")
```

Die Abbildung zeigt den Effekt der Formatierungen:



Abbildung 4.31: Auswirkung von Datumsformatierungen

4.10 Globalisierung und Mehrsprachigkeit

Webanwendungen werden zwangsläufig von einem internationalen Publikum gesehen. Wenn Produkt und Vertriebsweg es zulassen, bietet sich ein mehrsprachiges Angebot an, um den Bedürfnissen der Benutzer gerecht zu werden. Dabei sind zwei Aspekte zu unterscheiden. Zum einen der mehrsprachige Aufbau der Webseiten. Hier wird fast immer eine Mischung aus vorgefertigten, statischen Informationen und dynamischen Fragmenten verwendet. In .NET wird dies durch Ressource-Dateien unterstützt, die spezifische Informationen enthalten. Die zweite Technik nutzt die Globalisierungstechniken, um vom System erzeugte Daten, wie Währungen oder Datumsangaben, in einer landesspezifischen Form darzustellen, ohne dass dies explizit programmiert werden muss.

4.10.1 Grundlagen der Globalisierung

Die formatierten Ausgaben folgen der aktuellen Spracheinstellung. Die Ausgabe folgt auf einem deutschen Windows den Regeln der deutschen Sprache. Wenn das nicht gewünscht ist oder auch dann sicher gestellt werden soll, wenn die Sprachversion des Betriebssystems nicht garantiert werden kann, müssen Sie sich mit den Globalisierungseinstellungen beschäftigen. Der dafür benötigte Namensraum `System.Globalization` muss zusätzlich eingebunden werden:

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

Verwenden Sie hinterlegten Code, erfolgt die Einbindung folgendermaßen:

```
Imports System.Globalization
```

Aus diesem Namensraum können Sie die Klasse `CultureInfo` verwenden, deren Konstruktor eine Zeichenkette der Form »sprach-code-land« erwartet. Beispielsweise werden die deutschsprachigen Länder mit »de-DE«, »de-AT«, »de-CH« usw. bezeichnet. Ist die Unterscheidung nicht wichtig, reicht auch die Angabe »de« aus.

Globalisierungseinstellungen für Datumsangaben

Die hier vorgestellten Methoden zur Lokalisierung sind nur ein Teil der Möglichkeiten, die .NET bietet. Mehr Informationen dazu finden Sie im Abschnitt 4.10.2, »Mehrsprachige Seiten programmieren« ab Seite 296.

```
Private Sub Page_Load()  
    ausgabe.InnerHtml = "Wochentage, mehrsprachig:<br/>"  
    Dim kurs As DateTime = New DateTime(2002,5,26,9,0,0)  
    Dim deutsch As CultureInfo = New CultureInfo("de-DE")  
    Dim france As CultureInfo = New CultureInfo("fr-FR")  
    Dim rossija As CultureInfo = New CultureInfo("ru-RU")  
    Dim chinese As CultureInfo = New CultureInfo("zh-CN")  
    ausgabe.InnerHtml += "<br>"  
                        + kurs.ToString("dd/MM (dddd)", deutsch)  
    ausgabe.InnerHtml += "<br>"  
                        + kurs.ToString("dd/MM (dddd)", france)  
    ausgabe.InnerHtml += "<br>"  
                        + kurs.ToString("dd/MM (dddd)", rossija)  
    ausgabe.InnerHtml += "<br>"  
                        + kurs.ToString("dd/MM (dddd)", chinese)  
End Sub
```

Listing 4.34: Ausgabe von Datumsangaben in Abhängigkeit von Sprache und Ländercode (Ausschnitt aus `DateFormatCulture.aspx`)

Im Beispiel werden vier Sprachobjekte abgeleitet, die für die Ausgabe der Daten in Deutsch, Französisch sowie Russisch und Chinesisch sorgen – unabhängig von der Sprache des Webserver-Betriebssystems. Das Ergebnis zeigt die folgende Abbildung:



Abbildung 4.32: Mehrsprachige Ausgabe von Wochentagen



Beachten Sie, dass für die Darstellung von asiatischen Schriftzeichen die entsprechenden Sprachpakete im Browser installiert sein müssen.

4.10.2 Mehrsprachige Seiten programmieren

In der Windows-Programmierung sind mehrsprachige Anwendungen seit langem im Einsatz. Bei der Komplexität heutiger Programme ist es fast unmöglich, parallel mehrere Codebasen zu führen, nur um die Sprache der Benutzeroberfläche unterschiedlich auszugeben. In ASP.NET stehen auch hier die Techniken zur Verfügung, die das .NET-Framework generell bietet.

Sprache des aktuellen Thread

Die erste Maßnahme besteht darin, die Sprache des aktuellen Thread zu verändern. Dies kann auf der Basis der folgenden Einstellungen erfolgen:

- Ermittlung der Standardsprachen des Browsers
- Auswahl des Benutzers aus einer vorgegebenen Liste

Der zweite Punkt dürfte trivial sein. Generell ist es jedoch empfehlenswert, dem Benutzer die Auswahl zu überlassen und gleichzeitig eine »Vermutung« über seine Präferenzen anzustellen. Browser bieten die Möglichkeit, die bevorzugte Sprache einzustellen und über die Anforderung an den Server zu senden. Sicher werden die meisten Benutzer davon nicht Gebrauch machen, aber allein aufgrund der Standardeinstellungen dürfte die Sprachauswahl in den meisten Fällen korrekt sein.

Sprache im Internet Explorer

Die Einstellungen der Sprache erfolgen beim Internet Explorer über EXTRAS | INTERNETOPTIONEN. Dort kann auf der Registerkarte ALLGEMEIN die Option SPRACHEN angeklickt werden. Im folgenden Dialog SPRACHEINSTELLUNG finden Sie eine Liste von Sprachen, beginnend mit einer Standardsprache am Anfang der Liste und mehrerer anderer Einträge. Der Browser kombiniert die

in der Liste eingetragenen Werte und qualifiziert sie mit einer Gewichtung, die von der Position abhängt. Der erste Eintrag hat immer die Gewichtung 1 und definiert damit die Primärsprache.



Abbildung 4.33: Spracheinstellungen im Internet Explorer

Die Basis: Das HTTP-Feld »Accept-Language«

Das hier beschriebene Verhalten ist kein .NET-spezifisches oder vom Internet Explorer vorgegebenes, sondern basiert auf dem Feld »Accept-Language« des Protokolls HTTP. Dort wird definiert, dass der Client eine Sprachangabe übermitteln sollte und der Server diese nach Möglichkeit auch zu beachten hat. Beide Seiten sind aber nicht zwingend darauf angewiesen, das heißt, es ist dem Client erlaubt, das Feld nicht zu senden und dem Server, es zu ignorieren.

Der Aufbau des Feldes sieht folgendermaßen aus:

Accept-Language: de, en-gb;q=0.7, en;q=0.3

Die Standardsprache ist hier Deutsch (de), der Benutzer würde aber auch Englisch im britischen Stil oder notfalls allgemeines Englisch akzeptieren. Freilich wird ein chinesischer Server, der keine anderen Sprachen kennt, dies einfach ignorieren. Es gehört aber zu einem guten Stil bei der Programmierung von Webseiten, die Sprachinformationen nicht gänzlich unbeachtet zu lassen. Der Browser stellt die Gewich-

tung mit dem Attribut `q=` ein, wobei der Parameter eine Gleitkommazahl mit maximal 3 Nachkommastellen zwischen 0 und 1 sein darf. Die Abstufung ist standardmäßig in der Folge (1, 0.5), (1, 0.7, 0.3), (1, 0.8, 0.5, 0.3) usw. definiert, wobei dies ein browserspezifisches Verhalten darstellt. HTTP selbst legt die Werte nicht fest. Letztlich bleibt es dem Benutzer überlassen, feinere Abstufungen vorzunehmen.

Des Weiteren gilt auch hier der Fallback-Mechanismus für Sprachangaben auf Basis der Sprachcodes.

Prinzip der Sprachcodes

ISO 3166 Die Sprachcodes verwenden ein einfaches System aus einem reinen Sprachwert, der in ISO 3166 definiert ist. ISO 639 definiert darüber hinaus Ländercodes, mit denen die Sprachen weiter unterteilt werden. So gibt es für die Sprache Deutsch – Code »de« – mehrere Landesbezeichner, beispielsweise »CH« für die Schweiz. Dies ist dann von Bedeutung, wenn beispielsweise Währungen ausgegeben werden. Derzeit wird als Währung für Deutschland der Euro ausgegeben, während Besucher aus der Schweiz sicher gern mit Schweizer Franken rechnen.

Wenn nun Ihre Seite eine derart feine Unterscheidung nicht benötigt, weil nur grob die Sprachen unterschieden werden, bietet die Art der Darstellung einen einfachen Fallback. Aus der Angabe »de-CH« kann leicht die Stammsprache, hier »de« abgeleitet werden, da diese immer vorn links und immer mit einem Binde- oder Unterstrich getrennt ist. Es hat sich eingebürgert, die Landescodes groß zu schreiben, ein Server, der solche Codes auswertet, sollte jedoch nicht darauf vertrauen.

In .NET Im .NET-Framework wird die als »Kultur« bezeichnete Kombination aus Sprache und Land auf den Standards ISO 639-1 und ISO 3166 aufgebaut. Dabei wird der Sprachcode aus ISO 639-1 genommen (Version -2 bezeichnet den Code dreistellig), dann folgt immer ein Bindestrich und dann der Ländercode, wie er in ISO 3166 festgelegt ist. Sprachen können darüberhinaus mehrere Schriftsysteme verwenden. So ist Serbisch sowohl mit lateinischen als auch kyrillischen Schriftzeichen darstellbar. Hier wird vor dem Sprachcode noch ein Präfix davorgesetzt – »Cy« für Kyrillisch und »Lt«

für Lateinisch. Sprachen, in denen eine alternative Schrift nicht verfügbar ist, erhalten keinen Präfix.

.NET verwendet darüber hinaus noch einen hexadezimalen Code für die Kultur, der intern verwendet wird. Dort steht für Deutsch/Deutschland, Kultur »de-DE«, der Code 0x0407. Dabei bezeichnet das niedrigere Byte 0x07 den Sprachcode Deutsch, die 0x04 das Land Deutschland.

Hexadezimale Sprachecodes

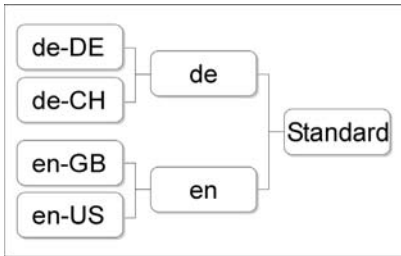


Abbildung 4.34: Ableitung der Sprache aus den vollständigen Kulturnamen

Bei der Programmierung stellt sich nun die Frage, wie man die Sprachcodes ermitteln kann.

Einstellungen des Browsers ermitteln

Der Browser liefert, wie bereits beschrieben, das Accept-Languages-Feld, wenn die entsprechenden Werte ausgefüllt sind. Wie dies erfolgt, wurde bereits in Abbildung 4.33 gezeigt. Die meisten HTTP-Felder, die der Browser sendet, werden als Kollektionen oder Eigenschaften im `Request`-Objekt gespeichert. Entsprechend einfach ist dann auch der Zugriff möglich. Das folgende Beispiel zeigt, wie die »Hauptsprache« und sämtliche sonst akzeptierten Sprachen ermittelt werden können. Die Ausgabe der `Page_Load`-Methode erfolgt in zwei Steuerelementen vom Typ `Label`.

```

Public Class CheckBrowserLanguage
    Inherits System.Web.UI.Page
    Protected ausgabe As Label
    Protected sprache As Label

    Private Sub Page_Load(ByVal sender As Object,   

        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Request.UserLanguages Is Nothing Then
            sprache.Text = Request.UserLanguages(0)
        End If
    End Sub
End Class
  
```

```

Dim lang As String
For Each lang In Request.UserLanguages
    ausgabe.Text += String.Format("Sprache {0} <br />", lang)
Next
End If
End Sub

```

Listing 4.35: Ermittlung der Sprachinformationen (Accept-Language) des Browsers (CheckBrowserLanguage.aspx.vb)

Wie es funktioniert

Die Kollektion für die Sprachcodes heißt `UserLanguages`. Der Index 0 enthält immer die Standardsprache:

```
sprache.Text = Request.UserLanguages(0)
```

Alle weiteren Indizes enthalten dann die Angaben, wie sie vom Browser gesendet werden. Die Auswertung der Qualifizierung müssen Sie selbst vornehmen.

Sprache des Browsers

Hauptsprache: **de**

Sie haben folgende Einstellungen im Browser:

Sprache de

Sprache en-us;q=0.8

Sprache en-au;q=0.5

Sprache en-gb;q=0.3

Abbildung 4.35: Ausgabe der Sprachangaben, die der Browser gesendet hat

Wenn Sie eine solche Liste empfangen, wäre es sinnvoll, dem Benutzer eine Auswahlmöglichkeit zu überlassen und zugleich den Standardwert entsprechend zu setzen. Das folgende Beispiel zeigt, wie dies erfolgen kann. Zur Ausgabe der Auswahlliste wird das Server-Steuerelement `RadioButtonList` verwendet:

```

<h1>Sprachauswahl</h1>
Bitte wählen Sie die Sprache für diese Website
<form id="BrowserLanguages" method="post" runat="server">
    <asp:RadioButtonList Runat="server" ID="sprachauswahl"/>
    <input type="submit" value="Ihre Auswahl"/>
</form>
<br/>
Ihre Auswahl: <asp:Label Runat="server" ID="auswahl"/>

```

Listing 4.36: Einfachste Form einer benutzerabhängigen Sprachauswahl (BrowserLanguages.aspx)

Die eigentliche Arbeit erledigt wieder der Code in der Code-Datei. Auf die Auswertung der gewählten Sprache selbst soll hier verzichtet werden – dies ist letztlich Aufgabe einer konkreten Applikation. Im Beispiel wird lediglich die tatsächliche Auswahl an das Label-Steuerelement `auswahl` übergeben.

```
Public Class BrowserLanguages
    Inherits System.Web.UI.Page
    Protected sprachauswahl As RadioButtonList
    Protected auswahl As Label
    Private ld As ListDictionary = New ListDictionary()
    Private ht As Hashtable = New Hashtable()

    Private Sub SetLanguageTable()
        ht.Add("de", "Deutsch, allgemein")
        ht.Add("de-ch", "Deutsch, Schweiz")
        ht.Add("de-at", "Deutsch, Österreich")
        ht.Add("de-de", "Deutsch, Deutschland")
        ht.Add("en", "English, common")
        ht.Add("en-gb", "English, Great Britain")
        ht.Add("en-us", "English, United States")
        ht.Add("en-au", "English, Australia")
    End Sub

    Private Sub Page_Load(ByVal sender As Object, 
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Page.IsPostBack Then
            SetLanguageTable()
            Dim s As String
            For Each s In Request.UserLanguages
                Dim subs() As String = s.Split(";")
                Dim langcode As String = subs(0).ToLower()
                If ht.Contains(langcode) Then
                    ld.Add(langcode, ht(langcode))
                End If
            Next
            sprachauswahl.DataSource = ld
            sprachauswahl.DataValueField = "Key"
            sprachauswahl.DataTextField = "Value"
            sprachauswahl.DataBind()
            sprachauswahl.SelectedIndex = 0
        Else
            auswahl.Text = sprachauswahl.SelectedItem.Value
        End If
    End Sub
End Class
```

Listing 4.37: Interaktive dynamische Auswahl der Sprache (BrowserLanguages.aspx.vb)

Wie es funktioniert

Im Beispiel wird eine Liste von Sprachen vorbereitet, die die Seite darstellen kann. Diese Liste wird in der Methode `SetLanguageTable` aufgebaut. Zum Einsatz gelangt hier der Typ `Hashtable`, mit dem einfache Schlüssel/Werte-Paare dargestellt werden können. Sie können so die Sprachcodes als Schlüssel verwalten und als Werte die Ausgabe für den Benutzer steuern.

Die eigentliche Arbeit wird in `Page_Load` erledigt. Zuerst wird festgestellt, ob das Formular das erste Mal aufgebaut wurde. Die Abfrage der Daten muss nur dies eine Mal erfolgen, nachfolgende Aufrufe behalten die Darstellung über den Anzeigestatus der Seite:

```
If Not Page.IsPostBack Then
```

Kollektion der Sprachen

Dann wird die `Hashtable`-Instanz `ht` aufgebaut. Nun kann die Kollektion der im Browser des Benutzers definierten Sprachen durchlaufen werden:

```
For Each s In Request.UserLanguages
```

Die Zeichenketten, die hier auftreten können, enthalten möglicherweise eine Qualifizierungsinformation, die vom Sprachcode mit einem Semikolon getrennt ist. Deshalb erfolgt eine Abtrennung des vorderen Teils:

```
Dim subs() As String = s.Split(";")
```

Der Sprachcode selbst wird nun in Kleinbuchstaben umgewandelt, falls die Browsereinstellungen abweichen:

```
Dim langcode As String = subs(0).ToLower()
```

Sprachcode ermitteln

Nun ist die Präsentation einer Auswahl auf Basis der Browsereinstellungen natürlich nur sinnvoll, wenn dies Ihre Site auch unterstützt. Deshalb wird in `ht` nachgeschaut, ob die Sprache auch definiert wurde:

```
If ht.Contains(langcode) Then
```

Ist das der Fall, wird eine zweite Liste aufgebaut, die nun eine Schnittmenge der Daten aus dem Browser und der vom Server akzeptierten enthält. Vermutlich ist diese Menge in der Praxis sehr klein, deshalb kommt der Typ `ListDictionary` zum Einsatz, der auf kleine Auflistungen bis 16 Elemente spezialisiert ist. Beachten Sie hier noch, dass die Definition aus dem Namensraum `System.Collections.Specialized` stammt. Mit dem Hinzufügen der passenden Elemente entsteht die Datenliste für die Optionsfelder:

```
ld.Add(langcode, ht(langcode))
```

Nun wird die fertige Liste dem Steuerelement zugewiesen:

Liste der Sprachen

```
sprachauswahl.DataSource = ld
```

Bei Optionsfeldern sind nun noch die Datenquellen für die Felder (value-Attribut) und der anzuzeigende Text zu definieren. Dazu werden die reservierten Wörter »Key« bzw. »Value« verwendet:

```
sprachauswahl.DataValueField = "Key"
```

```
sprachauswahl.DataTextField = "Value"
```

Mit der Bindung erfolgt die Anzeige:

```
sprachauswahl.DataBind()
```

Zuletzt wird noch die Auswahl auf den ersten Wert gesetzt:

```
sprachauswahl.SelectedIndex = 0
```

Im *Else*-Zweig wird, wenn der Benutzer seine Auswahl getroffen hat, der Code der Auswahl angezeigt. Diese Information würde in der praktischen Anwendung zur Steuerung der Seite herangezogen werden, beispielsweise zur Selektion der passenden Nachrichten aus einer Datenbank:

Auswahl der Sprache

```
auswahl.Text = sprachauswahl.SelectedItem.Value
```

Die folgende Abbildung zeigt die Reaktion des Formulars, wenn einige Sprachen im Browser eingerichtet wurden.

Sprachauswahl

Bitte wählen Sie die Sprache für diese Website

☐ Deutsch, allgemein

☐ English, common

☒ English, Australia

☐ English, Great Britain

☐ English, United States

Ihre Auswahl: en-au

Abbildung 4.36: Interaktive, browserabhängige Sprachauswahl

Nachdem nun die Sprachauswahl getroffen wurde, muss die Applikation selbst noch mehrsprachig programmiert werden. Auch hier bietet .NET eine exzellente Unterstützung.

Einrichten der Kultur für die aktuelle Sitzung

Um die Darstellung sprach- und kulturabhängiger Informationen anzupassen, ist der Zugriff auf zwei Namensräume notwendig. Zum einen betrifft das Klassen in `System.Globalization`. Alle Daten, die für die Einstellung einer bestimmten Kultur benötigt werden, sind hier verfügbar. Der zweite wichtige Namensraum ist `System.Threading`. Das mag auf den ersten Blick ungewöhnlich erscheinen, denn Multithreading-Programmierung ist bislang kein Thema in ASP.NET gewesen (diese Aufgabe übernimmt das ASP.NET-Modul selbst). Tatsächlich ist es aber so, dass .NET dem aktuellen Thread eine Information über die zu verwendende Kultur mitgeben kann. Damit wird auch sichergestellt, dass jeder Benutzer der Website seine eigenen sprachlichen Präferenzen einstellen kann.



Als »Kultur« wird hier allgemein die Zusammenfassung sprach- und landesspezifischer Informationen verstanden. Das betrifft die Darstellungen von Zahlen (Tausendertrennung, Dezimaltrennung), Schrift (Sprache, Richtung), Währungsangaben (Symbol, Ausrichtung) und natürlich Datums- und Zeitangaben (Kalender, Formatierung usw.).

CultureInfo

Die praktische Umsetzung nutzt die Klasse `CultureInfo` aus dem Namensraum `System.Globalization`. Dies wurde in Abschnitt 4.10.1, »Grundlagen der Globalisierung« ab Seite 294 bereits dargestellt. Nun gilt es die im letzten Beispiel gezeigte Applikation tatsächlich mehrsprachig zu programmieren. Dazu wird zuerst aus der Benutzerauswahl die Kultur ermittelt (`sprachauswahl` ist das `RadioButtonList`-Steuerelement):

```
MyCulture = New CultureInfo (sprachauswahl.SelectedItem.Value)
```

Diese kann dann dem Thread zugewiesen werden:

```
Thread.CurrentThread.CurrentCulture = MyCulture
```

Das folgende Beispiel zeigt die Ausgabe einer Währungsangabe und eines Kalenders in Abhängigkeit von der Sprachauswahl, die auf dem letzten Beispiel basiert.

```

<h1>Sprachauswahl</h1>
Bitte wählen Sie die Sprache für diese Website
<form id="BrowserLanguages" method="post" runat="server">
  <asp:RadioButtonList Runat="server" ID="sprachauswahl"/>
  <input type="submit" value="Ihre Auswahl"/>
<br/>
Ihre Auswahl: <asp:Label Runat="server" ID="auswahl"/>
<br/>
Währung: <asp:Label Runat="server" ID="waehrung"/>
<br/>
Kalender: <asp:Calendar Runat="server" ID="kalender"/>
</form>

```

Listing 4.38: Sprachauswahl und deren Auswirkung auf Ausgaben
(BrowserLanguagesCulture.aspx)

Die Code-Datei steuert nun zum einen den Aufbau der Liste der Optionsfelder, zum anderen die Reaktion der Ausgaben durch Verändern der aktuellen Kultur. Zu beachten ist hierbei, dass nur »echte« Kulturangaben verwendet werden können.

Ein »Fallback«-Pfad, wie in Abbildung 4.34 gezeigt, ist mit der statischen Methode *CreateSpecificCulture* nutzbar. Kulturen, die nicht vollständig aufgelöst sind, wie beispielsweise »fr«, werden als invariant bezeichnet. Vollständig wäre »fr-ch« (Französisch in der Schweiz) usw.



Die Auswertung ergänzt die Ausgabe im *Else*-Zweig, wie bereits in Listing 4.37 gezeigt wurde. Der erweiterte Zweig sieht nun folgendermaßen aus:

```

Dim culture As String = sprachauswahl.SelectedItem.Value
Thread.CurrentThread.CurrentCulture ←
    = CultureInfo.CreateSpecificCulture(culture)
auswahl.Text = sprachauswahl.SelectedItem.Value
Dim geld As Double = 59.9
waehrung.Text = geld.ToString("C")

```

Listing 4.39: Setzen einer Kultur für den aktuellen Thread (Ausschnitt aus BrowserLanguagesCulture.aspx.vb)

Dieses Beispiel kann nun nicht nur die im Browser eingestellte Sprache erfassen, sondern auch auf die Auswahl des Benutzers so umsetzen, dass sprachabhängige Daten passend formatiert werden.

Ihre Auswahl: de-at Währung: € 59,90 Kalender:	Ihre Auswahl: en-us Währung: \$59.90 Kalender:	Ihre Auswahl: fr-fr Währung: 59,90 € Kalender:
< Juli 2002 >	< July 2002 >	< juillet 2002 >
Mo Di Mi Do Fr Sa So	Sun Mon Tue Wed Thu Fri Sat	lun. mar. mer. jeu. ven. sam. dim.
24 25 26 27 28 29 30	30 1 2 3 4 5 6	24 25 26 27 28 29 30
1 2 3 4 5 6 7	7 8 9 10 11 12 13	1 2 3 4 5 6 7
8 9 10 11 12 13 14	14 15 16 17 18 19 20	8 9 10 11 12 13 14
15 16 17 18 19 20 21	21 22 23 24 25 26 27	15 16 17 18 19 20 21
22 23 24 25 26 27 28	28 29 30 31 1 2 3	22 23 24 25 26 27 28
29 30 31 1 2 3 4	4 5 6 7 8 9 10	29 30 31 1 2 3 4

Abbildung 4.37: Darstellung einer Währungsangabe und eines Kalenders in drei Kulturformen (Österreich, USA und Frankreich)

Wenn Sie die Einstellungen noch manipulieren möchten, ist indes das Anlegen einer Instanz der Klasse `CultureInfo` sinnvoll. Dies erfolgt entweder über den Konstruktor der Klassen, wenn die vollständige Zeichenkette einer Kultur vorliegt, oder über die statische Methode `CreateSpecificCulture`.

Kulturspezifische Informationen für Kalender

Die Klasse
Calendar

Beim Umgang mit verschiedenen Kulturen geht es nicht immer nur um ein Währungsformat oder die Darstellung von Zahlen. Komplexer sind meist Kalenderinformationen. So wird nicht überall auf der Welt der bei uns gebräuchliche Gregorianische Kalender verwendet. Wenn Sie eine Website entwickeln, die in Israel, Korea und Deutschland gleichermaßen verwendet wird, sollten Sie sich die Klasse `Calendar` anschauen, die ebenfalls in `System.Globalization` definiert ist.

Begriffe und Grundlagen zu Kalendern

Kalender sind, wenn komplexe Umrechnungen und Darstellungen benötigt werden, nicht trivial. .NET unterstützt Kalender so umfassend, dass praktisch jedes Datum darstellbar ist. Das verlangt in der Praxis, sich mit einigen Begriffen auseinander zu setzen.

Kalender definieren den Begriff der Ära. Der bei uns verwendete Gregorianische Kalender kennt zwei Ären: vor Christus und nach Christus. Die Ära nach Christus beginnt mit dem Jahr 1 und zählt die Jahre fortlaufend, bis heute sind es 2002. Die Implementierung in .NET verarbeitet allerdings nur die aktuelle Ära. Ähnlich funktioniert das auch beim hebräischen

Kalender, der nur die aktuelle Ära verarbeiten kann, das sind dort die Jahre 5343 bis 6000 (entspricht 1582 bis 2240 im gregorianischen Kalender). Auch dieser Kalender hat 12 Monate, im Gegensatz zu unserer Zählung schwankt aber Anzahl der Tage im Monat Cheschwan (2) und Kislew (3) je nach Lage der jüdischen Feiertage. Darüber hinaus gibt es statt eines Schalttags im Schaltjahr einen ganzen Schaltmonat. Einfacher ist der koreanische Kalender, der ähnlich dem unsrigen aufgebaut ist. Lediglich die Jahreszahl ist anders definiert. Das Jahr 2002 bei uns entspricht in Korea dem Jahr 4335.

Generell basieren Kalender auf einer Folge numerischer Werte, die den Abschnitten Jahr, Monat, Tag, Stunde, Minute, Sekunden und Sekundenbruchteil zugeordnet sind. Zu dieser Gruppe von Werten gibt es wiederum textuelle Darstellungen für die formatierte Ausgabe.

Es lohnt also, die vorgefertigten Kalender zu verwenden, statt mit viel Aufwand eigene Kreationen zu programmieren.

Die Klasse `Calendar` ist abstrakt. Sie können daraus eigene Kalender entwickeln oder eine der fertigen Ableitungen verwenden. Das folgende Beispiel lässt die Auswahl von einem aus drei Kalendern zu:

```
<h1>Kalender</h1>
Wählen Sie Ihren Kalender / Choose your calendar:
<form id="GlobalizationCalendars" method="post"
  runat="server">
  <asp:RadioButtonList Runat="server"
    ID="calendarselect"
    AutoPostBack="True">
    <asp:ListItem Runat="server" Selected="True"
      Value="greg">Gregorianisch</asp:ListItem>
    <asp:ListItem Runat="server" Selected="False"
      Value="korea">Koreanisch</asp:ListItem>
    <asp:ListItem Runat="server" Selected="False"
      Value="hebrew">Hebräisch</asp:ListItem>
  </asp:RadioButtonList>
  Das aktuelle Jahr in diesem Kalender ist:
  <asp:Label Runat="server" ID="calendar"/>
</form>
```

Listing 4.40: Auswahl eines Kalenders und Anzeige der Jahreszahl dieses Kalenders (GlobalizationCalendars.aspx)

Der aus der Optionsliste übertragene Wert wird nun in der Page_Load-Methode ausgewertet, ein passender Kalender erzeugt und das Jahr auf Basis des aktuellen Datums berechnet.

```
Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
    Handles MyBase.Load

    If (Page.IsPostBack) Then
        Dim ca As System.Globalization.Calendar = Nothing
        Dim dt As System.DateTime = System.DateTime.Now
        Select Case calendarselect.SelectedItem.Value
            Case "greg"
                ca = New System.Globalization.GregorianCalendar()
                calendar.Text = ca.GetYear(dt).ToString()
            Case "korea"
                ca = New System.Globalization.KoreanCalendar()
                calendar.Text = ca.GetYear(dt).ToString()
            Case "hebrew"
                ca = New System.Globalization.HebrewCalendar()
                calendar.Text = ca.GetYear(dt).ToString()
        End Select
    End If
End Sub
```

Listing 4.41: Nutzung von fertigen Kalendern zur Datumsdarstellung (Globalization-Calendar.aspx.vb)

Wie es funktioniert

Der Einfachheit halber wird hier die Variable `ca` als Typ `Calendar` angelegt:

```
Dim ca As System.Globalization.Calendar = Nothing
```

Dann wird das aktuelle Datum ermittelt:

```
Dim dt As System.DateTime = System.DateTime.Now
```

Kalender erzeugen

Je nach Auswahl wird dann der passende Kalender erzeugt:

```
ca = New System.Globalization.HebrewCalendar()
```

Die Kalenderklassen bieten viele Methoden zum Ausgeben der aktuellen Datumsfragmente und zum Ausführen von Berechnungen mit ihnen, beispielsweise `GetYear` und `AddYear` usw.:

```
calendar.Text = ca.GetYear(dt).ToString()
```

Der Umgang damit ist relativ unkritisch. Die Reaktion des Programms zeigt die nächste Abbildung.

Kalender

Wählen Sie Ihren Kalender / Choose your calendar:

- ☐ Gregorianisch
- ☒ Koreanisch
- ☐ Hebräisch

Das aktuelle Jahr in diesem Kalender ist: 4335

Abbildung 4.38: Ausgabe des aktuellen Jahres im koreanischen Kalender

In den Beispielen wurde die Klasse `Calendar` immer vollständig referenziert, trotz der Einbindung des passenden Namensraumes. Dies ist notwendig, weil »Calendar« auch in `System.Web.UI.WebControls` bekannt ist, dort zur Darstellung eines Kalenders als Steuerelement. Wenn Sie beide Namensräume gleichzeitig verwenden, was oft vorkommt, müssen Sie die Referenzierung dennoch vollständig vornehmen. Andernfalls erhalten Sie einen Compiler-Fehler.



Nachdem Währungs-, Datums- und Kalenderinformationen in der richtigen Kultur angezeigt werden, gilt es nun, auch die Sprache der gesamten Site anzupassen.

Ressourcen-Dateien als Basis mehrsprachiger Applikationen erzeugen

Für mehrsprachige Sites müssen Sie letztlich alle Texte, die irgendwo auftauchen, in die entsprechende Sprache übersetzen. Vermutlich stellt dies den größten Aufwand dar. Im internationalen Internet kann sich die Mühe aber lohnen. Es gibt prinzipiell zwei Strategien, eine Site mehrsprachig zu entwickeln. Beide werden praktisch verwendet.

Das einfachere Prinzip besteht darin, jede Vorlage, also die *aspx*-Dateien, für jede Sprache neu zu schreiben. Der hinterlegte Code sollte identisch bleiben. Dies ist praktikabel, wenn der Textanteil sehr groß ist und sich die Art und Weise der Gestaltung der Texte von Sprache zu Sprache stark unterscheidet. So kommt es häufig vor, dass deutsche Wörter sehr viel länger sind als ihre englischen

Prinzipien der Ressourcen-Dateien

Pendants. Ein einfacher Austausch führt dann zu einem verzerrten Layout. Eine gegen Längenänderungen resistente Gestaltung kann aufwändiger oder gar unmöglich sein. Dennoch bietet ASP.NET mit der Code-Behind-Technik (siehe Abschnitt 1.5.2, »Code Behind« ab Seite 79) bereits beste Voraussetzungen.

Anwendungsfall Wenn in einer Site wenig statischer Text vorhanden ist und mehrere Sprachen angeboten werden, ist die Verwendung von Ressourcendateien in Erwägung zu ziehen. Die Texte werden dann in externen Dateien gehalten und in den Vorlagen wird mit Variablen gearbeitet. Praktisch wird für jede unterstützte Sprache eine eigene Datei angelegt, die einen Schlüsselwert und eine Zeichenkette für jeden Text enthält:

```
;file culture.de.txt
title=Lokalisierte Texte in ASP.NET
langcode=Aktuell wurde folgender Code ausgewählt
date=In dieser Sprache sieht das Datum folgendermaßen aus
```

Dateinamen Der Dateiname ist frei wählbar, sollte jedoch einem bestimmten Schema folgen, damit Sie tatsächlich mit mehreren Sprachen arbeiten können. Notwendig ist eine Standarddatei, beispielsweise »culture.txt«. Darauf folgen die landesspezifischen Formen, »culture.de.txt«, »culture.en.txt«. In einem weiteren Konvertierungsschritt entsteht daraus die *resources*-Datei, die die Daten binär speichert.

Das resx-Format Es gibt statt dieser Textdarstellung auch die Möglichkeit, das *resx*-Format zu verwenden, ein einfaches XML-Format. Die Namensvergabe funktioniert ähnlich wie bei den Textdateien, auch hier muss als Endung *.resx* verwendet werden, um die Bedeutung der Datei zu unterstreichen. In einem weiteren Konvertierungsschritt entsteht daraus die *resources*-Datei, die die Daten binär speichert.

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" xmlns=""
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="data">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="value" type="xsd:string"
            minOccurs="0" msdata:ordinal="2" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" />
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <data name="title" value="Lokalisierte Texte in ASP.NET" />
  <data name="langcode" value="Aktuell wurde folgender Code ausgewählt" />
  <data name="date" value="In dieser Sprache sieht das Datum folgendermaßen aus" />
</root>
```

```
<xsd:attribute name="type" type="xsd:string" />
<xsd:attribute name="mimetype" type="xsd:string" />
</xsd:complexType>
</xsd:element>
<data name="title">
  <value>Lokalisierte Texte in ASP.NET</value>
</data>
<data name="langcode">
  <value>Aktuell wurde folgender Code ausgewählt</value>
</data>
<data name="date">
  <value>In dieser Sprache sieht das
    Datum folgendermaßen aus
  </value>
</data>
</root>
```

Listing 4.42: Vollständige Ressource-Datei mit Schema-Definition

Aus der Schemadefinition am Anfang der Datei lässt sich die Gestaltungsbreite der Tags ableiten. Generell wird jeder Datenwert in einem Element `<data>` gehalten. Darin sind ein oder mehrere Elemente vom Typ `<value>`, die Zeichenketten enthalten. Zulässige Attribute für `<data>` sind:

Aufbau der Datei

► name

Hiermit erfolgt die Angabe des Namens für diesen Ressourcewert. Dies ist der Schlüssel, um den Text später auf der Seite zuordnen zu können.

► type

Hier können Sie den Datentyp des Wertes festlegen, wenn er keine Zeichenkette darstellt. Sie müssen den Typ vollständig und mit Angabe der Assembly schreiben, beispielsweise `type="System.Int32, mscorlib"`.

► mimetype

Für die Speicherung von Bildern oder binären Daten sollten Sie den MIME-Typ angeben. Der MIME-Typ `image/png` weist darauf hin, dass die Daten als Binärdaten eines PNG-Bildes zu interpretieren sind.

Aus den Attributen lässt sich schon ableiten, dass die Ressourcendateien im XML-Format flexibler sind. Denn in modernen Webseiten kommen häufig Bilder zum Einsatz, die Texte enthalten.

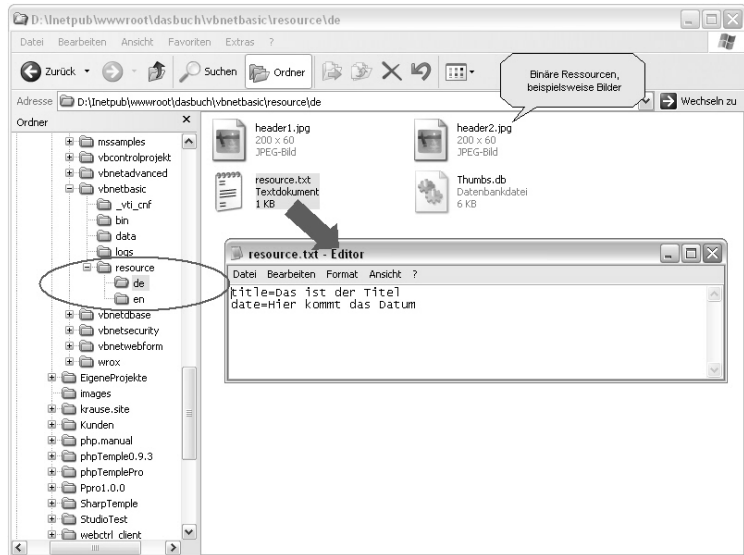


Abbildung 4.39: Aufbau der Quellverzeichnisse und der Textdatei für die automatische Erstellung von Ressourcen

Binäre Daten in Ressource-Dateien

Das Einbetten binärer Daten ist freilich nicht ganz einfach. Das Framework bietet aber auch hierzu alle denkbaren Hilfsmittel. Wenn Sie mit sehr vielen Ressourcen arbeiten, bietet sich folgender Weg an:

1. Erzeugen Sie ein Verzeichnis */resource* unterhalb des Stammverzeichnisses der Präsenz.
2. Erzeugen Sie für jede Sprache ein Unterverzeichnis mit dem Sprachcode, beispielsweise */de*.
3. Legen Sie alle Bilder oder Binärdateien in die passenden Unterverzeichnisse.
4. Legen Sie eine Ressourcendatei mit den Texten in das passende Unterverzeichnis.
5. Starten Sie das nachfolgend gezeigte Programm zum Erzeugen der Ressourcendateien.
6. Konvertieren Sie die Ressource-Datei in das interne Binärformat für Ressourcen. Dies wird nachfolgend beschrieben.

Mehr Komfort für Ressourcen-Dateien mit ResEditor

Gelegentlich ist von einem grafischen Editor mit dem Namen *ResEditor* zu lesen. Dieser soll der Erstellung komplexerer Ressourcendateien dienen. Tatsächlich wird dieses Programm nicht fertig mitgeliefert, sondern liegt dem Framework SDK als Beispielprogramm für eine WinForms-Applikation bei. Der Einsatz lohnt nicht unbedingt, wenn Sie das Programm jedoch kennen lernen möchten, müssen Sie es selbst übersetzen. Gehen Sie dazu folgendermaßen vor:

1. Öffnen Sie ein Kommandozeilenfenster, beispielsweise über **START | AUSFÜHREN**.
2. Suchen Sie den folgenden Pfad (die Laufwerksangabe müssen Sie an Ihre Installation anpassen):

```
C:\Programme\Microsoft Visual Studio .NET\FrameworkSDK  
\Samples\Tutorials\resourcesandlocalization\reseditor
```

3. Starten Sie dort die Stapelverarbeitungsdatei *build.bat*. Das Programm wird nun übersetzt und es entsteht ein Programm *ResEditor.exe*.
4. Starten Sie nun das Programm *ResEditor.exe*.

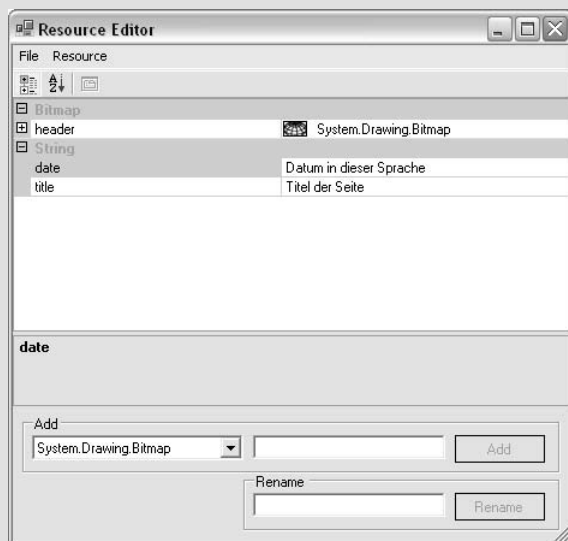


Abbildung 4.40: Der grafische Resource Editor in Aktion

Ressourcen- Dateien automatisch erstellen

In der Praxis hat sich freilich gezeigt, dass die so erstellten Dateien recht mühevoll zu bearbeiten sind. Oft werden Texte von externen Übersetzern angefertigt und Bilder von Grafikern entworfen. Eine »automatische« Entnahme aus einem Verzeichnis erscheint sinnvoller. Es gibt nun auch dafür zwei Wege. Zum einen können Sie *resx*-Dateien erstellen (das ist das XML-Format) und diese dann mit dem Kommandozeilenwerkzeug *resgen* in binäre Ressourcen-Dateien konvertieren. Dasselbe Werkzeug kann binäre Dateien, die dann die Dateierweiterung *.resources* tragen müssen, wieder zurück in *resx*-Dateien umwandeln. Für ein eigenes Programm ist es natürlich sinnvoll, gleich das binäre Format zu erstellen. Die Laufzeitumgebung, die später die Dateien in Abhängigkeit von der Benutzerwahl verwendet, kann nur mit dem binären Format umgehen. Dies dient vor allem höchster Effizienz. Das ständige Parsen von XML-Dateien wäre bei einem hochbelasteten Server nicht vertretbar.

System.Resources

Zum Erstellen und Verwenden von Ressourcen-Dateien dient der folgende Namensraum:

System.Resources

Eine kleine Webform dient der Auswahl der Verzeichnisse und der Festlegung des Stammnamens der Ressourcen-Dateien:

```
<h1>Hilfsprogramm zum Erstellen von Ressource-Dateien</h1>
Wählen Sie Ihr Ressource-Verzeichnis aus:
<form id="GlobalizationGetResource" method="post" ⌂
  runat="server">
  Verzeichnis: <asp:DropDownList Runat="server" ⌂
                                ID="stammverzeichnis"/>
  <br/>
  Basisname: <asp:TextBox Runat="server" ⌂
                                ID="basisname"/>.<i>culture</i>.resource
  <br/>
  <input type="submit" value="Konvertierung starten"/>
</form>
<asp:Label Runat="server" ID="ergebnis"/>
```

Listing 4.43: Formular des Konvertierungsprogramms (GlobalizationGetResource.aspx)

Die eigentliche Arbeit steckt hier wieder in der Code-Datei:

```
Public Class GlobalizationGetResource
  Inherits System.Web.UI.Page
  Protected stammverzeichnis As DropDownList
  Protected basisname As TextBox
  Protected ergebnis As Label
```

```

Private basispfad As String
Private Const ResExtension As String = ".resources"
Private di As DirectoryInfo

Private Sub Page_Load(ByVal sender As Object,   

                        ByVal e As System.EventArgs)   

Handles MyBase.Load
    If (Page.IsPostBack) Then
        di = New DirectoryInfo(   

            Server.MapPath(stammverzeichnis.SelectedItem.Value))
        ergebnis.Text = ""
        Dim rw As ResourceWriter
        Dim subdir As DirectoryInfo
        For Each subdir In di.GetDirectories()
            basispfad   

                = Server.MapPath(stammverzeichnis.SelectedItem.Value   

                    + "\" + basisname.Text + "." + subdir.Name +
ResExtension)
            rw = New ResourceWriter(basispfad)
            ergebnis.Text += basispfad + "<br/>"
            Dim fi As FileInfo
            For Each fi In subdir.GetFiles()
                Select Case fi.Extension.ToLower()
                    Case ""
                    Case ".gif"
                    Case ".png"
                        Dim img As System.Drawing.Image   

                            = System.Drawing.Image.FromFile(fi.FullName)
                        rw.AddResource(fi.Name, img)
                        Exit For
                    Case ".txt"
                        Dim sr As StreamReader = New
StreamReader(fi.FullName)
                        Dim sFile As String
                        While sr.Peek() <> -1
                            sFile = sr.ReadLine()
                            Dim aFile() As String = sFile.Split("="c)
                            rw.AddResource(aFile(0).Trim(),
aFile(1).Trim())
                        End While
                    End Select
                Next
            Next
            rw.Close()
        Next
    Else
        di = New DirectoryInfo(Server.MapPath("."))
        stammverzeichnis.DataSource = di.GetDirectories()
    End If
End Sub

```

```

        stammverzeichnis.DataBind()
    End If
End Sub
End Class

```

Listing 4.44: Programm zum Erstellen von binären resources-Dateien (Globalization-GetResource.aspx.vb)



Wie es funktioniert

Dieses Programm macht bereits Gebrauch von Datei- und Verzeichnis-zugriffen. Die Grundlagen dazu finden Sie im Abschnitt 4.10, »Globalisierung und Mehrsprachigkeit« ab Seite 294.

Beim ersten Aufruf wird der *Else*-Zweig ausgeführt. Hier wird das aktuelle Verzeichnis gelesen und eine Liste aller Unterverzeichnisse in einem DropDownList-Element angezeigt:

```
di = New DirectoryInfo(Server.MapPath("."))
```

Die Liste der Verzeichnisse kann dem Steuerelement direkt als Datenquelle zugewiesen werden:

```
stammverzeichnis.DataSource = di.GetDirectories()
```

Sie müssen nun lediglich das Verzeichnis wählen und den Stammnamen eintragen. Der Name wird dann nach dem Schema `<name>.<unterverzeichnis>.resources` festgelegt. Empfehlenswert ist es, die Verzeichnisse nach dem Sprachcode mit »de«, »en« usw. zu bezeichnen.

Hilfsprogramm zum Erstellen von Ressource-Dateien

Wählen Sie Ihr Ressource-Verzeichnis aus:

Verzeichnis: 
 Basisname: *cultur.resource*

c:\inetpub\wwwroot\dasbuch\csharpbasis\resource\culture.de.resources
 c:\inetpub\wwwroot\dasbuch\csharpbasis\resource\culture.en.resources

Abbildung 4.41: Das Programm in Aktion

Das Programm ermittelt zuerst den Basispfad, ab dem nach den Verzeichnissen gesucht werden soll, auf Grundlage des Optionswertes der Auswahlliste:

```

di = New DirectoryInfo (
    Server.MapPath (stammverzeichnis.SelectedItem.Value))

```


Die Verzeichnisse werden nun durchlaufen.

```
For Each subdir In di.GetDirectories()
```

Anschließend wird der Basispfad zusammengestellt und eine neue Ressourcen-Datei für diesen Zweig erzeugt:

```
rw = New ResourceWriter (basispfad)
```

Für diese Datei wird jetzt das betreffende Verzeichnis durchlaufen und alle Dateien werden gelesen:

```
For Each fi In subdir.GetFiles()
```

Im Beispiel werden nur Bilddateien mit den Erweiterungen *.gif*, *.png* und *.jpg* erkannt. Selbstverständlich können Sie jede Binärdatei integrieren. Der Zugriff auf Bilddateien erfordert die Klasse *Image* aus dem Namensraum *System.Drawing*. Weil *Image* auch in *System.Web.UI.WebControls* bekannt ist, muss der Namensraum komplett geschrieben werden:

```
Dim img As System.Drawing.Image ←  
    = System.Drawing.Image.FromFile(fi.FullName)
```

Das so gelesene Bild wird nun der Ressource hinzugefügt:

```
rw.AddResource (fi.Name, img)
```

Beim Zugriff auf die Textdatei wird erwartet, dass diese nach dem Muster »Schlüssel=Wert« aufgebaut ist. Geeignet für solche Zugriffe ist der *StreamReader*.

```
Dim sr As StreamReader = New StreamReader(fi.FullName)
```

Die Datei wird solange gelesen, bis keine Daten mehr verfügbar sind:

```
While (sr.Peek() <> -1)
```

Das Einlesen erfolgt zeilenweise:

```
sFile = sr.ReadLine()
```

Als Trennzeichen zwischen den Schlüsseln und Werten wird das erste Gleichheitszeichen verwendet:

```
Dim aFile() As String = sFile.Split("="c)
```

Die beiden Teile – links und rechts vom Gleichheitszeichen – werden dann der Ressource zugewiesen, nachdem sie sicherheitshalber noch von führenden oder anhängenden Leerzeichen befreit wurden:

```
rw.AddResource(aFile(0).Trim(), aFile(1).Trim())
```

**Kontrolle mit
resgen**

Damit ist das Programm auch schon fertig. Die Dateien liegen nun bereits als direkt verwendbare Ressourcen-Dateien vor. Zur Kontrolle bietet es sich an, diese mit *resgen* in lesbare XML-Dateien umzuwandeln. Dazu gehen Sie folgendermaßen vor:

- 1. Öffnen Sie die Visual Studio .NET Konsole über **START | ALLE PROGRAMME | MICROSOFT .NET FRAMEWORK SDK | MICROSOFT VISUAL STUDIO .NET | VISUAL STUDIO .NET TOOLS | VISUAL STUDIO .NET BEFEHLSZEILE**.
- 2. Wechseln Sie in das Verzeichnis, in dem die *.resources*-Dateien liegen.
- 3. Geben Sie Folgendes ein:

```
resgen culture.de.resources culture.de.resx
```

Ändern Sie die Namen entsprechend den tatsächlichen Bedingungen. Die Dateierweiterungen müssen zwingend verwendet werden.

Es entsteht nun eine *resx*-Datei, die exakt der *resources*-Datei entspricht.

Daten für data					
	name	value	comment	type	mimetype
►	date	Hier kommt das Datum	(Null)	(Null)	(Null)
	header1.jpg		(Null)	System.Drawing.Bitmap, System.Drawing, Version=1.0.3300.0, Cultu	application/x-microsoft.net.object.bytearray.base64
	header2.jpg		(Null)	System.Drawing.Bitmap, System.Drawing, Version=1.0.3300.0, Cultu	application/x-microsoft.net.object.bytearray.base64
	title	Das ist der Titel	(Null)	(Null)	(Null)

Abbildung 4.42: Die XML-Datei (*resx*) in der XML-Ansicht in Visual Studio

Sie können hier gut erkennen, wie die Datentypen und der MIME-Type eingerichtet wurde. Ein Blick in die XML-Daten selbst ist nur wenig mehr aufschlussreich, zeigt jedoch klar, welche mühevollen Arbeit das kleine Hilfsprogramm hier abgenommen hat.

[illegible]

Abbildung 4.43: Ausschnitt aus der erzeugten resx-Datei mit Text- und Bilddaten

Im nächsten Schritt geht es nun darum, die so erstellten Ressourcen-Dateien zu verwenden.

Verwendung von binären Ressourcen-Dateien

In diesem Abschnitt wird davon ausgegangen, dass bereits binäre Ressourcen-Dateien für alle benötigten Sprachen vorliegen, so wie es zuvor beschrieben wurde.

Die Basis der Programmierung bildet die Klasse `RessourceManager`, ebenfalls aus dem Namensraum `System.Resources`.

Vor den ersten Versuchen sollten Sie sich darüber im Klaren sein, wie Ihre Vorlagen mit Daten versorgt werden. Sie können prinzipiell jedes HTML-Element zu einem serverseitigen Steuerelement machen, ohne sich über die Funktionsweise dahinter Gedanken machen zu müssen. Das führt dann zu folgendem Code im HTML-Teil:

```
<h1 runat="server" id="title"></h1>
<asp:Label Runat="server" ID="showdate"/>
```

Listing 4.45: Für dynamische Ressourcen vorbereitete Vorlage
(Ausschnitt aus GlobalizationUseResource.aspx)

Angenommen, die binären Ressourcen-Dateien liegen im Unterverzeichnis */resource*, dann können Sie mit folgendem Code darauf zugreifen:

```
rm = ResourceManager.CreateFileBasedResourceManager ←  
("culture", Server.MapPath("resource"), Nothing)
```

Der Dateiname, nach dem gesucht wird, setzt sich aus dem Präfix (erstes Argument der statischen Methode `CreateFileBasedResourceManager`), dem Sprachcode oder einem davon abgeleiteten Fallback-Wert der aktuellen Kultur und der Dateierweiterung `resources` zusammen, jeweils durch Punkte getrennt.



Es ist dringend zu empfehlen, eine allgemeine Fallback-Datei mit dem Namen `<Präfix>.resources` anzulegen, die automatisch verwendet wird, wenn der Sprachcode nicht gefunden werden kann.

Das gesamte Programm zeigt das folgende Listing. Als Ausgangswert für die Wahl der Sprache wird der erste Wert der Browsereinstellungen verwendet:

```
Public Class GlobalizationUseResource
    Inherits System.Web.UI.Page
    Protected title As HtmlContainerControl
    Protected showdate As Label

    Private Shared rm As ResourceManager = Nothing

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Thread.CurrentThread.CurrentUICulture ←
            = CultureInfo.CreateSpecificCulture(Request.UserLanguages(0))
        rm ←
            = ResourceManager.CreateFileBasedResourceManager("culture", ←
                Server.MapPath("resource"), Nothing)
        title.InnerHtml = rm.GetString("title")
        showdate.Text = rm.GetString("date")
        rm.ReleaseAllResources()
    End Sub
End Class
```

Listing 4.46: Globalisierung basierend auf Ressourcen (Ausschnitt aus `GlobalizationUseResource.aspx.vb`, beachten Sie die Umbrüche der sehr langen Zeilen, beim Abtippen sind es innerhalb der Methode `Page_Load` nur fünf Zeilen)

Wie es funktioniert

Interessant ist die Zuweisung der tatsächlichen Werte an die HTML-Steuerelemente. Mit `GetString` haben Sie Zugriff auf alle Inhalte der Ressourcen-Datei:

```
title.InnerHtml = rm.GetString("title")
```

Abgesehen davon, dass in diesem Beispiel immer die aktuelle Kultur verwendet wird, können Sie dies mit dem zweiten Argu-

ment der Methode `GetString` überschreiben, beispielsweise um für eine Auswahlliste verfügbarer Sprachen die Optionen in den Landessprachen anbieten zu können.

Die hier vorgestellte Technik hat den Vorteil, dass die sprachlichen Fragmente außerhalb der Gestaltung und auch außerhalb des Codes liegen. Ändern Sie die Ressourcen-Dateien, ändert sich der Inhalt der Seiten sofort. Rechtschreibkorrekturen benötigen nun weder Code- noch Layout-Eingriffe. Gerade bei komplizierten Seiten ist dies ein enormer Vorteil.

Darüber hinaus sind die Zugriffe auf die Ressourcen auf eine Datei beschränkt. Diese Datei bleibt permanent geöffnet, sodass auch dies ein Leistungsvorteil darstellt. Allerdings haben Sie bei laufendem Betrieb keine Chance, auf die Ressourcen-Datei zuzugreifen, weil sie gesperrt ist. Wenn Sie erzwingen möchten, dass die Datei immer geschlossen wird, schreiben Sie folgenden Code ans Ende des Programms:

```
rm.ReleaseAllResources()
```

Sind Änderungen selten und ist die Systemleistung wichtig, lassen Sie die Zeile weg.

Für den dynamischen Zugriff auf die in den Ressourcen-Dateien versteckten Bilder muss man freilich etwas in die Trickkiste greifen. Ob der Aufwand lohnt, muss jeder selbst entscheiden. Tatsache ist aber, dass der Zugriff auf die Bilder immer mit einem Dateizugriff verbunden ist. Die bereits geöffnete Ressourcen-Datei wird jedoch nur ein Mal geöffnet, auch wenn darin Hunderte Bilder enthalten sind. Die Vereinfachung der Verwaltung mag ein Übriges dazu tun, tatsächlich Binärdaten in Ressourcen zu halten.

Damit Bilder dynamisch angezeigt werden, muss ein Programm die Bilddaten liefern. Entsprechend sieht der Aufruf im `<img>`-Tag aus:

```
<h1 runat="server" id="title"></h1>
<br/>
<br/>
<asp:Label Runat="server" ID="date"/>
```

*Listing 4.47: Anforderung eines Bildes aus der Ressource-Datei
(GlobalizationUseResourceImg.aspx)*


```

        Response.Write ("<a
href=\""GlobalizationUseResourceImg.aspx"> ␣
        GlobalizationUseResourceImg.aspx</a>")

    Else
        ' Start programm
        Response.ContentType = "image/jpeg"
        Thread.CurrentThread.CurrentUICulture ␣
            = CultureInfo.CreateSpecificCulture
        (Request.UserLanguages(0))
        Dim rm As ResourceManager ␣
            =
        ResourceManager.CreateFileBasedResourceManager("culture",␣
            Server.MapPath("resource"),Nothing)
        Dim img As System.Drawing.Image = CType(rm.GetObject( ␣
        Request.QueryString("image")), ␣
            System.Drawing.Image)

        Response.ClearContent ()
        img.Save (Response.OutputStream, ImageFormat.Jpeg)
        Response.End ()
        rm.ReleaseAllResources ()
    End If
End Sub
</script>

```

*Listing 4.48: Auslieferung eines Bildes aus der Ressourcen-Datei
(GlobalizationUseResourceImgSource.aspx)*

Der Zugriff auf die Ressource entspricht dem bereits für Text gezeigten Programm. Einzige Ausnahme bildet die Verwendung der Eigenschaft `CurrentUICulture` statt `CurrentCulture`, denn es geht hier nicht um die Einstellungen der Währung oder Zahlungsangaben, sondern um die Gestaltung der Benutzerschnittstelle (UI = User Interface, Benutzerschnittstelle). In der Praxis würde man wahrscheinlich beide Eigenschaften identisch setzen, weil die Lokalisierung der übrigen Werte fast immer dazu gehört. Sie können die eine Einstellung der anderen folgen lassen, indem Sie Folgendes schreiben:

```
Thread.CurrentThread.CurrentUICulturep; = Thread.CurrentThread.
CurrentCulture
```

Der Abruf erfolgt nun aber nicht mit `GetString`, sondern mit `GetObject`. Erzeugt wird wieder das ursprüngliche Bild, basierend auf der Auswahl des GET-Parameters:

```

System.Drawing.Image img = ␣
    CType(rm.GetObject(Request.QueryString("image"), ␣
        System.Drawing.Image)

```

**Wie es
funktioniert**

Damit der Browser mit den Daten etwas anzufangen weiß, ist der HTTP-Header Content-Type entsprechend zu setzen:

```
Response.ContentType = "image/jpeg"
```

Dann wird der aktuelle Puffer sicherheitshalber gelöscht:

```
Response.ClearContent ()
```

Im Anschluss wird das Bild dem Ausgabezeichenstrom übergeben:

```
img.Save (Response.OutputStream, ImageFormat.Jpeg)
```

Damit die Anzeige sofort erfolgt, wird die Übertragung beendet:

```
Response.End ()
```

Im Ergebnis erscheinen die passenden Bilder auf Basis der Browsereinstellungen. Beachten Sie, dass die Bilddaten nicht direkt von der Festplatte geholt werden. Änderungen daran wirken sich erst aus, wenn die Ressourcen-Datei neu erzeugt wurde. Freilich genügt es, das in Listing 4.44 gezeigte Programm ablaufen zu lassen.

Ressourcen in Assemblies speichern

Effiziente Speicherung durch Assemblies

Der bisher gezeigte dateibasierte Einsatz von Ressourcen ist nicht der einzige Weg. In großen Projekten kann die Verriegelung der Dateien zum Problem werden, wenn sich das ständige Schließen aus Leistungsgründen verbietet. Alternativ kann die Speicherung in Satellitenassemblies erfolgen. Prinzipiell wird für Ihr Projekt immer eine Hauptassembly erstellt. Normalerweise muss Sie das nicht kümmern, denn Visual Studio .NET erledigt das automatisch. Von ASP.NET kompilierte Programme landen automatisch im GAC (Global Assembly Cache), wo sie jederzeit gefunden werden können.

Wenn Sie nun mit Ressourcen arbeiten, wird für jede Kultur eine so genannte Satellitenassembly erstellt. Die Fallback-Ressource landet in der Hauptassembly. Mit Visual Studio .NET ist das sehr einfach. Die entsprechenden Compiler-Anweisungen werden automatisch verwendet, wenn Ressourcen-Dateien hinzugefügt werden. Dazu gehen Sie folgendermaßen vor:

1. Wählen Sie im Projektmanager das Projekt aus.
2. Suchen Sie im Kontextmenü den Eintrag HINZUFÜGEN und dort NEUES ELEMENT hinzufügen.
3. In der folgenden Auswahl suchen Sie das Symbol ASSEMBLY-RESSOURCENDATEI.
4. Vergeben Sie einen Namen und klicken Sie dann auf ÖFFNEN.

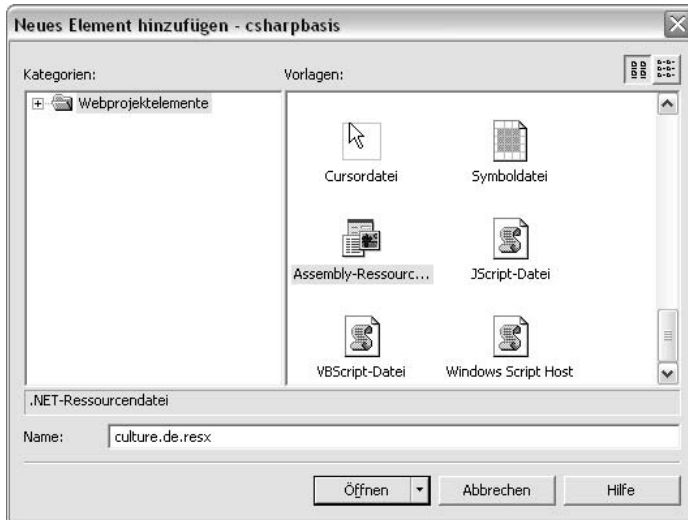


Abbildung 4.45: Erzeugen einer Ressourcen-Datei

Wenn Sie bereits über fertige Ressourcen-Dateien im *resx*-Format verfügen, können Sie diese ebenfalls verwenden.

Nach der Übersetzung des Projekts legt Visual Studio .NET unterhalb */bin* für jede Kultur ein Unterverzeichnis an und platziert dort die entsprechende DLL. Verwenden können Sie diese nun in Ihrem Code nach folgendem Schema:

```
Imports System.Resources
Imports System.Reflection
Imports System.Globalization
Imports System.Collections
Imports System.ComponentModel
Imports System.Drawing

Public Class GlobalizationAssemblies
    Inherits System.Web.UI.Page
    Protected title As HtmlContainerControl
    Protected showdate As Label
```

```

Private Shared rm As ResourceManager = Nothing

Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Dim ci As CultureInfo _
        = CultureInfo.CreateSpecificCulture( _
            Request.UserLanguages(0))
    rm = New ResourceManager("Addison.VBNet.Basis.culture", _
        System.Reflection.Assembly.GetExecutingAssembly())
    title.InnerHtml = rm.GetString("title", ci)
    showdate.Text = rm.GetString("date", ci)
    rm.ReleaseAllResources()
End Sub
End Class

```

Listing 4.49: Entnahme von Ressourcen-Informationen aus einer Satelliten-Assembly (GlobalizationsAssemblies.aspx.vb)

**Achtung,
Namensraum
angeben!**

Eine Falle stellt die Angabe des Präfixes dar. Sie müssen hier den Namensraum Ihrer Applikation mit angeben, andernfalls findet der Ressourcenmanager die Ressource in der Hauptassembly nicht:

```
Addison.VBNet.Basis.culture
```

Der Name »Addison.VBNet.Basis.« ist im Beispiel der Namensraum. Das Beispiel zeigt außerdem eine alternative Form des Zugriffs auf die Kultur. Statt den gesamten Thread zu ändern, wird hier die Kultur bei jedem Abruf der Ressource angegeben:

```
title.InnerHtml = rm.GetString("title", ci)
```



Wenn Sie nicht mit Visual Studio .NET arbeiten, können Sie selbstverständlich auch Ressourcen-Dateien in Satellitenassemblies verpacken. Nutzen Sie das Werkzeug al (Assembly Linker) zum Verbinden der Satelliten mit der Hauptassembly und die Option /res des Compilers vbc beim Übersetzen der Hauptdatei. Außerdem müssen Sie die fertigen Assemblies selbst in die entsprechenden Verzeichnisse kopieren (/bin/<culture>).

Praxistipps

In der Praxis müssen Sie, bevor aufwändige Lokalisierungen erfolgen, ein paar Fragen beantworten:

**Vor der
Entwicklung der
Ressourcen**

- ▶ Wer übersetzt die Texte?
- ▶ Wer übersetzt Bilddaten?
- ▶ Passt die Gestaltung zu allen Sprachen?
- ▶ Liegen auch Artikel- oder Inhaltsdaten in den passenden Sprachen vor?
- ▶ Wie gehen Sie mit Sprachanforderungen um, für die Sie keine Übersetzung haben?
- ▶ Beherrschen Sie die Aktualisierung Ihrer Applikation noch, wenn Sie viele Sprachen haben?

Wenn diese Fragen geklärt sind, bietet ASP.NET alles, um mehrsprachige Seiten professionell umzusetzen.

Nachdem die Umsetzung gelungen ist und alle Ressourcen funktionieren, bleiben Überlegungen zur Steigerung der Systemleistung. Dazu sollten Sie sich mit der Ablaufsteuerung beim Laden von Anwendungen auseinandersetzen. Ausführliche Informationen dazu finden Sie im Abschnitt 8.6.2, »Die Datei *global.asax*« ab Seite 808. An dieser Stelle soll nur ein kleiner Ausblick gegeben werden.

Im ereignisgesteuerten Modell der Verarbeitung von Anforderungen werden bestimmte Methoden zu genau definierten Zeitpunkten ausgeführt. Sie können diese Methoden mit Code hinterlegen, um Aktionen an die Ereignisse zu binden. Die Definition zentraler Ereignisbehandlungsmethoden erfolgt in der Datei *global.asax*, die jeder Applikation zugeordnet werden kann. So gibt es eine Methode `Application_OnStart`, die beim Start der Applikation mit der ersten Anforderung ausgelöst wird. Erst ein Stoppen des IIS-Dienstes oder ein Neustart des Computers startet die Applikation neu. Wenn Sie nun den Abruf der Ressourcen in den Zeitpunkt des Applikationsstarts verlegen und Applikationsvariablen zur Ablage des `ResourceManager`-Objekts verwenden, wird die Systemleistung verbessert. Applikationsvariablen speichern Daten im Hauptspeicher gemeinsam für alle Anforderungen bzw. Sitzungen. Auch wenn 1.000 Besucher gleichzeitig auf Ihre Homepage gehen, wird die Ressourcen-Datei nun nur einmal geladen. Praktisch sieht das folgendermaßen aus:

```

Sub Application_Start ()
    Application ("AllRecourses") ←
        = System.Resource.ResourceManager. ←
            CreateFileBasedResourceManager("culture", ←
                Server.MapPath("resource"), Nothing)
End Sub

```

Listing 4.50: Laden des Ressourcen-Managers beim Start der Applikation

Die Auswahl der Kultur muss nun natürlich noch an die Sitzung gekoppelt werden, denn dieser Vorgang ist für jeden Benutzer individuell:

```

Private Sub Application_BeginRequest(ByVal sender As Object, ←
    ByVal e As EventArgs)
    Try
        Thread.CurrentThread.CurrentCulture = ←
            CultureInfo(Request.UserLanguages(0))
    Catch
        Thread.CurrentThread.CurrentCulture = ←
            CultureInfo("de-DE")
    End Try
End Sub

```

Listing 4.51: Zuweisen der gewünschten Sprache oder Auslösen eines Fallback-Mechanismus, wenn die Sprache nicht unterstützt wird

Innerhalb Ihrer Seiten müssen Sie nun noch Zugriff auf die bereits geladenen Ressourcen erhalten. Dazu wird, vorzugsweise in der Methode `Page_Init`, folgender Code verwendet:

```

Dim rm As ResourceManager ←
    = CType(Application("AllResources"), ResourceManager)

```

Diese Zeile holt die Applikationsvariable zurück und erzeugt ein Objekt *rm* mit dem benötigten Typ `ResourceManager`.

4.10.3 Konfiguration in *web.config*

In der Datei *web.config* können die Globalisierungseinstellungen in einigen Punkten zentral festgelegt werden. Dazu ist innerhalb des Zweiges `<system.web>` ein Abschnitt `<globalization>` einzufügen. Folgende Attribute können Sie in diesem Tag verwenden:

► `requestEncoding`

Bestimmt die Kodierung, die ASP.NET für eingehende Anfragen erwartet. Der Standard ist UTF-8.

► `responseEncoding`

Bestimmt die Kodierung der gesendeten Daten. Der Standard ist UTF-8.

► `fileEncoding`

Wenn Dateien ohne explizite Angabe einer Kodierung geschrieben werden, bestimmt dieser Parameter die Einstellung. Der Standard ist UTF-8.

► `culture`

Bestimmt die Kultur in der Form »de-DE«, wie im Abschnitt weiter oben beschrieben.

► `uiCulture`

Bestimmt die Kultur die Steuerelemente bei der Ausgabe verwenden, wenn Ressource-Dateien eingesetzt werden.

4.11 Zugriff auf das Dateisystem

Für den Zugriff auf das Dateisystem über das Web gibt es viele sinnvolle Anwendungen. Das .NET-Framework bietet eine reichhaltige Unterstützung dafür – nicht nur in speziellen Klassen, sondern auch durch darauf abgestimmte Methoden und Eigenschaften anderer Klassen, beispielsweise zur Analyse von Dateidaten.

4.11.1 Einführung

Einige Einsatzbeispiele, wofür Zugriffe auf das Dateisystem erforderlich sein können, sind:

► *Protokolldateien*

Protokollieren Sie Vorgänge in Ihrer Applikation in eine Textdatei.

► *Ablage von Formulardaten*

Legen Sie den Inhalt eines Formulars als Textdatei ab.

► *News und Tipp des Tages*

Zeigen Sie auf Ihrer Website News an, die Sie als Text- oder HTML-Datei auf dem Server ablegen.

Zugriff auf den Namensraum System.IO

Namensraum aktivieren

Der Zugriff auf den Namensraum ist in ASP.NET standardmäßig nicht aktiviert. Sie müssen deshalb Ihren Programmen folgende Direktive voranstellen:

```
<% @Import Namespace="System.IO" %>
```

Wenn Sie mit hinterlegtem Code arbeiten, ist diese Anweisung im Kopf des Programms zu ergänzen:

```
Imports System.IO
```

Dieser Namensraum enthält eine große Anzahl Klassen, die hier nur teilweise vorgestellt werden können. Wesentlich sind:

► **Directory, DirectoryInfo**

Diese Klassen erlauben den Zugriff auf Verzeichnisse über statische Methoden und enthalten Informationen über ein Verzeichnis.

► **Path**

Path dient der Berechnung und Manipulation von Pfaden, wobei in den meisten Fällen nicht reflektiert wird, ob der Pfad wirklich existiert.

► **File, FileInfo**

Diese Klassen erlauben den Zugriff auf Dateien über statische Methoden und enthalten Informationen über eine Datei.

► **StreamReader, StreamWriter, BinaryReader, BinaryWriter**

Lesen und Schreiben von Textdaten bzw. Binärdaten aus bzw. in Dateien.

In System.IO werden außerdem auch die Klassen `StringWriter` bzw. `StringReader` definiert, die Daten in Zeichenketten schreiben bzw. aus diesen lesen. Da dies nicht unmittelbar etwas mit dem Dateizugriff zu tun hat, werden diese Klassen in Abschnitt »Die Klassen `StringWriter` und `StringBuilder` im Detail« ab Seite 262 betrachtet. Ebenso nimmt die Klasse `MemoryStream` eine Sonderstellung ein. Hier geht es um das Lesen und Schreiben in Speicherbereiche.

4.11.2 Zugriff auf Verzeichnisse und Dateien

An erster Stelle soll der Zugriff auf Verzeichnisse gezeigt werden. Das folgende Beispiel zeigt ein Formular, in das ein Pfad einge-

geben werden kann. Ist der Pfad vorhanden, wird eine Liste der Verzeichnisse und Dateien darin angezeigt. Das folgende Beispiel nutzt bereits die Datenbindungssyntax, statt einer Datenbank kommen hier Dateilisten zur Anzeige. Eine ausführliche Erklärung folgt im Zusammenhang mit der Ausgabe von Daten. Ebenso gelangen bereits einige Server-Steuerelemente zum Einsatz. Zum Verständnis des Datei- und Verzeichniszugriffs sind genaue Kenntnisse darüber nicht notwendig.

```
<body MS_POSITIONING="GridLayout">
<form id="FileReadDir" method="post" runat="server">
  <asp:textbox runat="server" id="Verzeichnisname"/>
  <asp:button runat="server" id="senden" text="Anzeigen"/>
</form>
<h2>Verzeichnisse</h2>
<asp:Repeater Runat="server" ID="Verzeichnisse">
  <ItemTemplate>
    <%# Container.DataItem.Name %>
    <br/>
  </ItemTemplate>
</asp:Repeater>
<h2>Dateien</h2>
<asp:Repeater Runat="server" ID="Dateien">
  <ItemTemplate>
    <%# Container.DataItem.Name %>
    (<%# Container.DataItem.Length %>
     Bytes)
    <br/>
  </ItemTemplate>
</asp:Repeater>
<asp:Label Runat="server" ID="Ausgabe"/>
</body>
```

Listing 4.52: Formular zur Anzeige von Verzeichnisinformationen (FileReadDir.aspx)

Verwendet werden hier mehrere Server-Steuerelemente. Das Steuerelement `Repeater` wiederholt die übergebene Datenliste entsprechend der vorgegebenen Gestaltung. Eine eingehende Betrachtung der verwendeten Steuerelemente finden Sie in Abschnitt 6.4, »Web Server-Steuerelemente (Web Server Controls)« ab Seite 535. Wichtiger ist im Zusammenhang mit Verzeichnisdaten der Zugriff auf die Informationen über Dateien und Verzeichnisse. Die Datenbindungssyntax ist typisch für wiederholende Steuerelemente und wird genauer in Abschnitt 6.6.2, »Aufbau der Vorlagen in Daten-Steuerelementen« ab Seite 570 betrachtet. Bei jeder Zuweisung eines einzelnen Datenelements wird dies als `Contai-`

Wie es funktioniert

ner.DataItem verfügbar gemacht. Container enthält alle Daten. Dann folgt der Zugriff auf das jeweilige Element mit DataItem, dem beispielsweise die Eigenschaft Name folgt (das zugrunde liegende Objekt muss natürlich über die Eigenschaft verfügen, damit die implizite Konvertierung gelingt):

```
<# Container.DataItem.Name %>
```

Die Klasse FileReadDir erledigt dann die eigentliche Arbeit, nämlich das Zuweisen der Daten an die Steuerelemente. Beachten Sie, dass Sie am Anfang den Namensraum System.IO nicht vergessen.

```
Public Class FileReadDir
    Inherits System.Web.UI.Page
    Protected Verzeichnisname As TextBox
    Protected Verzeichnisse As Repeater
    Protected Dateien As Repeater
    Protected Ausgabe As Label

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Page.IsPostBack Then
            Try
                Dim dinfo As DirectoryInfo = New
                DirectoryInfo(Server.MapPath(Verzeichnisname.Text))
                Verzeichnisse.DataSource = dinfo.GetDirectories()
                Verzeichnisse.DataBind()
                Dateien.DataSource = dinfo.GetFiles()
                Dateien.DataBind()
                Catch nichtgefunden As DirectoryNotFoundException
                    Ausgabe.Text = "<span style=""color:red"">Verzeichnis
nicht gefunden: "
                    Ausgabe.Text += nichtgefunden.Message + "</span>"
                End Try
            End If
        End Sub
End Class
```

Listing 4.53: Lesen eines Verzeichnisses (FileReadDir.aspx.vb)

Wie es funktioniert

Beim Zugriff auf Verzeichnisse tritt oft das Problem auf, dass der Pfad nicht gefunden werden kann oder die Zugriffsrechte nicht ausreichend sind. Um robuste Applikationen zu schreiben, sollten die Ausnahmen abgefangen werden, die die Klassen erzeugen. Ideal ist dafür die *Try-Catch*-Anweisung. Der *Try*-Zweig wird zuerst ausgeführt, bis eine Ausnahmebedingung auftritt.

Der Zugriff auf Verzeichnisinformationen erfolgt mit der Klasse `DirectoryInfo`. In `Verzeichnisname.Text` steht der Inhalt des Eingabefeldes des Formulars. Die Anweisung `Server.MapPath` sichert die Ermittlung des physischen Pfades zu den virtuellen des IIS:

**Verzeichnis-
informationen**

```
Dim dinfo As DirectoryInfo ←  
    = New DirectoryInfo(Server.MapPath(Verzeichnisname.Text))
```

Die Unterverzeichnisse können nun über eine Kollektion ermittelt werden, die `GetDirectories` zurückgibt. Für jedes Unterverzeichnis wird wiederum ein Objekt vom Typ `DirectoryInfo` erzeugt. Diese Kollektion der Objekte wird dem Repeater-Steuerelement zugewiesen:

```
Verzeichnisse.DataSource = dinfo.GetDirectories()
```

Mit der Datenbindung erfolgt die Anzeige.

```
Verzeichnisse.DataBind()
```

Die Auswahl der gewünschten Informationen aus `DirectoryInfo` erfolgt über den Container.

Dateiinformationen setzen voraus, dass ein Zugriff auf eine Datei vorliegt. Mit der Methode `GetFiles` erhalten Sie eine Kollektion von `FileInfo`-Objekten. Die folgende Zeile weist die Liste dem Repeater-Steuerelement zu:

**Dateiinforma-
tionen**

```
Dateien.DataSource = dinfo.GetFiles()
```

Der Zugriff auf die gewünschten Informationen erfolgt über den Container. Dort wird der `FileInfo`-Typ erzeugt und dann kann auf alle Eigenschaften zugegriffen werden.

Falls die Anzeige des Verzeichnisses misslingt, wird eine Ausnahme vom Typ `DirectoryNotFoundException` erzeugt. Mit `Catch` können Sie diese abfangen:

Fehler abfangen

```
Catch nichtgefunden As DirectoryNotFoundException
```

Interessant ist eventuell die Fehlermeldung, die die Klasse hier erzeugt:

```
ausgabe.Text += nichtgefunden.Message
```

Das vorgestellte Beispiel zeigte, wie Verzeichnis- und Dateiinformationen mit wenigen Schritten zur Anzeige gebracht werden können. Die große Anzahl an Eigenschaften erlaubt nun komfortable Lösungen für Dateioperationen.



Abbildung 4.46: Ausgabe des Inhalts eines Verzeichnisses als Liste der Unterverzeichnisse und Dateien

Methoden und Eigenschaften für den Verzeichniszugriff

Die wichtigsten Methoden der Klasse `Directory` zeigt die folgende Tabelle. Alle Methoden sind statisch, werden also in der Form `Directory.Methode()` verwendet, ohne dass die Klasse instanziiert werden muss.

Eigenschaft	Bedeutung
<code>GetDirectories</code>	Gibt eine Kollektion der enthaltenen Verzeichnisse zurück: ▶ <code>GetDirectories(pfad)</code> ▶ <code>GetDirectories(pfad, platzhalter)</code>
<code>GetCreationTime</code> <code>SetCreationTime</code>	Das Datum der Erzeugung lesen und schreiben
<code>GetFiles</code>	Gibt eine Kollektion der Dateien zurück: ▶ <code>GetFiles(pfad)</code> ▶ <code>GetFiles(pfad, platzhalter)</code>

Tabelle 4.11: Methoden der Klasse `Directory`

Eigenschaft	Bedeutung
GetParent	Name des übergeordneten Verzeichnisses
CreateDirectory	Erzeugt ein Verzeichnis
SetDirectory	Setzt das aktuelle Verzeichnis
Move	Verschiebt ein Verzeichnis mit dem gesamten Inhalt: ► <code>Move(pfad_quelle, pfad_ziel)</code>
Delete	Löscht ein Verzeichnis und – wenn <code>subdirs</code> gleich <code>True</code> ist – auch den Inhalt: ► <code>Delete(pfad[, subdirs])</code>
Exists	Prüft, ob ein Verzeichnis existiert

Tabelle 4.11: Methoden der Klasse *Directory* (Forts.)

Die Kollektionen implementieren die Schnittstelle *ICollection* und können direkt an *ArrayList*, *Hashtable* und *Array* und – wie bereits gezeigt – an alle wiederholenden Steuerelemente übergeben werden.



Der Zugriff auf die Informationen über ein Verzeichnis erfolgt mit *DirectoryInfo*. Im Gegensatz zu *Directory* sind die Methoden nicht statisch und es gibt viele Eigenschaften. Die Anwendung erfolgt durch direkte Instanziierung des Objekts:

```
Dim dinfo As DirectoryInfo = New DirectoryInfo("pfadangabe")
```

Wenn Sie einen virtuellen Pfad in einen physischen umrechnen müssen, verwenden Sie die Methode *Server.MapPath*.



Es stehen hier ähnliche Methoden zur Verfügung wie bei der Klasse *Directory*. Hinzu kommen Eigenschaften, die Auskunft über Attribute und Namen geben. Die wichtigsten finden Sie nachfolgend:

**Verzeichnis-
eigenschaften**

Eigenschaft	Beschreibung
Attributes	Attribute
CreationTime	Datum der Erzeugung
FullName	Pfad und Name
Parent	Übergeordnetes Verzeichnis
Root	Stammverzeichnis

Tabelle 4.12: Wichtige Eigenschaften der Klasse *DirectoryInfo*

Dateiattribute Die Attribute sind eine Aufzählung mit dem Namen `FileAttributes`. Diese Aufzählung ist intern ein Bitfeld. Die Erkennung eines spezifischen Attributes kann durch die Formel `Attribut & FileAttributes.AttributeName` erfolgen. Das im letzten Beispiel gezeigte Programm kann leicht um eine Auswertung der Attribute erweitert werden. Sie finden das komplette Programm unter dem Namen *FileReadDirAttr.aspx*. Nachfolgend die Erweiterung gegenüber dem letzten Beispiel:

```
Public Class MyFileInfo
    Public Shared Function ShowAttributes(ByVal fa As
FileAttributes) As String
        Dim sa As String = ""
        If (fa And FileAttributes.Archive) > 0 Then sa += "A"
        If (fa And FileAttributes.Compressed) > 0 Then sa += "C"
        If (fa And FileAttributes.Directory) > 0 Then sa += "D"
        If (fa And FileAttributes.Hidden) > 0 Then sa += "H"
        If (fa And FileAttributes.System) > 0 Then sa += "S"
        If (fa And FileAttributes.ReadOnly) > 0 Then sa += "R"
        Return sa
    End Function
End Class
```

*Listing 4.54: Methode zur Ermittlung der Verzeichnis- und Dateiattribute
(Ausschnitt aus FileReadDirAttr.vb)*

Wie es funktioniert

Der Zugriff aus der Vorlage heraus ist etwas trickreich, soll hier aber exemplarisch gezeigt werden. Sie können in der Datenbindungssyntax natürlich Methoden oder Eigenschaften vorhandener Klassen verwenden. Sie können aber keine Instanzen erzeugen. Das setzt voraus, dass Methoden direkt, ohne Instanziierung verwendbar sein müssen. Dies wird im Beispiel mit dem Schlüsselwort `Shared` erreicht. Der Aufruf – mit vollständiger Referenzierung des Namensraumes – erfolgt in der HTML-Vorlage folgendermaßen:

```
<%#
Addison.VBNet.Basis.MyFileInfo.ShowAttributes
(Container.DataItem.Attributes)
%>
```

Lesen Sie diesen Ausdruck von innen nach außen. Zuerst erfolgt der Zugriff auf den `Container` und dort auf das aktuelle Element mit `DataItem`. Dann wird der Typ mit `(System.IO.FileInfo)` festgelegt. Von diesem Objekt wird nun die Eigenschaft `Attributes` abgerufen, die ein Objekt vom Typ `FileAttributes` zurückgibt. Dieses

wird an die statische Methode `ShowAttributes` übergeben. Deren Rückgabewert gelangt dann zur Anzeige.

Die Ausgabe sieht dann – je nach Inhalt des Verzeichnisses – folgendermaßen aus:

A = Archiv
D = Directory
H = Hidden
S = System
R = Read Only
C = Compressed

Dateien
anchorstate.aspx (1911 Bytes) AH
application_getvar.aspx (540 Bytes) AHR
application_setvar.aspx (492 Bytes) AHR
arrays_foreach.aspx (328 Bytes) AC
arrays_formulti.aspx (561 Bytes) ACH
arrays_string.aspx (397 Bytes) AHR
arrays_string2.aspx (343 Bytes) AHR
array_methods1.aspx (490 Bytes) AH
array_methods2.aspx (619 Bytes) A
array_methods3.aspx (894 Bytes) A
basisscript.aspx (211 Bytes) AH
books.cs (1393 Bytes) A
books2.cs (1210 Bytes) A
button_style.aspx (1139 Bytes) AC
calendar.aspx (766 Bytes) AC
class_construct.aspx (798 Bytes) AR
class_events.aspx (2038 Bytes) AR
class_inherit.aspx (959 Bytes) A
class_overload.aspx (984 Bytes) A
class_overloadops.aspx (1140 Bytes) A

Abbildung 4.47: Anzeige von Dateiattributen

Die Methoden der Klasse `DirectoryInfo` erlauben ähnliche Operationen wie bei `Directory`. Eine Auswahl zeigt die folgende Tabelle:

Methode	Beschreibung
Create	Verzeichnis bzw. mehrere Unter- verzeichnisse erzeugen
CreateSubdirectory	
Delete	Verzeichnis löschen
MoveTo	Verzeichnis verschieben: ► <code>MoveTo (ziel)</code>
GetFiles	Kollektion der Dateien
Refresh	Aktualisiert Statusdaten
GetFiles	Kollektion der Dateien

Tabelle 4.13: Methoden der Klasse `DirectoryInfo`

Das Prinzip der Anwendung wurde bereits im ersten Beispiel dieses Abschnitts gezeigt. Beachten Sie bei `MoveTo`, dass die Angabe der Quelle nicht erforderlich ist. Das `DirectoryInfo`-Objekt enthält

bereits die Pfadinformation der Quelle. Die Methode `Move` der Klasse `Directory` verlangt dagegen die Angabe von Quelle und Ziel.

Zugriff auf Laufwerke

Eine explizite »Drive«-Klasse existiert nicht. Tatsächlich ist dies kein Mangel, denn schon seit einiger Zeit spielen Laufwerke nur noch eine untergeordnete Rolle im Dateisystemzugriff. Mit der zunehmenden Vernetzung erscheinen lokale Laufwerksnamen wie ein Relikt aus alter Zeit. Grundsätzlich erlauben alle Verzeichniszugriffe die Spezifizierung von Laufwerksnamen. Darüber hinaus ist die Nutzung der Methode von `GetLogicalDrives` möglich, die alle Laufwerke liefert, nicht nur die physischen. Zurückgegeben wird ein Zeichenketten-Array; das Format der Laufwerksnamen entspricht folgendem Muster: »C:\«.

Methoden und Eigenschaften für den Dateizugriff

Der Zugriff auf Dateien basiert auf zwei vergleichbaren Klassen: `File` mit ausschließlich statischen Methoden für den direkten Zugriff auf Dateien und `FileInfo` für den Zugriff über Objekte. Eine Auswahl wichtiger Methoden zeigt die folgende Tabelle:

Methode	Bedeutung
<code>GetAttributes</code>	Aufzählung der Dateiattribute
<code>Get<XXX>Time</code>	Ermittelt (Get) bzw. setzt (Set) die Dateizeiten, wobei für <XXX> Folgendes stehen kann: ► <code>Creation</code> – Datum der Erzeugung ► <code>LastAccess</code> – Datum des letzten Zugriffs ► <code>LastWrite</code> – Datum des letzten Schreibvorgangs
<code>Set<XXX>Time</code>	
<code>Open</code>	Öffnet eine Datei zum Lesen oder Schreiben
<code>OpenRead</code>	Zugriff auf eine Datei zum Lesen, Schreiben oder als Textdatei im UTF-8-Format
<code>OpenWrite</code>	
<code>OpenText</code>	
<code>Create</code>	Erzeugt eine Datei
<code>SetDirectory</code>	Setzt das aktuelle Verzeichnis

Tabelle 4.14: Methoden der Klasse `File`

Methode	Bedeutung
Move	Verschiebt eine Datei mit dem gesamten Inhalt: ► <code>Move(quelle, ziel)</code>
Delete	Löscht eine Datei
Exists	Prüft, ob eine Datei existiert

Tabelle 4.14: Methoden der Klasse `File` (Forts.)

In Abschnitt 4.11.4, »Dateien erzeugen und schreiben« ab Seite 341 finden Sie Informationen darüber, wie Dateien erzeugt, geschrieben und gelesen werden können.

Eine Instanz der Klasse `FileInfo` erlaubt den Zugriff auf Dateiinformationen:

FileInfo

```
Dim dinfo As FileInfo = New FileInfo("pfad/dateiname")
```

Es stehen hier ähnliche Methoden zur Verfügung wie bei `File`. Hinzu kommen Eigenschaften, die Auskunft über Attribute und Namen geben. Die wichtigsten finden Sie nachfolgend:

Eigenschaft	Beschreibung
Attributes	Attribute
CreationTime	Datum der Erzeugung
FullName	Pfad und Name
Name	Dateiname
Parent	Übergeordnetes Verzeichnis
Length	Größe der Datei in Byte
Root	Stammverzeichnis

Tabelle 4.15: Wichtige Eigenschaften der Klasse `FileInfo`

Die Eigenschaft `Attributes` basiert auf derselben Aufzählung wie die bei den Verzeichnissen beschriebene. Das letzte Beispiel verwendete diese bereits für Dateien.

**Datei-
eigenschaften**

Die Methoden erlauben ähnliche Operationen wie bei `File`. Eine Auswahl zeigt die nächste Tabelle:

Methode	Beschreibung
Create	Datei erzeugen
CreateText	Datei und <code>StreamWriter</code> darauf erzeugen
Delete	Datei löschen
MoveTo	Datei verschieben: ► <code>MoveTo(pfad_ziel)</code>
Open	Universell öffnen zum Lesen oder Schreiben
OpenRead	Zugriff auf eine Datei zum Lesen, Schreiben oder als Textdatei im UTF-8-Format
OpenWrite	
OpenText	
Refresh	Status erneuern

Tabelle 4.16: Methoden der Klasse `FileInfo`

In Abschnitt 4.11.4, »Dateien erzeugen und schreiben« ab Seite 341 finden Sie Informationen darüber, wie Dateien erzeugt, geschrieben und gelesen werden können.

4.11.3 Ausnahmebehandlung bei Dateioperationen

**Dateizugriffsfehler
abfangen**

Operationen mit Verzeichnissen und Dateien scheitern oft an mangelnden Rechten, falschen Pfadangaben oder Störungen der Speichergeräte. In jedem dieser Fälle erzeugt die .NET-Laufzeitumgebung eine Ausnahme. Am Anfang wurde bereits gezeigt, wie Sie diese mit *Try-Catch* abfangen können:

```
Try
    // Versuche Zugriff
Catch Ausnahmevariable As Ausnahmetyp)
    // Behandle Ausnahme
Catch (Andere_Ausnahmevariable As Anderer_Ausnahmetyp)
    // Behandle andere Ausnahme
Finally
    // Aktionen, die immer ausgeführt werden sollen
End Try
```

Der *Finally*-Teil ist optional. Hier platzieren Sie Code, der immer ausgeführt werden soll, egal ob eine Ausnahme auftrat oder nicht. Um nun eine spezifische Ausnahme abfangen und behandeln zu können, müssen Sie die Art des Fehlers entsprechend definieren:

`catch e As DirectoryNotFoundException`

Diese Zeile führt dazu, dass der Fehler »Verzeichnis nicht gefunden« abgefangen wird. Das Fehler-Objekt vom Typ `DirectoryNotFoundException` wird in `e` gespeichert. Bevor Sie jedoch feinkörnige Fehlerbehandlungen schreiben, müssen Sie die im aktuellen Kontext möglichen Ausnahmen kennen:

Ausnahmetyp	Behandelte Ausnahme
<code>DirectoryNotFoundException</code>	Verzeichnis nicht gefunden
<code>FileNotFoundException</code>	Datei nicht gefunden
<code>EndOfStreamException</code>	Ende der Datei erreicht
<code>PathTooLongException</code>	Pfad zu lang (maximal sind 248 Zeichen erlaubt)
<code>SecurityException</code> <code>UnauthorizedAccessException</code>	Mangelnde oder völlig fehlende Zugriffsrechte
<code>ArgumentException</code> <code>ArgumentNullException</code>	Ein Argument ist falsch oder <i>null</i>
<code>IOException</code>	Allgemeiner IO-Fehler. Tritt meist beim Löschen von Nur-Lese-Objekten auf

Tabelle 4.17: Im Zusammenhang mit Verzeichnis- und Dateioperationen auftretende Ausnahmen

Nicht in allen Fällen können alle Ausnahmen auftreten. Besonders wichtig sind: `DirectoryNotFoundException`, `FileNotFoundException` und `SecurityException`.

Die Klassen `SecurityException` und `UnauthorizedAccessException` stammen aus dem Namensraum `System.Security`. Sie müssen diesen entsprechend einbinden, wenn Sie derartige Ausnahmen verarbeiten möchten:

`Imports System.Security`

Wichtige Ausnahmen

Namensraum der Sicherheitsklassen

4.11.4 Dateien erzeugen und schreiben

Der Zugriff auf Verzeichnisse und Dateien dient meist nur dem Zweck, lesend und schreibend auf Dateien zuzugreifen. Das folgende Beispiel zeigt, wie mit jedem Aufruf einer Seite Informationen über die Verbindung in eine Datei geschrieben werden – eine Protokolldatei entsteht. Zur Kontrolle wird die Datei gleich angezeigt. Zuerst die *aspx*-Datei zur Ausgabe:

```

<html>
  <head>
    <title>Protokolldatei</title>
  </head>
  <body>
    <h3>Ausgabe einer Protokolldatei</h3>
    <pre><asp:label runat="server" id="ausgabe"/></pre>
  </body>
</html>

```

Listing 4.55: Ausgabe von Protokollinformationen (FileLog.aspx)

Die eigentliche Arbeit erledigt wieder eine Code-Datei:

```

Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls
Imports System.IO
Imports System.Security

Public Class FileLog
  Inherits Page
  Public ausgabe As Label

  Private Sub ShowFile(ByVal logfile As String)
    Dim test As StreamReader = File.OpenText(logfile)
    ausgabe.Text += test.ReadToEnd()
    test.Close()
  End Sub

  Private Sub Page_Load(ByVal sender As Object, _
    ByVal args As EventArgs) _
    Handles MyBase.Load
    Dim log As String = Server.MapPath(".\logs\log.txt")
    Dim logeintrag As String ←
      = Request.ServerVariables("REMOTE_ADDR") ←
      & " " & DateTime.Now
    If File.Exists(log) Then
      Dim logs As StreamWriter = File.AppendText(log)
      logs.WriteLine(logeintrag)
      logs.Close()
      ShowFile(log)
    Else
      Try
        Dim logs As StreamWriter = File.CreateText(log)
        logs.WriteLine(logeintrag)
        logs.Close()
        ShowFile(log)
      Catch
      End Try
    End If
  End Sub
End Class

```

```

        Catch e As DirectoryNotFoundException
            ausgabe.Text = "<span style=""color:red""> ⚡
                Verzeichnis nicht vorhanden: "
            ausgabe.Text += e.Message + "</span>"
        Catch e As UnauthorizedAccessException
            ausgabe.Text = "<span style=""color:red""> ⚡
                Kein Zugriff: "
            ausgabe.Text += e.Message + "</span>"
        End Try
    End If
End Sub
End Class

```

Listing 4.56: Programm zum Schreiben und Lesen einer Protokolldatei (FileLog.aspx.vb)

Das Programm beginnt mit der Festlegung des Pfades. Die Protokolldatei soll innerhalb der Webapplikation gespeichert werden. Aus Sicht des Frameworks ist das Stammverzeichnis `%systemroot%\system32`. Dass es sich um eine Webapplikation handelt, ist erstmal nicht bekannt. Schließlich werden dieselben Klassen auch in der Windows-Programmierung verwendet. In Abschnitt 8.2, »Die Welt der Standardobjekte« ab Seite 747 werden die Klassen vorgestellt, die in ASP.NET den Zugriff auf den Webserver erlauben. Dazu gehört auch `Server`. Diese Klasse enthält eine Methode `MapPath`, die einen lokalen oder virtuellen Pfad einer Webapplikation in eine physische Pfadangabe umwandelt:

Wie es funktioniert

```
Dim log As String = Server.MapPath(".\logs\log.txt")
```

Dann wird der Eintrag, der das Protokoll füllen soll, erzeugt. Verwendet wird hier die IP-Adresse des Browsers und das aktuelle Datum einschließlich der Zeit:

```
Dim logeintrag As String ⚡
    = Request.ServerVariables("REMOTE_ADDR") & " " & DateTime.Now
```

Zuerst wird geprüft, ob die Datei schon existiert:

```
If File.Exists(log) Then
```

Jetzt kann jetzt der Eintrag hinzugefügt werden. Dazu wird ein `StreamWriter`-Objekt erzeugt:

```
Dim logs As StreamWriter = File.AppendText(log)
```

In dieses wird mit `WriteLine` der Protokolleintrag geschrieben:

```
logs.WriteLine(logeintrag)
```

Dann wird die Datei geschlossen und zur Kontrolle angezeigt. Die Vorstellung der Anzeigefunktion `ShowFile` erfolgt weiter unten.



Sie sollten Dateien immer explizit und so schnell wie möglich schließen. In Mehrbenutzerumgebungen führen offene Dateien zu Fehlern oder Leistungseinbußen. Vertrauen Sie nicht darauf, dass offene Dateien am Ende der Seite sofort automatisch geschlossen werden, auch wenn dies beispielsweise in einer Testumgebung der Fall sein dürfte.

Falls die Datei noch nicht existierte, wird der zweite Teil der *If-Then*-Anweisung ausgeführt. Der einzige Unterschied besteht darin, dass statt `AppendText` zum Anhängen an die Datei mit `CreateText` eine neue Textdatei erzeugt wird:

```
Dim logs As StreamWriter = File.CreateText(log)
```

Zwei Ausnahmen werden außerdem verarbeitet, falls sie auftreten. Zuerst wird auf ein fehlendes Verzeichnis reagiert:

```
Catch e As DirectoryNotFoundException
```

Außerdem reichen die Zugriffsrechte möglicherweise nicht aus:

```
Catch e As UnauthorizedAccessException
```

Ausgabe einer Protokolldatei

```
127.0.0.1 16.7.2002 14:33:47  
127.0.0.1 16.7.2002 14:33:51  
192.168.100.22 16.7.2002 14:36:41  
127.0.0.1 16.7.2002 14:36:49  
192.168.100.22 16.7.2002 14:36:52  
127.0.0.1 16.7.2002 14:36:56
```

Abbildung 4.48: Darstellung der Protokolldatei nach ein paar Seitenabrufen

5 Grundlagen der Datenspeicherung

Keine Anwendung kann auf die Speicherung von Daten in einem Datenbankmanagementsystem (DBMS) verzichten. Vor allem bei großen Datenmengen und komplexen Zugriffen sind DBMS unverzichtbar. Wenn es jedoch nur um kleine Datenmengen geht, wird immer häufiger XML als Speicherformat eingesetzt. In beiden Fällen sind vergleichbare Zugriffstechniken verfügbar. ADO.NET ist ein Oberbegriff für die vielen Klassen in .NET, die der Datenspeicherung, dem Selektieren von Daten und dem Schreiben dienen. Dieses Kapitel führt in die Grundlagen der Datenspeicherung ein, behandelt die elementaren Funktionen der Abfragesprache SQL und seinem XML-Pendant XQuery.

5.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

5.1.1 Über dieses Kapitel

Dieses Kapitel behandelt die Grundlagen der Speicherung von Daten. Für die Programmierung von Webseiten kommen zwei Methoden in Betracht: Relationale Datenbankmanagementsysteme (RDBMS) und XML. Es ist trotz des Hypes um XML absehbar, dass beide Techniken auf lange Sicht parallel verwendet werden. In der Praxis stellt sich schnell heraus, dass es für die eine wie die

andere Methode sinnvolle Anwendungen gibt. Erst durch die Beherrschung von beiden Varianten können optimale Programme entwickelt werden.

Die Grundlage der Abfrage und Steuerung von Datenbanken bildet die Abfragesprache SQL. Sie wird in diesem Kapitel vorrangig behandelt. Ein umfangreicher Teil geht dann auf den SQL Server 2000 und den SQL-Dialekt T-SQL (Transact-SQL) ein. Solide Kenntnisse in T-SQL sind unentbehrlich, wenn große Projekte mit ASP.NET und ADO.NET entwickelt werden sollen.

Ein weiterer großer Teil dieses Kapitels befasst sich mit XML und der Transformationssprache XSLT. Damit wird eine profunde Basis für die in .NET weit verbreitete Nutzung von XML geliefert. Die XML-Fähigkeit des SQL Servers 2000 führt darüber hinaus zu einer wichtigen Schnittstelle, die die Beherrschung von SQL und XML gleichermaßen erfordert.

Dieses Kapitel beschäftigt sich vor allem mit den theoretischen Grundlagen der Datenverarbeitung. Die praktische Anwendung folgt mit der Einführung in ADO.NET im Kapitel 9, »Datenbanken und ADO.NET« ab Seite 823.

Empfehlenswert ist auch die Konsultation des Abschnitts 6.6, »Vorlagentgebundene Daten-Steuer-elemente« ab Seite 566, wo auf die Ausgabe der Daten aus Datenbanken mit Hilfe entsprechenden Steuer-elemente eingegangen wird.

5.2 Die Grundlagen

Vor den ersten Schritten mit einem Datenbankmanagementsystem steht die Auswahl. Dabei spielt neben dem verfügbaren Budget auch die spätere Applikation bei der Auswahl eine Rolle. Provider bieten meist MS Access-Unterstützung ohne Aufpreis an, während SQL Server 2000-fähiger Webspace relativ teuer ist.

5.2.1 Auswahl des Datenbankmanagementsystems

Auswahl Für dieses Buch wird weitgehend der SQL Server 2000 als Datenbankmanagementsystem eingesetzt. Im professionellen ASP.NET-Umfeld gibt es dazu – zumindest in der Microsoft-Welt – kaum eine Alternative. Die nachfolgenden Ausführungen zu SQL basie-

ren deshalb weitestgehend auf Transact-SQL. Sie sind jedoch, wenn dadurch keine Nachteile in der Darstellung entstehen, so weit vereinfacht, dass die Übertragung auf MS Access oder ein anderes Datenbankmanagementsystem leicht möglich ist. Als drittes System, das für den Entwickler unter Windows verfügbar ist, steht MSDE zur Verfügung. Dieses DBMS ist ein SQL Server-Derivat, das über keine Benutzeroberfläche verfügt, also auch keine Managementkonsole. Die Leistungsfähigkeit entspricht der Standardversion des SQL Servers.

Für den Einsatz mit ASP.NET kommt MSDE in der Praxis nicht in Betracht. Sie können jedoch Ihre lokalen Anwendungen damit testen und dann im Praxisbetrieb auf SQL Server laufen lassen. Die größte Einschränkung stellt dabei die Anzahl der Verbindungen dar, die geöffnet werden können. Diese sind bei MSDE auf zehn beschränkt.

5.2.2 Die Basistechnologien

Gerade am Anfang ist es wichtig, den Überblick zu behalten und die geeignete Methode zum Speichern und Abfragen von Daten auszuwählen. Ist die Wahl gefallen, kann eine systematische Einarbeitung in die speziellen Methoden der gewählten Technologie erfolgen. Relationale Datenbankmanagementsysteme (RDBMS) bestimmen seit langer Zeit den Stand der Datenspeichertechnik. In vielen Fällen lassen sich Daten damit effektiv ablegen und schnell wieder abfragen. Als Abfragesprache kommt meist SQLStructured Query Language – Structured Query Language – zum Einsatz. Diese Sprache liefert einige elementare Befehle, die sowohl das Speichern als auch das Auslesen von Daten in einfacher Form gestatten. Freilich liegt auch hier die Tücke im Detail. SQL kann, wenn die Problemstellung entsprechend komplex ist, sehr anspruchsvoll sein. Für die ersten Schritte im Zusammenhang mit ASP.NET gibt es dennoch einen einfachen Einstieg.

ADO.NET suggeriert, dass die Abfrage von Datenbanken durch geeignete Methoden und Eigenschaften erfolgen kann. Tatsächlich geht es aber mehr darum, die Art und Weise des Zugriffs so einfach wie möglich zu gestalten und gleichzeitig die technischen Ressourcen bestmöglich auszunutzen. Im Endeffekt basiert jedoch

SQL-Datenbanken

**SQL trotz
ADO.NET?**

auch mit ADO.NET jede Abfrage und jeder Einfügevorgang, der direkt in der Datenbank ausgeführt werden soll, auf SQL. SQL umfasst sehr viele Befehle. Unverzichtbar sind die folgenden vier:

► *INSERT*

Damit werden Daten eingefügt.

► *DELETE*

Dieser Befehl löscht Daten wieder.

► *UPDATE*

Vorhandene Daten werden damit aktualisiert.

► *SELECT*

Der sehr mächtige Befehl erlaubt die Abfrage von Daten.



Der folgende Abschnitt führt sehr kompakt in die Grundlagen von SQL ein, ohne die Sie ADO.NET nicht dauerhaft erfolgreich einsetzen können. Das Kapitel zu ADO.NET zeigt dann dennoch, wie weit Sie ohne SQL kommen können. In der Praxis wird oft mehr verlangt. Blättern Sie dann in dieses Kapitel hier und den Abschnitt über Transact-SQL zurück.

Was ist ADO.NET?

Mit der Datenbank auf der einen Seite und einer Abfragesprache gelangen die Daten noch nicht zum Benutzer. Zwischen ASP.NET und der Datenbankschnittstelle stellt das .NET-Framework eine Schicht zur Verfügung, die den Zugriff weitgehend abstrahiert. Diese Schicht wird ADO.NET genannt. Sie bildet sich aus einer ganzen Reihe von Klassen im Namensraum `System.Data`. Weitere Namensräume folgen darunter, was auf den Umfang der Möglichkeiten hindeutet, denn jeder enthält unzählige Klassen. Vom Prinzip her schafft ADO.NET eine lokale Kopie des aktuell gefragten Teils der Datenbank. Diese Kopie wird, wann immer es möglich ist, im Speicher gehalten. Entsprechend effizient können Abfragen erfolgen.

**Daten-Steuer-
elemente**

Nun steht dem programmatischen Zugriff von ASP.NET nichts mehr im Wege. Was noch fehlt, sind geeignete Web Server-Steuer-elemente, die die Darstellung übernehmen. Auch hier kann der Softwareentwickler aus dem Vollen schöpfen. Nahezu jedes Element einer ASP.NET-Seite kann eine Datenquelle an sich binden und so die Darstellung sehr einfach und direkt übernehmen. Die Datenquelle beschränkt sich dabei nicht auf Objekte aus

ADO.NET, sondern kann auch Typen wie `Array`, `Hashtable` oder `SortedList` umfassen. Letztere können ihren Inhalt natürlich aus Datenbankabfragen bezogen haben.

5.3 SQL mit MS Access lernen

Am besten lernen Sie SQL, wenn Sie mit Daten spielen. Die folgenden Beispiele zeigen, wie eine neue Datenbank mit MS Access erzeugt und mit Tabellen und Daten gefüllt wird. Unabhängig von den sonstigen Möglichkeiten werden jedoch für die Erfassung und den Abruf von Daten bewusst SQL-Kommandos verwendet. Die Abschnitte über ADO.NET nutzen sowohl diese Tabellenstrukturen als auch vergleichbare Kommandos.

Dieser Abschnitt bereitet die Nutzung der Datenbank in den nächsten Schritten vor. Ziel ist es, eine Datenbank zu entwickeln, die als Basis eines Shops dienen kann. In weiteren Beispielen wird die fertige Version dann eingesetzt, um die Bestellinformationen aufzunehmen.

**Praktischer
Nutzen inklusive**

5.3.1 Was ist SQL?

SQL ist eine Datenbankabfragesprache. Der große Vorteil liegt darin, dass Sie mit der Kenntnis dieser Sprache viele Datenbanken benutzen können, die mit SQL kompatibel sind. Dadurch werden Ihre Datenbankanwendungen unabhängig von einer bestimmten Datenbank eines einzelnen Herstellers.

Im deutschen Sprachraum hat es sich übrigens eingebürgert, die drei Buchstaben auch deutsch und einzeln auszusprechen, wenn von SQL die Rede ist.



Technisch sind SQL-Kommandos kleine Befehlszeilen oder Anweisungen, die an den Datenbankserver gesendet werden. Die Befehle sind in englischer Sprache gehalten und sollten den Sinn der Abfrage erkennen lassen. Sie können in SQL auch kleine Programme schreiben. Allerdings ist SQL ganz streng auf die Bedienung von Datenbanken ausgerichtet. SQL alleine bringt wenig.

Richtig leistungsfähig wird SQL in Verbindung mit einer Programmierungsumgebung – in diesem Buch sind dies die Sprache VB.NET und die Bibliotheken unter dem Namen ADO.NET.

Richtig leistungsfähig wird SQL in Verbindung mit einer Programmierungsumgebung – in diesem Buch sind dies die Sprache VB.NET und die Bibliotheken unter dem Namen ADO.NET.

Die Geschichte relationaler Datenbanken und der Abfragesprache SQL

Relationale Datenbanken können auf eine – für IT-Verhältnisse – beachtliche Geschichte zurückblicken. Das Konzept basiert auf mathematischen Modellen, die Anfang der siebziger Jahre an IBM-Forschungslaboren in Kalifornien entwickelt wurden. Das erste relationale Datenbanksystem, das System R, konnte erstmals mit Tabellen umgehen. Schon für dieses System wurde eine spezielle Abfragesprache entwickelt, SQL genannt. 1979 war die Entwicklung weitgehend abgeschlossen und das erste kommerziell verfügbare System wurde auf den Markt gebracht. 1981 war es dann soweit, SQL/DS erschien für das Betriebssystem DOS/VSE. 1983 folgte die erste Version des DBMS DB2 für IBMs Betriebssystem MVS. Parallel dazu bauten weitere Hersteller eigene Datenbankmanagementsysteme. Zu den ersten gehörte Oracle mit einer Datenbank für DEC (Digital Equipment Corporation) im Jahre 1979. Kurz danach trat schon Informix auf den Plan.

1982 gründete ANSI (American National Standards Institute) ein Komitee, das sich mit der Normierung von SQL auseinandersetzte. Dies führte erst 1986 zu ersten Ergebnissen, dem Standard ANSI X3.135. Für Europa übernahm daraufhin die ISO (International Standards Organization) diese Empfehlung als SQL-1. Wie üblich in der sich rasant entwickelnden Welt der EDV waren die Hersteller schneller mit Erweiterungen am Markt, als es den Normungsgremien gelang, die Funktionen festzuschreiben. So folgte der 1992 verabschiedete Standard SQL-2 oder SQL-92 zwar den damaligen Vorgaben des Marktes, konnte jedoch nur einen Teil der Erweiterungen umsetzen.

Heute gilt SQL-92 als kleinste gemeinsame Menge der SQL-Dialekte aller Datenbanken. Trotzdem sind teilweise drastische Unterschiede festzustellen, auch bei DBMS aus demselben Haus. So hat Microsoft mit Transact-SQL (T-SQL) einen

sehr umfassenden eigenen Dialekt entwickelt, der im SQL Server zum Einsatz kommt. Die Funktionen der Datenbank MS Access basieren dagegen auf Visual Basic, nicht auf T-SQL.

5.3.2 Die Bestandteile einer Datenbank

Wenn Sie noch keine klare Vorstellung davon haben, was eine Datenbank eigentlich ist, lesen Sie die folgende kurze Einführung. Ansonsten springen Sie zum nächsten Abschnitt 5.3.5, »Daten löschen mit DELETE« ab Seite 364.

Spalte und Datensatz

Datenbanken dienen der Speicherung von Daten. Das können Namen, Adressen, Zahlen, Zeichenketten usw. sein. Daten stehen auch untereinander in Beziehung. So bilden die Teile einer Adresse eine zusammengehörende Einheit. Eine solche Einheit nennt man Datensatz. Die Teile eines Datensatzes bilden die Felder. Alle Felder eines Typs werden als Spalte bezeichnet.

Jedes Feld hat zwei grundlegende Eigenschaften – einen Namen und einen Datentyp. Die Datentypen sind in SQL denen von VB.NET ähnlich und werden vergleichbar streng überprüft. Die Namen zur Angabe sind jedoch anders und die Einstellmöglichkeiten sind noch feiner. Größere Datenbankmanagementsysteme, wie der Microsoft SQL Server, kennen noch mehr Datentypen und prüfen noch strenger feinste Einstellungen, als dies mit dem .NET-Framework oder einer Programmiersprache möglich wäre. Wenn Sie Adressen speichern, könnten die Felder und Datentypen folgendermaßen aussehen:

SQL-Datentypen

```
FIRMA, Varchar(80)
STRASSE, Varchar(80)
ORT, Varchar(50)
PLZ, Integer(5)
```

Die Bedeutung dieser Angaben ist leicht zu verstehen. Das Feld *FIRMA* kann Zeichenketten mit bis zu 80 Zeichen aufnehmen, das Feld *PLZ* (Postleitzahl) kann ganzzahlige Werte mit 5 Stellen aufnehmen usw. Beachten Sie, dass SQL andere Namen für Datentypen als .NET verwendet. So werden Zeichenketten in den Typen

`Char` (feste Länge zwischen 1 und 255 Zeichen) oder `Varchar` (variable Länge bis 255 Zeichen) gespeichert, wobei in beiden Fällen die maximale Zeichenanzahl festgelegt ist. `Char` in SQL und `Char` VB.NET sind also sehr verschiedene Datentypen.

Tabelle und Datenbank

Datenbank Eine Datenbank ist eine Sammlung von Datenbankobjekten, die neben den eigentlichen Daten auch weitere Hilfsinformationen und Anweisungen enthalten kann. Wesentlicher Bestandteil einer Datenbank sind die Tabellen, in denen der Betreiber die Daten ablegt. In der Datenbank werden aber auch Prozeduren, Zugriffsrechte und Verknüpfungen hinterlegt. Man spricht deshalb allgemein von Datenbankobjekten. Diese sollten Sie nicht mit den in .NET verwendeten Objekten verwechseln, auch wenn einige sehr direkt durch entsprechende Klassen abgebildet werden.

Tabelle Tabellen sind Datenbankobjekte, in denen die Daten abgelegt sind. Tabellen werden durch eine Anordnung von Spalten und Reihen dargestellt. Wenn eine komplette Reihe aus der Tabelle extrahiert wird, um die darin enthaltenen Daten zu betrachten oder zu manipulieren, wird von einem Datensatz gesprochen. Wenn aus dem Datensatz eine bestimmte Spalte ausgewählt wird, nennt man dies ein Feld. Daraus resultieren oft Missverständnisse. Felder können, müssen aber nicht, gleich Spalten sein. Und Datensätze können, müssen aber nicht, gleich Reihen sein. Die Unterscheidung ist dann von Bedeutung, wenn davon gesprochen wird, ob Befehle »spaltenweise« oder »reihenweise« arbeiten. Ebenso können Datentypen nur für eine Spalte und nicht für ein Feld eingestellt werden. Die folgende Abbildung zeigt die Zusammenhänge auf einen Blick.

Typische Objekte einer Datenbank sind also in jedem Fall Tabellen, die die Daten enthalten. Definiert werden können auch Indizes, die dem schnelleren Auffinden und Sortieren der Daten dienen. Letztlich ist es noch wichtig zu wissen, dass das Datenbankmanagementsystem die Datenbank selbst als Datei (oder in mehreren Dateien) im Dateisystem ablegt. Bei Microsoft Access wird eine Datei mit der Dateierweiterung `.mdb` verwendet.



Bei den folgenden Seiten werden Sie sich vielleicht fragen, ob Sie die Datenbankgrundlagen wirklich brauchen – nur um ein kleines Projekt zu entwickeln. Die Antwort ist ein klares »Ja«: auch kleine Datenbanken

sind, wenn ein falsches Modell zugrunde liegt, nicht wirklich beherrschbar. Sie investieren ein Mehrfaches an Zeit, um einfachste Funktionen mit einer Programmiersprache und dem Framework umständlich nachzubilden oder können bestimmte Strukturen überhaupt nicht sinnvoll abbilden. Das Verständnis für diese kompakte Einführung ist unbedingt notwendig.

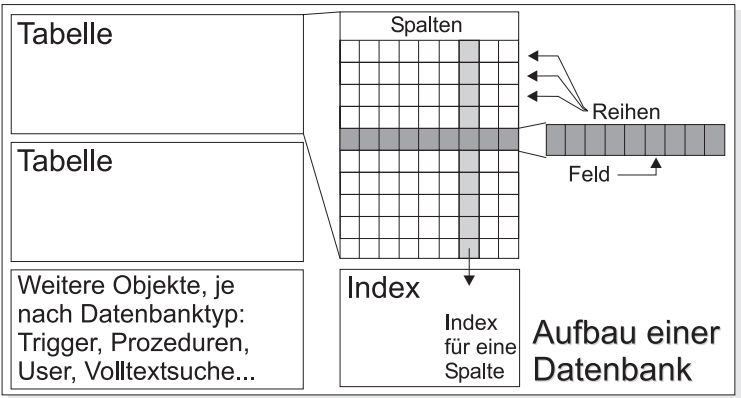


Abbildung 5.1: Prinzipieller Aufbau einer Datenbank

Das relationale Modell und die Datennormalisierung

Das relationale Datenbankmodell verlangt eine ganz bestimmte Form von internen Beziehungen (Relationen) zwischen den Daten. Bei der Wahl der Tabellenstruktur werden Sie damit konfrontiert, Ihre Daten in Tabellen zu speichern. Denken Sie dabei an den bereits mehrfach angesprochenen Shop – sicher eine sehr typische Anwendung. Die folgende Tabelle zeigt eine Reihe eindeutiger Bestellvorgänge, die einige Personen ausgelöst haben:

id	name	artikel	aid	preis
1	Clemens Krause	Windows XP Pro	1325	199,99
2	Clemens Krause	MSDN Abo Professional	6668	2850,00
3	Lukas Werlich	Windows XP Home	1326	139,99
4	Lukas Werlich	Big Color Keyboard	7738	99,00
5	Carolin Scholz	Windows XP Home	1326	139,99
6	Carolin Scholz	Big Color Keyboard	7738	99,00

Tabelle 5.1: Eine nicht normalisierte Tabelle mit Bestelldaten

Die Tabelle enthält alle Informationen, die für die Abwicklung benötigt werden, sowohl die Namen als auch die Daten der Artikel und die Bestellinformationen. Stellen Sie sich außerdem vor, dass weitere nicht gezeigte Spalten für die Adresse und die Artikelbeschreibung usw. in der Praxis hinzukommen. Diese Tabelle ist offensichtlich sehr unübersichtlich und – was noch schwerer wiegt, – sie enthält redundante Daten. So tauchen sowohl die Namen als auch Artikelnummern und Preise mehrfach auf. Die Pflege einer solchen Tabelle ist sehr aufwändig. Die Änderung an einem Datensatz – beispielsweise einem Benutzernamen – würde dazu führen, dass mit der unmittelbaren Operation nicht im Zusammenhang stehende Änderungen an anderen Datensätzen erforderlich sind.

Hintergründe zum relationalen Modell

Den ursprünglichen Anforderungen an die Datenspeicherung in Datenbanken kommt man mit dem relationalen Modell am nächsten. Diesem allgemeinen Modell liegt das *Entity-Relationship-Modell* zugrunde, das auf der Relationentheorie basiert. Aus dieser entwickelte E. F. Codd 1970 bei IBM das relationale Modell.

Das relationale Modell basiert auf der Mengenlehre und bezeichnet gleichartige Objekte der realen Welt als Entitätstypen. Als Entität wird allgemein ein Ding oder Wesen bezeichnet, das als Einheit abgebildet wird. Der Begriff fußt auf dem englischen Wort »entity«, wurde jedoch relativ konsequent eingedeutscht. Entitäten sind real, bilden in einer Datenbank so etwas wie Personen, Produkte oder Kostenstellen ab; Dinge also, die existieren, Eigenschaften haben und erzeugt und vernichtet werden können. Ausprägungen der Entitätstypen sind konkrete Objekte. Der Entitätstyp Person wird in der Praxis mit wirklichen Personen hinterlegt, bzw. deren Daten in einer Datenbank.

Entitätstypen haben bestimmte Merkmale, durch die sie näher beschrieben werden können. Bei einer Person kann dies der Name, eine Altersangabe, das Geschlecht usw. sein. Diese Merkmale werden allgemein als Attribut bezeichnet, wobei es sich bei Entitätstypen korrekterweise um Attributtypen handelt. Der

Attributtyp umfasst Merkmale des Attributes, beispielsweise einen Datentyp, Wertebereich oder andere Beschränkungen des Abbildungsraumes der Werte im Attribut. So könnte man eine Altersangabe dahingehend einschränken, dass sie nicht negativ werden kann, weil dies per Definition sinnlos ist.

Die mit Entitätstypen definierten Entitäten stehen nicht isoliert im Raum. Sie bilden tatsächlich Beziehungen untereinander aus. So kann eine Entität »Telefon« einer Entität »Person« zugeordnet werden, weil dies die Beziehung »Person hat ein Telefon« aus der realen Welt abbildet. Solche Beziehungen werden im Englischen als »relationship« bezeichnet. Daraus wurde der allgemeine und im Zusammenhang mit Datenbank korrektere Begriff »Relation« entwickelt. Datenbanken, die Entitäten und deren Relationen abbilden können, heißen folgerichtig »relationale Datenbanken«.

Das relationale Modell basiert auf so genannten Schlüsseln. Dabei wird einer Entität ein Schlüssel zugeordnet, der mit einem anderen Schlüssel einer anderen Entität die Relation abbildet. Der eigene Schlüssel wird als Primärschlüssel bezeichnet, der Schlüssel in der verknüpften Entität als Fremdschlüssel. Abgebildet werden die Schlüssel als Attributtypen. Bei Beziehungen zwischen Personen und ihnen zugeordneten Dingen handelt es sich oft um Kundennummern. Der Schlüssel muss letztlich nur eindeutig sein.

Um die Tabelle in eine für relationale Datenbanken passende Form zu bringen, werden die Daten »normalisiert«. Dazu werden folgende vier Regeln angewandt:

Normalisierungsregeln

► Eliminierung sich wiederholender Gruppen

Für jede Gruppe von Daten, die sich wiederholen, legen Sie eine eigene Tabelle an. Im Beispiel haben Sie drei Gruppen: Adressen, Bestellungen und Artikel. Also werden drei Tabellen angelegt. Jede Tabelle bekommt einen eindeutigen Schlüssel, das heißt eine Spalte, anhand der die Datensätze eindeutig unterschieden werden können.

► Eliminierung redundanter Daten

Wenn eine Eigenschaft mehrere Werte annehmen kann, bringen Sie diese in eine eigene Tabelle. Wenn Sie zum Beispiel

feststellen, dass jeder Artikel mehrere Preise haben kann, legen Sie zusätzlich eine Tabelle für die Preise an.

- **Eliminieren von Spalten, die von keinem Schlüssel abhängen**
Wenn Eigenschaften keinen Zusammenhang mit dem Schlüsselfeld haben, also nicht ebenso eindeutig zugeordnet werden können, übertragen Sie diese Eigenschaften in eine eigene Tabelle. Wenn Sie im Beispiel zwei Adressen pro Kunden haben (Lieferanschrift und Rechnungsanschrift), können Sie die Adresse nicht mehr eindeutig einem Schlüsselfeld Kunden-ID zuordnen. Trennen Sie die Anschriften des Kunden in weitere Tabellen ab.
- **Isolierung unabhängiger Beziehungen**
Keine Tabelle darf zwei oder mehr Beziehungen haben, die nicht direkt abhängig sind.

Die oben gezeigte Tabelle 5.1 sollten Sie also in mindestens drei Tabellen splitten. Die erste enthält alle Adressinformationen. Die gezeigte Auswahl ist nur ein Teil dieser Tabelle, damit hier die Übersicht nicht verloren geht:

id	name	strasse	plz	ort
1	Clemens Krause	Dornenweg 33	12345	Berlin
2	Lukas Werlich	Spielallee 11	89487	München
3	Carolin Scholz	Eichenwall 4	40748	Düsseldorf

Tabelle 5.2: Eine Tabelle, die ausschließlich Adressdaten speichert (Adressen)

Die zweite Tabelle enthält alle Artikelinformationen. Auch hier können Sie sich beliebig viele weitere eindeutige Informationen in weiteren Spalten vorstellen. Redundante Informationen – beispielsweise Eigenschaftsklassen – könnten zu weiteren Tabellen führen. Hier ein Ausschnitt:

id	name	preis	nummer
1	Windows XP Pro	199,99	1325
2	MSDN Abo Professional	2850,00	6668
3	Windows XP Home	139,99	1326
4	Big Color Keyboard	99,00	7738

Tabelle 5.3: Die Artikeltablelle (Artikel)

Die eigentlichen Bestellungen enthalten weit weniger Stammdaten und dafür mehr Verweise auf andere Tabellen.

id	k-id	a-id	menge	summe
1	1	1	1	199,99
2	1	2	1	2850,00
3	2	3	1	139,99
4	2	4	1	99,00
5	3	3	1	139,99
6	3	4	1	99,00

Tabelle 5.4: Die Bestelltabelle – ohne redundante Informationen (Bestellungen)

Der Gesamtpreis wurde mit aufgenommen, weil der Preis der Artikeltabelle sich ändern kann, während der Preis einmal verkaufter Artikel sich nicht mehr ändern darf. Beim Normalisieren muss also immer beachtet werden, wo sich Daten ändern können und welche Auswirkungen die Verknüpfungen haben und ob dies auch beabsichtigt ist. Man muss deshalb sorgfältig zwischen »Stammdaten« und »Betriebsdaten« unterscheiden. Im Beispiel ist der Preis in der Tabelle *Produkte* vom Typ Stammdaten. Alle Operationen, die Werte dieser Art übernehmen, entnehmen sie aus einer Stammdatentabelle. Die Bestelltabelle zeichnet dagegen einen Momentanzustand auf. Dieser muss erhalten bleiben, auch wenn sich Stammdaten ändern. Es ist Aufgabe der Programmlogik, zwischen beiden Zuständen eine Beziehung herzustellen, wenn dies erforderlich sein sollte.

Stamm- und Betriebsdaten

Unabhängig vom Typ der Daten werden Beziehungen hergestellt. Die folgende Abbildung zeigt die möglichen Verbindungen. Genutzt werden hier so genannte 1:n-Beziehungen, das heißt auf der einen Seite sind die Schlüssel eindeutig, auf der anderen können dagegen ein oder auch mehrere Datensätze zugeordnet werden. Es gibt auch 1:1- und n:n-Beziehungen.

Beziehungen = Relationen

Man kann derartige Beziehungen recht gut grafisch darstellen. Die folgende Abbildung zeigt, wie die Informationen in der Tabelle *Bestellungen* mit denen in *Adressen* und *Artikel* verknüpft sind.

Lassen Sie sich an dieser Stelle nicht davon abbringen, sich intensiv mit Datenbanken auseinander zu setzen, auch wenn dies kompliziert erscheint. Es gibt einfache Anwendungsfälle, die keine Normalisierung benötigen und die nur aus einer einzigen Tabelle bestehen. Es ist aber notwendig zu wissen, was auf Sie zukommt, wenn komplexere Zusammenhänge datentechnisch abgebildet werden müssen. Es ist der Anspruch dieses Buches, dieses Wissen zu vermitteln.



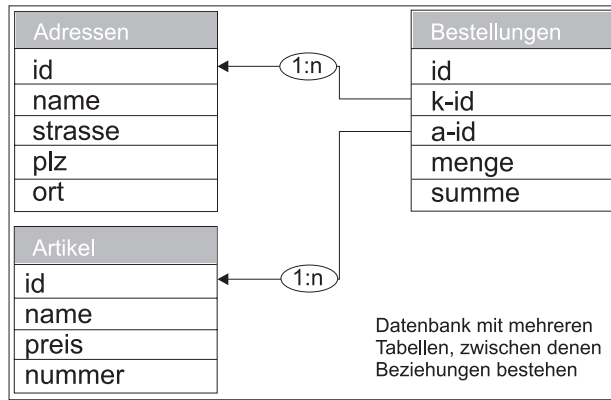


Abbildung 5.2: Beziehungen zwischen den Tabellen

5.3.3 Praktische Arbeit mit Datenbanken

Nach all der Theorie sollten Sie jetzt mit der praktischen Arbeit beginnen. Für die ersten Schritte werden drei Tabellen erzeugt und mit ein paar Musterdaten gefüllt. Für das Erzeugen wird Access und der Assistent für neue Tabellen verwendet.

Freilich nutzen Sie danach auf dem Weg zum Profi nicht die Assistenten von Access dafür, sondern SQL, um Daten einzufügen, zu löschen oder zu ändern. Denn exakt dieses Wissen wird danach in der ASP.NET/ADO.NET-Programmierung eingesetzt, um die Datenbank über ein Programm zu bedienen.

Der Datenbankzugriff mit Access

Neue Datenbank anlegen

Starten Sie Access und legen Sie mit DATEI | NEUE DATENBANK ANLEGEN... eine neue Datenbank an. Sie können einen Assistenten oder eine der mitgelieferten Vorlagen benutzen. Für die meisten Webprojekte dürfte das nicht sehr sinnvoll sein. Legen Sie für die Beispiele eine leere Datenbank mit dem Namen *Shop* an. Sie wird in allen restlichen Abschnitten dieses Kapitels verwendet. Speichern Sie diese im Verzeichnis */shop* unterhalb des Übungsverzeichnisses, beispielsweise unter *c:/inetpub/wwwroot/dotnet*.



Für Eilige ist die Tabelle auf der CD zum Buch zu finden. Sie sollten aber die Gelegenheit nutzen, die Schritte selbst anhand der folgenden Anleitung nachzuvollziehen.

Mit dem Assistenten ERSTELLT EINE TABELLE IN DER ENTWURFS-ANSICHT... können Sie nun eine neue Tabelle anlegen. Geben Sie die Informationen wie folgt ein:

1. Feldname *id*, Felddatentyp AUTOWERT

Klicken Sie außerdem mit der rechten Maustaste auf die äußerst linke Spalte und wählen Sie dann aus dem Kontextmenü PRIMÄRSCHLÜSSEL.

Aufbau der Tabelle Adressen

2. Feldname *name*, Felddatentyp TEXT

Im unteren Teil des Assistenten wählen Sie als Feldgröße 80.

3. Feldname *strasse*, Felddatentyp TEXT

Im unteren Teil des Assistenten wählen Sie als Feldgröße 80.

4. Feldname *plz*, Felddatentyp TEXT

Im unteren Teil des Assistenten wählen Sie als Feldgröße 5. Text wird hier verwendet, weil die Postleitzahl auch mit 0 beginnen kann. Diese führende Null würde bei numerischen Feldern verloren gehen.

5. Feldname *ort*, Felddatentyp TEXT

Im unteren Teil des Assistenten wählen Sie als Feldgröße 50.

Klicken Sie dann auf SPEICHERN (oder Menü DATEI | SPEICHERN).

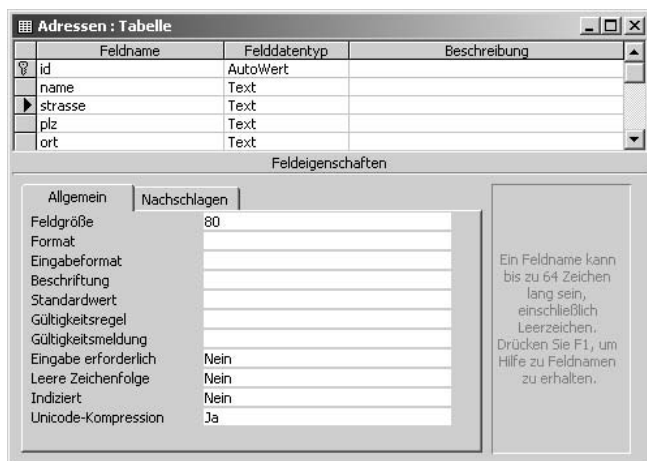


Abbildung 5.3: Die Tabelle Adressen in Access

Schließen Sie nun den Assistenten. Starten Sie ihn erneut, um die Tabelle *Artikel* zu erzeugen. Auch diese Tabelle wird für einige der folgenden Übungen benötigt.

Aufbau der Tabelle Artikel

- Feldname *id*, Felddatentyp AUTOWERT
Klicken Sie außerdem mit der rechten Maustaste auf die äußerst linke Spalte und wählen Sie dann aus dem Kontextmenü PRIMÄRSCHLÜSSEL.
- Feldname *name*, Felddatentyp TEXT
Im unteren Teil des Assistenten wählen Sie als Feldgröße 50.
- Feldname *preis*, Felddatentyp ZAHL
Im unteren Teil des Assistenten wählen Sie als Feldgröße DEZIMAL. In der Zeile DEZIMALSTELLEN wählen Sie den Wert 2.
- Feldname *nummer*, Felddatentyp ZAHL
Im unteren Teil des Assistenten wählen Sie als Feldgröße LONG INTEGER.

Klicken Sie dann auf SPEICHERN (oder Menü DATEI | SPEICHERN) und vergeben Sie den Namen *Artikel*.

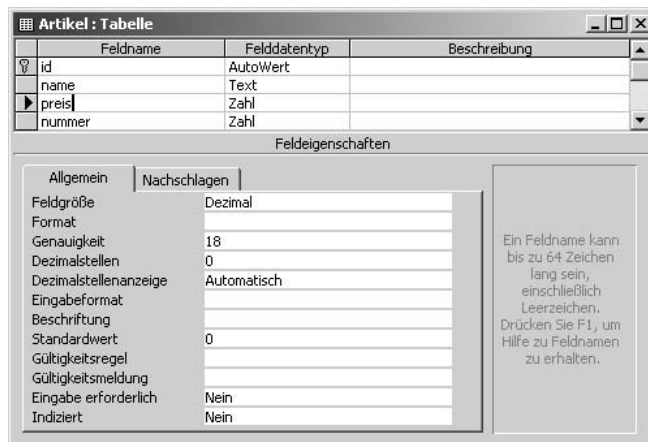


Abbildung 5.4: Die Tabelle Artikel in Access

In einer weiteren Tabelle sollen die Bestellungen des Shops gespeichert werden. Wie in der Übersicht bereits angedeutet, handelt es sich teilweise nur um Spalten zur Speicherung der Relationen:

Aufbau der Tabelle Bestellungen

1. Feldname *id*, Felddatentyp AUTOWERT
Klicken Sie außerdem mit der rechten Maustaste auf die äußerst linke Spalte und wählen Sie dann aus dem Kontextmenü PRIMÄRSCHLÜSSEL.

2. Feldname *k-id*, Felddatentyp ZAHL

Dieses Feld soll die Verknüpfung zu *id* der Tabelle *Adressen* herstellen.

3. Feldname *a-id*, Felddatentyp ZAHL

Dieses Feld soll die Verknüpfung zu *id* der Tabelle *Artikel* herstellen.

4. Feldname *menge*, Felddatentyp ZAHL, Feldgröße SINGLE.

5. Feldname *summe*, Felddatentyp ZAHL

Im unteren Teil des Assistenten wählen Sie als Feldgröße DEZIMAL. In der Zeile DEZIMALSTELLEN wählen Sie den Wert 2.

Klicken Sie dann auf SPEICHERN (oder Menü DATEI | SPEICHERN) und vergeben Sie den Namen *Bestellungen*.

Feldname	Felddatentyp	Beschreibung
id	AutoWert	ID
k-id	Zahl	
a-id	Zahl	
menge	Zahl	
preis	Zahl	

Feldeigenschaften

Allgemein | Nachschlagen

Feldgröße: Dezimal
 Format: 18
 Genauigkeit: 2
 Dezimalstellen: Automatisch
 Eingabeformat: 0
 Beschriftung: Standardwert
 Gültigkeitsregel: Nein
 Gültigkeitsmeldung: Nein
 Eingabe erforderlich: Nein
 Indiziert: Nein

Die Feldbeschreibung ist optional. Sie hilft, den Feldinhalt zu erklären und wird auch in der Statusleiste angezeigt, wenn Sie dieses Feld in einem Formular markieren.

Abbildung 5.5: Die Tabelle *Bestellungen* in Access

Wenn Sie in SQL Spaltennamen verwenden, die Leerzeichen oder Minuszeichen enthalten, müssen Sie bei der späteren Verwendung in Ausdrücken diese Namen in eckige Klammern setzen: [*a-id*]. In den entsprechenden Beispielen wird diese Syntax verwendet, wenn es notwendig ist.



Wo Sie SQL-Kommandos eingeben

Mancher eingefleischte Access-Nutzer wird großartige Projekte erstellt haben und dabei nie mit SQL in Berührung gekommen sein. Es gibt tatsächlich eine Vielfalt von Assistenten und Hilfen,

die auch ohne SQL-Kenntnisse ansprechende Ergebnisse hervorbringen. Diese nützen Ihnen nur von ADO.NET aus recht wenig.

SQL-Eingabefenster

Um zum SQL-Eingabefenster zu gelangen, erstellen Sie eine neue Abfrage. Klicken Sie in der Datenbank unter OBJEKTE auf ABFRAGEN. Starten Sie den Assistenten ERSTELLT EINE NEUE ABFRAGE IN DER ENTWURFSANSICHT.

SQL-Ansicht

Danach erscheint ein Fenster mit der Auswahl der Tabellen – anschließend wählen Sie die eben erzeugten an und klicken Sie dann auf HINZUFÜGEN und dann auf SCHLIESSEN. Klicken Sie nun mit der rechten Maustaste in einen freien Bereich und wählen Sie aus dem Kontextmenü den Eintrag SQL-ANSICHT.



Abbildung 5.6: Umschalten in die SQL-Ansicht in Access



Im danach erscheinenden Editor können Sie SQL-Kommandos eingeben und mit Hilfe des entsprechenden Icons ausführen (siehe Randspalte). Entfernen Sie den vom Assistenten vorgegebenen Text. Alle folgenden Beispiele, die sich um SQL-Kommandos ranken, werden in dieses Fenster eingegeben.

5.3.4 Daten speichern mit INSERT

INSERT INTO

Die SQL-Befehle zum Füllen mit Werten sind ein recht umständlicher Weg, Daten einzugeben. Auch hier gilt jedoch, dass Sie im Umgang mit diesen Befehlen sicher sein müssen, um für den Einsatz in ASP.NET vorbereitet zu sein. Was Sie hier mühevoll eintippen, kann dort freilich automatisiert werden.

Mit INSERT INTO fügen Sie Daten hinzu. Der Befehl wird folgendermaßen eingesetzt:

```
INSERT INTO adressen (name, strasse, plz, ort) ➡  
VALUES ('Clemens Krause', 'Dornenweg 33', ➡  
      '12345', 'Berlin')
```

Der mehrzeilige Aufbau verbessert nur die Lesbarkeit. SQL erwartet Kommandos in einer Zeile, wird jedoch durch Zeilenumbrüche nicht gestört. Ebenso spielt bei den Befehlen Groß- und Kleinschreibung keine Rolle.



Die Spaltennamen müssen nur angegeben werden, wenn nicht alle Spalten angesprochen werden. Die Reihenfolge der Namen spielt keine Rolle, muss also nicht der Definition entsprechen. Die Werte im `VALUE`-Teil müssen dagegen exakt der Reihenfolge der Spaltenliste folgen. Zeichenketten müssen in Anführungszeichen stehen. Sie können hier – im Gegensatz zu VB.NET – einfache Anführungszeichen verwenden. Dies hat den Vorteil, dass sie nicht mit den doppelten kollidieren, wenn Sie SQL und VB.NET später kombinieren.

Die Syntax des Befehls `INSERT` ist einfach. Zu beachten ist, dass die Anzahl der Spalten mit den Werten übereinstimmen muss. Ebenso ist die Reihenfolge entscheidend. Zeichenketten müssen in Anführungszeichen stehen. Es gibt auch eine verkürzte Form, bei der hinter `VALUES` exakt die Anzahl der definierten Spalten bedient werden muss:

```
INSERT INTO tabelle VALUES (wert, wert,...)
```

Die Spalte `id` wurde im Beispiel weggelassen, weil sie durch das Feldattribut `AutoWert` automatisch mit fortlaufenden Nummern gefüllt wird.



Geben Sie nun noch einige weitere Datensätze ein, um ein wenig mit den Daten spielen zu können. Es erscheint übrigens vor jeder Einfügeoperation ein Warnhinweis in Access, bei dem Sie die Eingabe bestätigen müssen. Beim Absenden von `INSERT` über ein Programm wird diese Barriere nicht mehr stören.

Mit einem Doppelklick auf den Namen der Tabelle können Sie sehen, ob der Vorgang erfolgreich war und alle Daten erfasst wurden. Wiederholen Sie die letzte Eingabe noch mit einem fehlerhaften Wert – in der folgenden Abbildung steckt der Fehler in Zeile 2. Die fehlerhafte Zeile wird im folgenden Abschnitt mit einem neuen Kommando korrigiert.

	id	name	strasse	plz	ort
▶	1	Clemens Kraus	Dornenweg 33	12345	Berlin
	2	Lukass Werlich	Spielallee 11	89487	München
	3	Lukas Werlich	Spielallee 11	89487	München
	4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf
*	(AutoWert)				

Datensatz: 1 von 4

Abbildung 5.7: Tabelle Adressen mit einigen Musterwerten

Für die nächsten Aufgaben werden alle drei Tabellen benötigt. Füllen Sie auch die Tabellen *Artikel* und *Bestellung* mit ein paar Artikelinformationen nach folgendem Muster:

```
INSERT INTO Artikel (name, nummer, preis)
VALUES ('Windows XP Home', 1326, 139.99)
```

Beachten Sie, dass Zahlen nicht in Anführungszeichen gesetzt werden und für Dezimalzahlen mit Kommastellen als Trennzeichen der Punkt geschrieben wird, auch wenn Access in der Anzeige die deutsche Darstellung wählt.

	id	name	preis	nummer
▶	1	Windows XP Pro	199,99	1325
	2	MSDN Abo Professional	2850	6668
	3	Windows XP Home	139,99	1326
	4	Big Color Keyboard	99	7738
*	(AutoWert)		0	0

Datensatz: 1 von 4

Abbildung 5.8: Tabelle Artikel mit Musterdaten

Ebenso sollten Sie ein paar Bestellungen in die Bestelltabelle eingeben, damit Musterdaten für die Beispiele vorhanden sind:

```
INSERT INTO Bestellungen ( [k-id], [a-id], menge, summe )
VALUES (3, 4, 1, 99);
```

Mit diesen Daten wird der kleine SQL-Kurs nun fortgesetzt.

5.3.5 Daten löschen mit DELETE

DELETE Neben dem Einfügen von Datensätzen ist auch das gezielte Löschen möglich. Dazu wird das Kommando `DELETE` eingesetzt. Die

Selektion der zu löschenden Datensätze erfolgt mit einer `WHERE`-Bedingung, die ebenso wie bei `SELECT` beschrieben, eingesetzt wird:

```
DELETE FROM Adressen WHERE id = 2
```

Lassen Sie die `WHERE`-Bedingung weg, werden alle Datensätze gelöscht. Die Tabelle selbst bleibt aber erhalten. Mehr zu `WHERE` finden Sie im Abschnitt 5.3.7, »Daten abfragen mit `SELECT`« ab Seite 365. Die hier verwendete Bedingung löscht den Datensatz, in dem das Feld `id` den Wert 2 hat (`id = 2`). SQL kennt ähnliche Operatoren wie VB.NET; der Vergleich wird mit einem statt mit zwei Gleichheitszeichen gebildet.

Eine andere Möglichkeit, den Fehler aus dem letzten Abschnitt zu beseitigen, wäre die Veränderung der Daten.

5.3.6 Daten ändern mit UPDATE

Neben dem Einfügen und Löschen können Sie Datensätze auch ändern. Dazu wird `UPDATE` eingesetzt. Sie müssen die zu ändernden Spalten und den neuen Wert angeben. Der Wert kann auch durch eine Formel definiert werden. Welche Datensätze geändert werden, entscheidet wieder die `WHERE`-Bedingung:

UPDATE

```
UPDATE Adressen SET plz = '10999' WHERE id = 1
```

Auch hier erscheint in Access wie schon bei `INSERT` und `DELETE` der Warnhinweis. Ohne `WHERE` werden alle Datensätze geändert – was meist fatale Folgen hat. Mehrere Spalten ändern Sie gleichzeitig, in dem die entsprechenden Anweisungen der Form »Spalte = Neuer Wert« als Liste, durch Kommata getrennt, aufgeführt werden:

```
UPDATE Adressen SET name = 'Jörg Krause', plz = '12683' WHERE id = 1
```

5.3.7 Daten abfragen mit SELECT

Häufiger werden Daten abfragt, wozu `SELECT` verwendet wird. Es ist empfehlenswert, sich intensiv mit `SELECT` auseinander zu setzen, denn die sorgfältige Auswahl der benötigten Daten durch das Datenbankmanagementsystem ist meist schneller, als wenn Sie alle Daten lesen und dies anschließend programmtechnisch mit VB.NET erledigen.

SELECT

Die einfachste Abfrage ist die Ausgabe aller Datensätze einer Tabelle; mit dem folgenden Befehl werden alle Name aus der Tabelle *Adressen* angezeigt:

```
SELECT * FROM Adressen
```

Es erscheint nun die Ansicht dieser Abfrage:

id	name	strasse	plz	ort
1	Jörg Krause	Dornenweg 33	12683	Berlin
3	Lukas Werlich	Spielallee 11	89487	München
4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf
*	(AutoWert)			

Abbildung 5.9: Ergebnis der Abfrage der Tabelle mit *SELECT*

Nun besteht die Kunst im Umgang mit SQL nicht darin, ganze Tabellen abzurufen. Oft wird ein ganz bestimmter Datensatz benötigt oder es ist sogar ein Zusammenhang zwischen mehreren Tabellen herzustellen. Letzterer Fall resultiert vor allem aus der Normalisierung der Daten, die zur »Auftrennung« redundanter Informationen führt.

SELECT verstehen

Das »*« in der letzten Abfrage ist eine Kurzform für »alle Felder«. Der Aufbau des Kommandos orientiert sich ansonsten an folgendem Muster:

```
SELECT spalte, spalte, ...
FROM tabelle, tabelle, ...
WHERE bedingung
```

Dabei ist mindestens eine Spalte und eine Tabelle anzugeben, weitere Angaben sind optional. Der Befehl besteht aus zwei Schlüsselworten. Das Schlüsselwort *SELECT* (dt. Auswahl) leitet den Befehl ein, *FROM* (dt. aus) wählt die Tabelle und *WHERE* (dt. wo, wobei) ist die Bedingung.



SQL kennt keine Zeilennummern, nach denen Daten ausgewählt werden könnten. Sie müssen als Alleinstellungsmerkmal selbst eine Spalte dafür einrichten. In den gezeigten Tabellen übernimmt diese Aufgabe jeweils die Spalte id. Dass die Zahlen darin am Anfang fortlaufend mit 1 beginnend zählen, ist gewollt, aber kein typisches Merkmal dieser Spalten.

Anstatt eines Spaltennamens kann auch ein Ausdruck genutzt werden, beispielsweise `spalte * 1.16`. SQL verfügt hier je nach Datenbankmanagementsystem über ein reiches Spektrum an Funktionen und Operatoren. Das Schlüsselwort `WHERE` und die nachfolgenden Bedingungen sind optional. Mit ihnen kann die Abfrage eingeschränkt werden.

Datenbankabfragen oder, allgemein, Abfragen (engl. query), kennen Sie sicher schon. Jede Anfrage an die Suchmaschinen wie AltaVista oder Yahoo löst eine Datenbankabfrage aus. Viele dieser Abfragefelder kennen Boolesche (logische) Operatoren, UND (engl. and), ODER (engl. or), NICHT (engl. not) usw. So können Sie in AltaVista nach »VB.NET' AND 'SQL'« suchen. Nur jene Datensätze werden ausgegeben, die »VB.NET« UND »SQL« im Suchtext hatten (beachten Sie die einfachen Anführungszeichen zum Kombinieren der Wörter zu einer Phrase).

Nach einem `SELECT`-Kommando erscheint das Ergebnis der Abfrage im Fenster und die SQL-Ansicht ist verschwunden. Klicken Sie mit der rechten Maustaste auf die Kopfzeile des Fensters und wählen Sie im Kontextmenü erneut die Option SQL-ANSICHT.



In SQL schreiben Sie die Abfrage, bezogen auf die Mustertabelle *Adressen*, folgendermaßen:

```
SELECT name, ort FROM Adressen WHERE name = 'Jörg Krause'
```

Abfragen gestalten

Sehen Sie sich den folgenden Zugriff auf die Tabelle *Artikel* an:

```
SELECT name, preis FROM Artikel
```

Es entsteht folgendes Ergebnis:

name	preis
Windows XP Pro	199,99
MSDN Abo Professional	2850
Windows XP Home	139,99
Big Color Keyboard	99

Abbildung 5.10: Ergebnis der Abfrage

Sie können sich nun alle Preise mit und ohne Mehrwertsteuer ansehen:

```
SELECT name, preis AS netto, preis * 1.16 AS brutto FROM Artikel
```

- AS** Mit Hilfe des Operators **AS** werden neue Spaltennamen für die Ausgabe erzeugt. **AS** steht für Alias. Außerdem werden Berechnungen ausgeführt, ebenso wie Sie dies aus VB.NET bereits kennen. Hier können Sie alle Operationen ausführen, die zu kompletten Ausdrücken führen. Als Werte können Spaltennamen oder Konstanten eingesetzt werden.

name	netto	brutto
Windows XP Pro	199,99	231,9884
MSDN Abo Professional	2850	3306
Windows XP Home	139,99	162,3884
Big Color Keyboard	99	114,84

Abbildung 5.11: Ergebnis der Abfrage

- ORDER BY ... DESC|ASC** Für eine bestmögliche Übersicht können Sie die Ausgabe sortieren lassen:

```
SELECT name, preis AS netto, preis * 1.16 AS brutto
FROM Artikel
ORDER BY preis DESC
```

- Sortieren von Daten** Mit **DESC** (vom engl. descend) wird absteigend sortiert und mit **ASC** (vom engl. ascend) können Sie auch aufsteigend sortieren. Sortiert wird nach dem ersten oder dem angegebenen Feld, hier also nach *preis*. Zwei Dinge sollten Sie beachten: Das Sortieren großer Datenbanken beansprucht den Server stark. Sie können außerdem nur nach den Spalten sortieren, die auch ausgegeben werden. Normalerweise können Sie aber davon ausgehen, dass die Sortierung in der Datenbank schneller ist, als mit VB.NET und dem Framework. Voraussetzung ist jedoch eine entsprechende Indizierung der Datenbank.

Was ist ein Index?

Ein Index ist eine zusätzliche Option zur Verwaltung von Tabellen in Datenbanken. Der Index bildet eine Zugriffstabelle auf Basis einer Spalte einer normalen Tabelle ab. Das Durch-

suchen oder Sortieren funktioniert erheblich schneller, als wenn die gesamte Tabelle behandelt werden müsste. Spalten, die häufig als Kriterium für solche Vorgänge herangezogen werden, sollten indiziert werden. In Microsoft Access wählen Sie bei der Tabellendefinition (Entwurfsansicht) die entsprechende Spalte und dann in Eigenschaftsliste, Zeile INDIZIERT, die Option JA.

Die Verwendung des Index passiert intern, sodass Sie sich bei der Gestaltung der `SELECT`-Kommandos darauf nicht mehr konzentrieren müssen. Indizes wirken übrigens auch positiv auf die Geschwindigkeit beim Einfügen mit `INSERT`, wenn bereits sehr viele Daten vorhanden sind.

Der bereits angesprochene Primärschlüssel führt übrigens automatisch zu einem Index nach der Spalte *id*. Das ist kein Merkmal von Access, sondern gilt für alle Datenbankmanagementsysteme. Dies impliziert auch, dass jede Tabelle mehrere Indizes besitzen kann.

Manchmal sind Datensätze oder Teile davon doppelt vorhanden. So gibt es in der Tabelle *Bestellungen* viele Artikel mehrfach, wenn entsprechende Bestellungen erfolgten. Wenn Sie einfach nur wissen möchten, welche Artikel bereits bestellt wurden, deren Anzahl aber nicht benötigen, hilft `DISTINCT` weiter:

```
SELECT DISTINCT [a-id] FROM Bestellungen
```

Dieser Befehl braucht mehr Leistung von der Datenbankmaschine. Setzen Sie ihn nicht in häufig generierten Abfragen ein.

DISTINCT



In den meisten Fällen werden Zeichenketten verarbeitet. SQL unterstützt die Arbeit mit Zeichenketten deshalb sehr komfortabel. So ist es leicht möglich, eine Teilzeichenkette zur Selektion anzugeben, die in allen Feldern an allen Positionen getestet wird. Das Schlüsselwort dafür ist `LIKE`. Der folgende Befehl sucht alle Personen, in deren Namen das Wort »Krause« steckt:

```
SELECT * FROM Adressen WHERE name LIKE '*Krause*'
```

LIKE

MS Access verwendet hier als Platzhalter das bekannte Sternchen bzw. für ein einzelnes Zeichen das Fragezeichen. Wenn Sie mit SQL Server oder einer anderen Datenbank arbeiten, werden üblicherweise die in SQL



definierten Platzhalterzeichen »%« und »_« verwendet. Im Abschnitt zum SQL Server 2000 wird darauf noch gesondert eingegangen.

Bedingungen mit WHERE definieren

Sie können mit WHERE logische Bedingungen angeben. Wollen Sie alle Artikel in einem bestimmten Preisfenster wissen, können Sie Folgendes definieren:

```
SELECT * FROM Artikel WHERE preis > 100 AND preis < 200
```

id	name	preis	nummer
3	Windows XP Home	139,99	1326
1	Windows XP Pro	199,99	1325

Abbildung 5.12: Resultat der Abfrage

BETWEEN

Solche Kombinationen aus zwei komplementären Bedingungen lassen sich auch – und oft besser lesbar – mit BETWEEN ausdrücken:

```
SELECT * FROM Artikel WHERE preis BETWEEN 100 AND 200
```

Die Schreibweise mit BETWEEN (dt. zwischen) ist zur vorher genutzten Form äquivalent; das Ergebnis ist identisch, wenn Sie beachten, dass BETWEEN auf Gleichheit der Grenzwerte prüft. Dies entspräche <= bzw. >= im vorhergehenden Beispiel.

SQL-Funktionen

Als Nächstes sollen Artikel anhand des Namens ausgewählt werden. Die einfache Abfrage lautet:

```
SELECT nummer, name
FROM Artikel
WHERE LCASE (name) LIKE '*pro*'
```

Hier wird zusätzlich eine Funktion eingesetzt: LCASE. Sie überführt alle Zeichen in Kleinbuchstaben. Dadurch erfüllt die Suche nach »pro« auch den Namen, der »Professional« enthält. Es gibt in SQL sehr viele solcher Funktionen; welche Sie einsetzen können, ist in der entsprechenden Programm-Hilfe zu finden. Leider ist hier SQL nicht gleich SQL, jedes Datenbankmanagementsystem verwendet teilweise eine unterschiedliche Syntax für diese Funktionen.



MS Access verwendet keine SQL-spezifischen Funktionen, sondern das VBA-Modul (VBA = Visual Basic for Applications). Damit stehen über 100 Funktionen bereit. Im SQL Server 2000 stehen die Funktionen aus dem Transact-SQL-Dialekt zur Verfügung.

Alle Reihen sollen ermittelt werden, die in der Spalte *plz* die Zeichenkette »12683« oder »89487« haben. Dies erledigt zuverlässig der Operator **IN**:

```
SELECT * FROM Adressen WHERE plz IN ( '12683' , '89487' )
```

Um die Auswahl zu negieren, ist das Schlüsselwort **NOT** erlaubt. Bei logischen Bedingungen mit **AND** und **OR** kennen Sie das schon vom VB.NET-Befehl *If...Then*. Auch die Schlüsselworte **BETWEEN** und **IN** lassen sich mit **NOT** kombinieren.

Abfragen mehrerer Tabellen

Wenn Daten aus mehreren Tabellen benötigt werden, werden die Abfragen schon etwas komplizierter. Andererseits kann hier SQL seine Vorteile voll ausspielen – es gibt fast nichts, was man nicht auf direktem Weg abfragen könnte.

Am Anfang wurde angenommen, dass neben Adressen auch Bestelldaten und Artikel gespeichert werden. Nun sind die Daten aber – wegen der angesprochenen Normalisierung – auf drei Tabellen verteilt. Der Abruf einer bestimmte Bestellung oder auch aller Daten erfordert, dass alle drei Tabellen gleichzeitig angesprochen werden. Probieren Sie zuerst Folgendes aus:

```
SELECT Adressen.name, Artikel.name, Bestellungen.id
FROM Adressen, Artikel, Bestellungen
```

Die Abbildung in der Randspalte zeigt, dass das Kommando zwar ausgeführt wurde, allerdings nicht unbedingt sinnvolle Daten lieferte. SQL erstellt, wenn keine Verknüpfung angegeben wurde, eine so genannten Kreuztabelle aus den Daten – kombiniert also jede Spalte der linken Tabelle mit jeder der rechten, was einer Multiplikation der Anzahl der Zeilen entspricht.

Das ist sicher nicht das erwartete Ergebnis, denn Access hat hier einfach alle Tabellen miteinander vermischt. Die Beziehungen, die anfangs ja nur gedanklich existierten, sind nicht angegeben worden. Sie können nicht davon ausgehen, dass nur die Benennung mit »id« zu brauchbaren Ergebnissen führt. Letztlich steht Ihnen die Namenswahl sowieso frei. Also müssen die Beziehungen angegeben werden. Dazu dient wiederum die **WHERE**-Bedingung:

```
SELECT a.name, p.name, b.id AS BestellID
FROM Adressen a, Artikel p, Bestellungen b
```

IN

NOT

Adressen.name	Artikel.name	id
Jörg Krause	Windows XP Pro	1
Lukas Werlich	Windows XP Pro	1
Carsten Scholz	Windows XP Pro	1
Jörg Krause	MSDN Abo Profess	1
Lukas Werlich	MSDN Abo Profess	1
Carsten Scholz	MSDN Abo Profess	1
Jörg Krause	Windows XP Home	1
Lukas Werlich	Windows XP Home	1
Carsten Scholz	Windows XP Home	1
Jörg Krause	Big Color Keyboard	1
Lukas Werlich	Big Color Keyboard	1
Carsten Scholz	Big Color Keyboard	1
Jörg Krause	Windows XP Pro	2
Lukas Werlich	Windows XP Pro	2
Carsten Scholz	Windows XP Pro	2
Jörg Krause	MSDN Abo Profess	2
Lukas Werlich	MSDN Abo Profess	2
Carsten Scholz	MSDN Abo Profess	2
Jörg Krause	Windows XP Home	2
Lukas Werlich	Windows XP Home	2
Carsten Scholz	Windows XP Home	2
Jörg Krause	Big Color Keyboard	2
Lukas Werlich	Big Color Keyboard	2
Carsten Scholz	Big Color Keyboard	2
Jörg Krause	Windows XP Pro	3
Lukas Werlich	Windows XP Pro	3
Carsten Scholz	Windows XP Pro	3
Jörg Krause	MSDN Abo Profess	3
Lukas Werlich	MSDN Abo Profess	3
Carsten Scholz	MSDN Abo Profess	3
Jörg Krause	Windows XP Home	3
Lukas Werlich	Windows XP Home	3
Carsten Scholz	Windows XP Home	3
Jörg Krause	Big Color Keyboard	3
Lukas Werlich	Big Color Keyboard	3
Carsten Scholz	Big Color Keyboard	3
Jörg Krause	Windows XP Pro	4
Lukas Werlich	Windows XP Pro	4
Carsten Scholz	Windows XP Pro	4
Jörg Krause	MSDN Abo Profess	4
Lukas Werlich	MSDN Abo Profess	4
Carsten Scholz	MSDN Abo Profess	4
Jörg Krause	Windows XP Home	4
Lukas Werlich	Windows XP Home	4
Carsten Scholz	Windows XP Home	4
Jörg Krause	Big Color Keyboard	4
Lukas Werlich	Big Color Keyboard	4
Carsten Scholz	Big Color Keyboard	4
Jörg Krause	Windows XP Pro	5
Lukas Werlich	Windows XP Pro	5
Carsten Scholz	Windows XP Pro	5
Jörg Krause	MSDN Abo Profess	5
Lukas Werlich	MSDN Abo Profess	5
Carsten Scholz	MSDN Abo Profess	5
Jörg Krause	Windows XP Home	5
Lukas Werlich	Windows XP Home	5
Carsten Scholz	Windows XP Home	5
Jörg Krause	Big Color Keyboard	5
Lukas Werlich	Big Color Keyboard	5
Carsten Scholz	Big Color Keyboard	5
Jörg Krause	Windows XP Pro	6
Lukas Werlich	Windows XP Pro	6
Carsten Scholz	Windows XP Pro	6
Jörg Krause	MSDN Abo Profess	6
Lukas Werlich	MSDN Abo Profess	6
Carsten Scholz	MSDN Abo Profess	6
Jörg Krause	Windows XP Home	6
Lukas Werlich	Windows XP Home	6
Carsten Scholz	Windows XP Home	6
Jörg Krause	Big Color Keyboard	6
Lukas Werlich	Big Color Keyboard	6
Carsten Scholz	Big Color Keyboard	6

```
WHERE a.id = b.[k-id]  ←
AND p.id = b.[a-id]
```

Aliase Neben der Angabe hinter `WHERE` wurden hier auch noch so genannte Alias-Namen verwendet (`a` für Adressen, `p` für Artikel und `b` für Bestellungen). Diese dienen nur der Verkürzung der Schreibweise. Schauen Sie sich zum Vergleich mit dem letzten Ergebnis die verwendete Tabelle *Bestellungen* an. Das Ergebnis der Abfrage spiegelt diese Tabelle wieder, allerdings sind neben den Bestell-IDs auch die Namen der Käufer und Artikel zu finden:

a.name	p.name	BestellID
Jörg Krause	Windows XP Pro	1
Jörg Krause	MSDN Abo Professional	2
Lukas Werlich	Windows XP Home	3
Lukas Werlich	Big Color Keyboard	4
Carolin Scholz	Windows XP Home	5
Carolin Scholz	Big Color Keyboard	6

Abbildung 5.13: Korrekte Abfrage der Bestellungen mit Zuordnungen aus anderen Tabellen

INNER JOIN Letztlich basiert das ganze Verfahren nur auf der Verbindung der ID-Spalten: `a.id = b.[k-id]` usw. Die Verknüpfung von zwei Tabellen kann in SQL auch mit dem Operator `INNER JOIN` ausgedrückt werden. Das Ergebnis erscheint immer dann, wenn zwei Felder der beiden Tabellen übereinstimmen:

```
SELECT a.name, b.id AS BestellID  ←
FROM Bestellungen b  ←
INNER JOIN Adressen a  ←
ON a.id = b.[k-id]
```

Dieses Kommando zeigt alle Namen aus der *Adressen*-Tabelle an, zu denen Bestellungen vorliegen.

Zusammenfassung

Gezeigt wurde ein Überblick über den grundsätzlichen Aufbau einer Datenbank, die Arbeitsweise eines Datenbankmanagementsystems und Sie haben die elementarsten SQL-Befehle kennen gelernt. Damit lassen sich bereits kleinere datenbankgestützte Webprojekte umsetzen. Für den Zugriff auf die Datenbank aus

einem ASP.NET-Programm heraus benötigen Sie nun noch ADO.NET – eine Sammlung von entsprechenden Klassen – und einige auf die Ausgabe von Daten spezialisierte Steuerelemente.

Dieser Abschnitt konnte lediglich eine kompakte Einführung in SQL bieten, gerade genug, um ADO.NET nun erfolgreich einsetzen zu können. Wenn Sie weiterhin mit MS Access arbeiten möchten, fahren im Abschnitt 9.2, »Grundlagen zu ADO.NET« ab Seite 825 fort. Für kleinere Projekte ist dies völlig ausreichend und zudem in fast allen Webpace-Angeboten bei Providern ohne zusätzliche Kosten enthalten.

**Weiter zu
ADO.NET**

Wenn Sie den Umstieg auf den SQL Server 2000 planen, sollten Sie sich zuvor mit Transact-SQL beschäftigen. Dies wird im folgenden Abschnitt 5.5, »Transact-SQL – SQL mit dem SQL Server 2000 lernen« ab Seite 376 behandelt. Mit diesen Grundlagen können dann auch größere Datenbankprojekte umgesetzt werden. Viele Provider stellen auf Anfrage oder gegen Aufpreis auch den Zugriff auf SQL Server 2000 zur Verfügung. Sie haben damit die optimale Plattform für Projekte, die unter hoher Last laufen.

5.4 Der Datenbankzugriff in Visual Studio .NET

5.4.1 Datenzugriff zur Entwurfszeit

In Visual Studio .NET (VS.NET) kann bereits während der Entwicklungszeit Verbindung mit der Datenbank aufgenommen werden. Sie können so leicht Daten manipulieren und Designänderungen ausführen. Der folgende Abschnitt geht davon aus, dass Sie einen SQL Server 2000 im lokalen Netzwerk installiert haben und dieser von Ihrem Entwicklungssystem aus erreichbar ist.

**VS.NET
verwenden**

Existiert die Datenbank noch nicht, legen Sie diese nun auf dem SQL Server Enterprise Manager an. Wenn Sie die Schritte im vorhergehenden Abschnitt mit MS Access ausgeführt haben, können Sie die Tabellen mit den DTS-Assistenten importieren.

5.4.2 Herstellen einer Datenbankverbindung

Datenbank- verbindung

Zum Herstellen einer Datenbankverbindung in Visual Studio .NET gehen Sie folgendermaßen vor:

1. Wählen Sie im Menü EXTRAS die Option MIT DATENBANK VERBINDEN.
2. Prüfen Sie auf der Registerkarte PROVIDER, ob als Treiber OLE DB PROVIDER FOR SQL SERVER ausgewählt ist.
3. Auf der Registerkarte VERBINDUNG geben Sie den Namen des SQL Servers ein.
4. Aktivieren Sie das Optionsfeld INTEGRIERTE SICHERHEIT VON WINDOWS NT VERWENDEN, wenn der SQL Server entsprechend eingerichtet wurde.
5. Klicken Sie auf VERBINDUNG TESTEN.

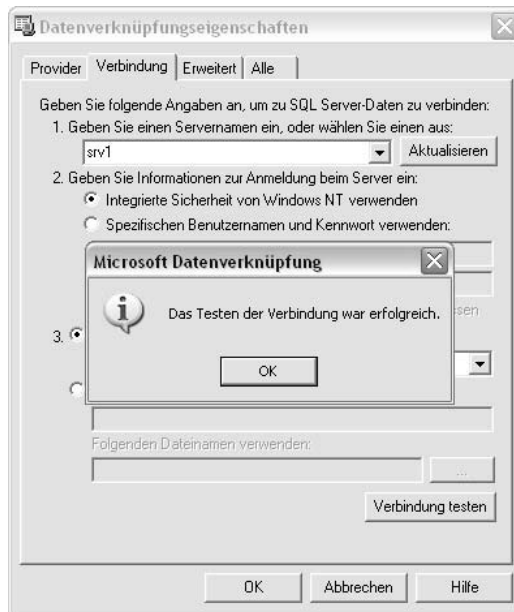


Abbildung 5.14: Datenbankserver verknüpfen, Datenbank erzeugen und Verbindung testen

5.4.3 Zugriff auf die Daten in Visual Studio .NET

Sie können auf bereits existierende Tabellen in VS.NET zugreifen und Daten direkt manipulieren, beispielsweise zu Testzwecken.

Der Zugriff auf den SQL Server 2000 zum Anlegen von neuen Datenbanken ist nur mit der Enterprise Version von Visual Studio .NET möglich. Mit der Professional Version können Sie lediglich Datenbanken in der SQL Server Desktop Edition (MSDE) erstellen, die für Webanwendungen kaum zum Einsatz kommen dürfte. Die Manipulation von Daten in vorhandenen Strukturen ist dagegen auch mit VS.NET Professional möglich, was in den meisten Fällen ausreichend ist.



Wenn Sie den Datenbankserver im lokalen Netzwerk oder auf Ihrem Entwicklungssystem betreiben, ist der direkte Zugriff unkritisch. Steht der Server dagegen beim Provider, wird er mit Sicherheit hinter einer Firewall platziert sein. Dann können Sie mit VS.NET ohnehin nicht direkt zugreifen, sondern nur über eine Web-Schnittstelle. Informieren Sie sich bei Ihrem Provider, wie er diese Schnittstelle realisiert hat.



Öffnen Sie dazu den Server-Explorer mit `Strg-Alt-S` oder über das Menü ANSICHT, Option SERVER-EXPLORER.

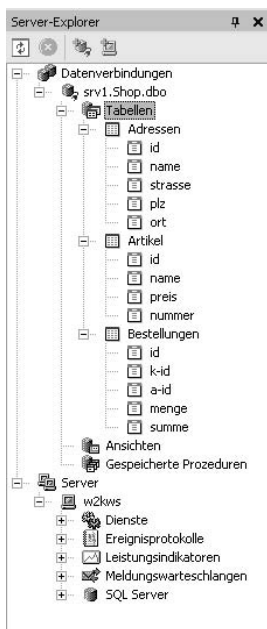


Abbildung 5.15: Ansicht des Server-Explorers mit Datenbankverbindung

Zur Vorbereitung der Übungen sollten Sie sich mit der Struktur der kleinen Spiel-Datenbank *Shop* vertraut machen. Der Diagramm-Designer des SQL Servers zeigt diese folgendermaßen:

Shop-Datenbank (vereinfacht)

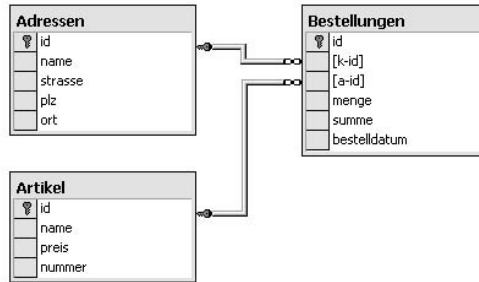


Abbildung 5.16: Struktur der Datenbank im Diagramm-Designer

5.5 Transact-SQL – SQL mit dem SQL Server 2000 lernen

Dieser Abschnitt führt in die wichtigsten Funktionen des SQL Servers auf Basis der Abfragesprache Transact-SQL (T-SQL) ein. Die mit SQL-92 kompatiblen Befehle, die im vorigen Abschnitt zu Access bereits erklärt wurden, werden hier nicht noch einmal ausführlich gezeigt.

5.5.1 Datentypen in T-SQL

Datenbankmanagementsysteme sind extrem typstrenge. Dies geht auch über das hinaus, was Sie vom .NET-Framework kennen. Neben einer Reihe elementarer Datentypen können sehr feine Einschränkungen der Wertebereiche definiert werden, so genannte »Constraints«. Dadurch ist es beispielsweise möglich, festzulegen, dass einer Spalte nur Werte zugewiesen werden können, die Ganzzahlen zwischen 1 und 5 sind. Die Integrität der Tabelle wird durch die Datenbank allein überwacht, was einiges an Programmierung in ASP.NET sparen kann.

Die Datentypen für Zeichenketten

Zeichenketten

Häufig werden die Zeichenkettentypen benötigt. Es gibt vier Datentypen, die zur Verwaltung von Zeichenketten verwendet werden können:

- ▶ `VARCHAR(länge)`
Zeichenkette mit minimal 0 und maximal *länge* Zeichen. Die Länge kann höchstens 8.000 Zeichen betragen.
- ▶ `CHAR(länge)`
Zeichenkette mit genau *länge* Zeichen. Maximal 8.000 Zeichen sind möglich.
- ▶ `NVCHAR(länge)`
Unicode-Zeichenkettenfeld mit variabler Länge. Das Feld kann maximal 4.000 Unicode-Zeichen in 8.000 Bytes speichern.
- ▶ `NCHAR(länge)`
Unicode-Zeichenkette mit fester Länge. Das Feld kann maximal 4.000 Unicode-Zeichen in 8.000 Bytes speichern.

Binärdaten speichern

Binäre Felder können bis zu 8.000 Byte aufnehmen. Wenn immer es der Platzverbrauch erlaubt, sollten Sie `BINARY` verwenden. Diese Form ist schneller, da der SQL-Server intern mit Speicherseiten fester Größe arbeitet und deshalb mit Feldern konstanter Größe besser umgehen kann. Sie werden wie folgt definiert:

Binäre Felder

- ▶ `BINARY(byte)`
Binärfeld fester Länge, beginnend mit 1. Der tatsächliche Speicherplatz beträgt $n+4$ Byte. Der physische Platzverbrauch ist unabhängig vom Inhalt und abhängig von der Definition.
- ▶ `VARBINARY(byte)`
Dieses Feld hat keine feste Länge, sondern passt sich dem Inhalt an und verbraucht nur so viel Platz, wie der Inhalt in Anspruch nimmt.

Große Text- oder Binärobjekte

8.000 Zeichen sind für Textfelder und Bilder nicht ausreichend. Folgende Typen verkraften größere Datenmengen:

Große Datenfelder

- ▶ `TEXT`
Dieser Datentyp dient der Speicherung von Textfeldern bis zu einer Größe von 2 GByte.

► NTEXT

Dieser Datentyp speichert 1.073.741.823 Unicode-Zeichen. Da Unicode-Zeichen 16 Bit breit sind, entspricht dies 1 Milliarde Zeichen oder 2 GByte.

► IMAGE

Dieser Datentyp speichert 2 GByte (2.147.483.647 Byte) binärer Daten. Wenn die Daten mit `INSERT` direkt eingefügt werden (siehe auch `WRITETXT`), muss vor der Zeichenkette das Symbol »0x« stehen und der Rest eine hexadezimale Zeichenfolge der Binärdaten ergeben.

Numerische Datentypen

Ganzzahlen Umfangreicher sind die numerischen Datentypen. Zuerst die Ganzzahltypen:

► BIGINT

Der Wertebereich dieses Datentyps reicht von -2^{63} (-922.337.203.685.477,5808) bis $+2^{63}$ (+922.337.203.685.477,5807, ca. 922 Billionen).

► INT

Wertebereich -2^{31} (-2.147.483.646) bis $+2^{31}$ (+2.147.483.647), nur ganzzahlige Werte (32 Bit Integer).

► SMALLINT

Wertebereich -32.767 bis +32.768, nur ganzzahlige Werte (16 Bit Integer).

► TINYINT

Wertebereich von 0 bis 255 (1 Byte). Negative Werte werden nicht unterstützt.

Gleitkommazahlen Verschiedene Gleitkommatypen erlauben präzise Berechnungen. Beachten Sie, dass bei der Abfrage mit `WHERE`-Bedingungen Gleitkommawerte nur selten exakt übereinstimmen. Folgende Typen können verwendet werden:

► `NUMERIC(int, frac)`

Gleitkommazahlen von $-10^{38}+1$ bis $10^{38}-1$. Der Parameter *int* gibt die Anzahl der Stellen vor dem Komma an, *frac* die Anzahl der Dezimalstellen.

► DECIMAL

Dieser Typ entspricht exakt `NUMERIC` und existiert aus Gründen der Kompatibilität mit dem ANSI SQL-92-Standard.

► `FLOAT(n)`

Gleitkommazahlen von $2,23 \times 10^{-308}$ bis $1,79 \times 10^{308}$. Der Parameter *n* gibt die Genauigkeit an.

► DOUBLE PRECISION

Dieser Typ entspricht exakt `FLOAT` und existiert aus Gründen der Kompatibilität mit dem ANSI SQL-92-Standard.

Bei der Arbeit mit Geldwerten können die Währungstypen interessant sein. Diese Datentypen verwenden immer vier Kommastellen, werden bei der Ausgabe auf 1 Hundertstel (Pfennig, Cent) gerundet und rechnen intern mit einem Zehntausendstel des Basiswertes. Es sind aber reine numerische Werte, die keine Währungsformatierungen mitführen (wie beispielsweise € oder \$). Hier die beiden möglichen Typen:

Monetäre Werte

► MONEY

Planen Sie große Geschäfte, speichern Sie damit Ihre Kontostände. Der Wertebereich dieses Datentyps reicht von -922.337.203.685.477,5808 bis +922.337.203.685.477,5807 (922 Billionen).

► SMALLMONEY

Knapp für den Heimgebrauch reicht der Wertebereich dieses Datentyps von -214.748,3648 bis +214.748,3648.

Sie können die Währungstypen auch für technische Daten verwenden, denn sie werden im Gegensatz zu den Gleitkommatypen exakt abgespeichert und intern nicht gerundet. Mit diesen Typen funktionieren auch WHERE-Abfragen auf Gleichheit korrekt.



Binärwerte

Auch logische oder besser Boolesche Werte lassen sich speichern. Der SQL Server 2000 legt einfach ein Bitfeld an, das 1 Bit enthält. 0 ist `FALSE` (falsch), 1 ist gleich `TRUE` (wahr). Das Schlüsselwort dafür ist `BIT`. Intern werden Bit-Felder in Bytes konvertiert, 1 Bit nimmt mindestens 1 Byte in Anspruch, mehrere Bitfelder werden aber in einem Byte gesammelt.

Bitfelder

Datums- und Zeitwerte

Datum und Zeit Zwei weitere Datentypen dienen der Speicherung von Datum und Zeit. Bei der Eingabe wird ein alphanumerisches Format verwendet und in Form einer Zeichenkette übergeben, also in einfachen Anführungszeichen.

► **DATETIME**

Daten ab dem 1.1.1753 bis zum 31.12.9999 und darin Zeiten in Schritten zu einer Millisekunde lassen sich speichern. Das klingt besser als es tatsächlich ist. Falls Sie die Geschichte des Christentums in einer SQL-Datenbank erfassen wollen, werden Sie schnell bemerken, dass Computer nicht bibelfest sind. Immerhin ist der Datentyp Jahr-2000-sicher. Vor allem aber sind die in .NET verfügbaren Kalender für größere Datumsbereiche ausgelegt (meist ab dem Jahr 1). Überlegen Sie deshalb sorgfältig, ob **DATETIME** für Ihre Zwecke ausreichend ist.

► **SMALLDATETIME**

Dieser Typ speichert vom 1.1.1900 bis 6.6.2079.

Spezielle interne Datentypen

TIMESTAMP Ein Feld mit dem Datentyp **TIMESTAMP** enthält einen sich selbst bei jedem Zugriff mit **INSERT** oder **UPDATE** erhöhenden Zähler. Der Typ hat nichts mit den Zeit- und Datentypen zu tun. Er dient nur der Registrierung von Änderungen. Parameter sind nicht notwendig. Jede Tabelle kann nur eine Spalte mit dem Datentyp **TIMESTAMP** haben. Der Datentyp benötigt intern 8 Byte.

UNIQUEIDENTIFIER Wenn Sie Ihre Datensätze eindeutig identifizieren müssen und keine andere Spalte dafür in Frage kommt, eignet sich dieser Typ evtl. Intern wird mit 16 Byte gearbeitet. Der Datentyp **UNIQUEIDENTIFIER** benötigt keine Parameter.

CURSOR Der SQL Server kann Zugriffe auf mehrere Datensätze über Zeiger steuern. Diese Technik werden Sie möglicherweise verwenden müssen, weil die mit ADO.NET eingeführte zeigerlose Methode nicht immer optimal ist, auch wenn dieser Mangel durch eine Vielzahl sehr fortschrittlicher Funktionen weitgehend ausgeglichen wird. Wollen Sie Zeiger speichern, verwenden Sie den Datentyp **CURSOR**.

Wenn Sie einer Spalte einen bestimmten Datentyp nicht zuweisen können, verwenden Sie `SQL_VARIANT`. Dies sollte die ganz große Ausnahme sein. Es ist wichtig, Wertebereiche immer so eng wie möglich zu deklarieren. `SQL_VARIANT` findet gelegentlich in temporären Tabellen Verwendung.

SQL_VARIANT

Der Datentyp `TABLE` speichert Ergebnisse von Abfragen, die für sich genommen wieder Tabellen darstellen. Er spart möglicherweise das Erzeugen einer temporären Tabelle.

TABLE

Benutzerdefinierte Datentypen

Wenn Sie bestimmte parameterisierte Datentypen häufig benötigen, ist es sinnvoll, diese als benutzerdefinierte zu speichern. Die Erstellung erfolgt mit einer gespeicherten Systemprozedur, `sp_addtype`. Drei Angaben sind notwendig:

Systemprozeduren zum Anlegen benutzerdefinierter Datentypen

1. Der künftige Name des benutzerdefinierten Datentyps
2. Der Systemdatentyp, der zugrunde liegt, sowie dessen Parameter
3. Die Zulässigkeit des Wertes `NULL` für den neuen Typ ('`NULL`' oder '`NOT NULL`')

In T-SQL erstellen Sie solche Datentypen durch Aufruf der Prozedur `sp_addtype`:

```
EXECUTE sp_addtype postleitzahl, 'CHAR(5)', 'NOT NULL'
```

Löschen können Sie solche Typen mit der Prozedur `sp_droptype`:

```
EXECUTE sp_droptype 'postleitzahl'
```

5.5.2 Tabellen in T-SQL erzeugen, ändern und löschen

Die Möglichkeit, Tabellen komfortabel mit dem Enterprise Manager anzulegen, ist nicht immer gegeben. Zum einen kann der SQL Server bei einem Webspace-Provider laufen, wo der direkte Zugriff nicht möglich ist. Zum anderen kann die Kompatibilität einer Anwendung zu MSDE notwendig sein. MSDE hat keine Benutzeroberfläche. Es ist aber auch dann notwendig, Tabellenstrukturen programmtechnisch zu erzeugen, wenn Sie eine Applikation weitergeben möchten. Auch hier kann man schlecht vom Benutzer verlangen, die Tabellenstrukturen selbst anzulegen.

T-SQL ist mit den üblichen DDL-(Data Definition Language)-Befehlen ausgestattet, um Datenbanken, Tabellen und andere funktionale Bestandteile der Datenbank anzulegen.

Eine Datenbank anlegen und Datenbank löschen

Der folgende Befehl legt eine neue Datenbank an:

```
CREATE DATABASE Datenbankname
```

Dies ist die einfachste Form. Eine Reihe zusätzlicher, optionaler Parameter kann angegeben werden, um die Datenbank an persönliche Bedürfnisse anzupassen. Die Syntax kann deutlich ausgebaut werden:

```
CREATE DATABASE Shop ON PRIMARY
    FILENAME='Shop_Data.mdf',
    SIZE=1,
    MAXSIZE=1000,
    FILEGROWN=10
    LOG ON FILENAME='Shop_Log.ldf',
    SIZE=2,
    MAXSIZE=300,
    FILEGROWN=1
```

Dateistruktur und -größe

Jede einzelnen Angabe ist dabei optional. Die Zahlenangaben entsprechen Megabyte. Sie können mehrere Dateien angeben. Die Datenbank wird dann auf mehrere physische Dateien verteilt. Es ist nicht unbedingt notwendig, die Dateigröße anzugeben. Der SQL Server 2000 erhöht die Größe der Datei nach Bedarf. Dabei wird die Größe immer bei Erreichen der Grenzwerte um 10 % erhöht. Wenn Sie eine andere Strategie der Vergrößerung wünschen, setzen Sie den Parameter `FILEGROWN` ein.



Wenn Sie mehrere Dateien angeben und diese auf Festplatten ablegen, die durch getrennte Adapter bedient werden, erhöhen Sie deutlich die Geschwindigkeit der Datenverteilung.

Systemdatenbank model

Die neue Datenbank ist nicht leer. Sie enthält alle Objekte, die in der Systemdatenbank *model* untergebracht sind. Wenn Sie in jeder neuen Datenbank immer wieder bestimmte Objekte benötigen, beispielsweise nutzerdefinierte Datentypen, dann definieren Sie diese in der Datenbank *model*.

Wenn Sie eine Datenbank nicht mehr benötigen, können Sie diese folgendermaßen entfernen:

```
DROP DATABASE name [ , name]
```

Es ist erlaubt, mehrere Datenbanken mit einem Befehl zu löschen. Wenn sich Benutzer gerade mit der Datenbank verbunden haben und Lese- oder Schreibanforderungen ausgeführt werden, können Sie die Datenbank nicht löschen. Wenn Sie Datenbanken (und dies gilt auch für Tabellen und andere Objekte) mit dem Befehl `DROP` löschen, wird dies ohne Rückmeldung sofort ausgeführt. Der SQL Server fragt nicht zurück, ob der Befehl wirklich ausgeführt werden soll.

Datenbankdefinition ändern

Sie können die Eigenschaften bestehender Datenbanken leicht ändern. Dazu wird der Befehl `ALTER DATABASE` verwendet. Die Parameter entsprechen dem bereits besprochenen Befehl `CREATE DATABASE`. Neben der Änderung der Dateigröße können Sie vor allem weitere Dateien hinzufügen oder löschen. Die letzte verbleibende Datei muss groß genug sein, um alle Objekte der Datenbank aufnehmen zu können. Das folgende Beispiel entfernt eine Datei aus der Datenbank:

ALTER DATABASE

```
ALTER DATABASE Shop ➡  
    REMOVE FILE Shop_Temp.mdf
```

Das nächste Beispiel zeigt, wie die Größe auf »unbegrenzt« gesetzt werden kann:

```
ALTER DATABASE Shop ➡  
    MODIFY FILE FILENAME='Shop_Data.mdf' MAXSIZE=UNLIMITED
```

Eine andere Datenbank verwenden

Wenn Sie in gespeicherten Prozeduren oder in ASP.NET-Programmen arbeiten, beziehen sich alle eingegebenen Befehle auf die aktuelle Datenbank. Diese kann bei der Festlegung der Verbindungszeichenfolgen angegeben werden. Dies vereinfacht erheblich die Notation, denn Sie müssten sonst vor jeder Tabelle die Datenbank benennen. Manchmal kann es aber notwendig sein, innerhalb eines laufenden Programmes auf eine andere Datenbank zu wechseln. In diesem Fall würde das explizite Ab- und

Anmelden erhebliche Zeit und Ressourcen in Anspruch nehmen. Einfacher geht es mit dem Befehl `USE`:

```
USE database
```

Danach ist die Datenbank *database* die aktuelle Datenbank.

Tabellen anlegen und verwalten

Eigenschaften von Tabellen

Am Anfang des Kapitels wurde bereits einiges zu der prinzipiellen Organisation von SQL-Datenbanken gesagt. Tabellen sind darin das wichtigste Objekt; in ihnen werden die eigentlichen Daten abgelegt. Tabellen müssen bestimmte Eigenschaften haben, die hier noch einmal zusammengefasst sind:

- ▶ Der Name muss in der Datenbank eindeutig sein.
- ▶ Tabellen sind in Spalten (Felder) und Reihen (Datensätze) organisiert.
- ▶ Jede Reihe einer Tabelle muss eindeutig sein. Um das sicherzustellen, wird eine Spalte zum Primärschlüssel erklärt. Jede Reihe enthält mindestens in dieser Spalte einen eindeutigen (einmaligen) Wert.
- ▶ Jede Spalte hat für alle Datensätze der Spalte eindeutig gültige und einheitliche Eigenschaften. Spaltennamen müssen innerhalb einer Tabelle eindeutig sein.
- ▶ Die Reihenfolge (Anordnung) der Spalten und Reihen spielt keine Rolle. Die Art und Weise der physischen Speicherung hat keinen Einfluss auf die Daten.

Insgesamt wird die Verwaltung von Tabellen mit drei Befehlen ausgeführt: `CREATE TABLE`, `DROP TABLE` und `ALTER TABLE`.



Die folgenden Tabellen werden für spätere Übungen in diesem Kapitel und im Datenbankkapitel bei der Nutzung der Daten mit ADO.NET benötigt.

CREATE TABLE

Eine Tabelle *Bestellungen* können Sie leicht folgendermaßen anlegen:

```
CREATE TABLE [dbo].[Bestellungen] (
    [a-id] [int] IDENTITY (1, 1) NOT NULL ,
    [k-id] [int] NOT NULL ,
    [a-id] [int] NOT NULL ,
    [menge] [real] NOT NULL ,
```

```
[summe] [money] NOT NULL ,  
[bestell]datum [datetime] NOT NULL  
)
```

Ebenso werden die Tabellen *Artikel* und *Adressen* definiert:

```
CREATE TABLE [dbo].[Artikel] (  
    [id] [int] IDENTITY (1, 1) NOT NULL ,  
    [name] [nvarchar] (50) NULL ,  
    [preis] [numeric](18, 2) NULL ,  
    [nummer] [int] NULL  
)
```

```
CREATE TABLE [dbo].[Adressen] (  
    [id] [int] IDENTITY (1, 1) NOT NULL ,  
    [name] [nvarchar] (80) NULL ,  
    [strasse] [nvarchar] (80) NULL ,  
    [plz] [nvarchar] (5) NULL ,  
    [ort] [nvarchar] (50) NULL  
)
```

Sie haben jetzt einige neue Tabellen. Der Vorgang ist jedoch erklärungsbedürftig, denn der Teufel steckt im Detail. Hinter jedem Spaltennamen steht der Datentypbezeichner. Diesen sollten Sie genau kennen. Mehr Informationen dazu finden Sie in Abschnitt 5.5.1, »Datentypen in T-SQL« ab Seite 376. Über diese Datentypen hinaus sind jedoch weitere Einschränkungen und Definitionen möglich.

Bei der Schreibweise der Spaltennamen ist zu beachten, dass Minus- und Leerzeichen nur toleriert werden, wenn der Name in eckigen Klammern steht. Wenn Sie Tabellen mit den Visual Tools des SQL Servers entwerfen und dann SQL-Skripte generieren, stehen alle Namen in eckigen Klammern. Dies ist der sicherste Weg.



Wie an den Beispielen zu sehen ist, besteht eine Tabellendefinition eigentlich aus einer Auflistung von Spaltendefinitionen. Über die Struktur der Tabelle und die Namen der Spalten müssen Sie sich daher vorher klar werden. Neben den bereits besprochenen Datentypen gibt es fünf grundlegende Parameter, die auf Spalten angewendet werden können und als Einschränkungen zu verstehen sind:

Spalteneinschränkungen und -eigenschaften

► PRIMARY KEY

Erklärt diese Spalte zum Primärschlüssel. Nur eine Spalte einer Tabelle darf einen Primärschlüssel haben. Jeder Datensatz der Spalte muss eindeutig sein und dient der Referenzierung

der Datensätze. Die Eindeutigkeit können Sie mit dem Parameter `IDENTITY` sicherstellen. Primärschlüsselspalten werden automatisch indiziert.

- `FOREIGN KEY spaltenname REFERENCES fremdtabelle(fremdspalte)`

Diese Schlüsselspalte definiert eine Relation zu einer anderen Tabelle. Mehrere Spalten (max. 63) dürfen dieses Attribut tragen. Bei Relationen stellt sich die Frage, was mit Schlüsseln passiert, wenn ein »Ende« der Relation gelöscht wird. Ergänzen Sie `FOREIGN KEY` mit der Option `ON DELETE NO ACTION` oder `ON DELETE CASCADE`. Die zweite Form führt dazu, dass auch die verknüpften Datensätze gelöscht werden.

- `NOT NULL`

Der Wert `NULL` ist für die Felder dieser Spalte nicht erlaubt.

- `CHECK`

Dieser Parameter schränkt den Wertebereich des Inhalts ein. Mit `CHECK` werden die Werte auf bestimmte Bedingungen überprüft. Sie können jeden zulässigen Ausdruck angeben, der als Ergebnis `TRUE` oder `FALSE` zurückgeben kann. Wenn der Ausdruck bei einem `INSERT` oder `UPDATE` auf die Tabelle `FALSE` ergibt, wird der Eintrag der Werte zurückgewiesen. Beispielsweise können Sie bestimmte Postleitzahlbereiche beschränken:

```
CHECK (plz IN ('12683', '12587', '10407', '10115')
      OR pub_id LIKE '11[0-9][0-9][0-9]'),
```

Diese Definition lässt explizit die Postleitzahlen 12683, 12587, 10407, 10115 und alle Postleitzahlen mit 11 am Anfang zu.

- `UNIQUE`

Der Inhalt jedes Datensatzes muss eindeutig sein, bereits vorhandene Werte werden abgelehnt. 249 Spalten dürfen maximal dieses Attribut tragen. Der Parameter kann mit `IDENTITY` kombiniert werden, um die Eindeutigkeit sicherzustellen.

- `IDENTITY`

Am Anfang wurden die Schlüssel eingeführt, um Tabellen zu verknüpfen. Es gibt ein ganz wichtiges Kriterium für solche Schlüssel. Sie müssen eindeutig sein. Das Schlüsselwort `IDENTITY` erzwingt diese Bedingung, indem es für das betroffene Feld prüft, ob der Inhalt schon einmal vorhanden ist.

```
CREATE TABLE firmen
  ( key NUMERIC(10) IDENTITY(10000,1)
  )
```

Achten Sie auf den Datentyp. Er muss einen ausreichenden Wertebereich für alle Datensätze haben; `TINYINT` wäre hierfür kaum geeignet. Die Parameter hinter `IDENTITY` sind optional, ohne Angabe wird (1,1) angenommen. Der erste Wert ist der Startwert und der zweite die Schrittweite.

**Datentyp für
IDENTITY-Spalten**

`IDENTITY` ist nur innerhalb einer Tabelle eindeutig. Für globale, die gesamte Datenbank betreffende Eindeutigkeit nutzen Sie besser `UNIQUEIDENTIFIER` oder `ROWGUIDCOL`. Wenn Sie ein eindeutiges Feld über die gesamte Datenbank benötigen, nutzen Sie den Datentype `UNIQUEIDENTIFIER` und setzen den Startwert mit der Funktion `NEWID()`:

```
CREATE TABLE firmen
  ( key UNIQUEIDENTIFIER IDENTITY(NEWID())
  )
```

► **DEFAULT**

Damit bei neuen Feldern nichts schiefgeht, können Sie auch gleich einen Standardwert zuweisen. Wird kein Inhalt geliefert, trägt sich die Tabelle eben selbst einen Wert ein:

```
CREATE TABLE artikel
  ( artikel VARCHAR(60),
    text TEXT,
    preis NUM,
    bestelldatum DATETIME DEFAULT '01.01.1998' )
```

► **ROWGUIDCOL**

Mit dieser Eigenschaft erzwingen Sie eine global eindeutige Spalte. Nur eine Spalte einer Tabelle darf diese Eigenschaft haben. Der Begriff »global« ist wörtlich zu nehmen, es wird garantiert, dass dieser Wert auch bei keiner anderen Installation des SQL Servers verwendet wird.

Tabellen ändern, löschen und leeren

Wenn Sie eine Tabelle nicht mehr benötigen, sollten Sie sie zerstören. Sie sparen mit diesem Befehl Speicherplatz und Systemleistung:

**Tabellen löschen
mit DROP TABLE**

```
DROP TABLE temporaer
```

Tabellen leeren mit TRUNCATE TABLE

Wollen Sie nur sämtliche Datensätze löschen, verwenden Sie diesen Befehl:

```
TRUNCATE TABLE temporaer
```

Tabellenstruktur ändern mit ALTER TABLE

Manchmal kann es später notwendig sein, Änderungen an der Feldstruktur vorzunehmen. Der Befehl führt notwendige Änderungen an der Feldstruktur aus:

```
ALTER TABLE artike1 ADD farbe VARCHAR(10) NULL
```

Sie können mit ALTER keine neue Tabelle erzeugen, wenigstens eine Spalte muss bereits vorhanden sein.

5.5.3 Indizes

Tabellen werden oft und intensiv abgefragt. Auf die problematische Belastung des Servers wurde bereits hingewiesen. Eine Möglichkeit, effizienter zu arbeiten, sind Indizes. Ähnlich einem Buch, in dem der Index schnelleres Suchen ermöglicht, kann ein Feld einer Tabelle zum Indexfeld erklärt werden.

Arten von Indizes im SQL Server

Clustered versus Non clustered- Index

Der SQL Server unterstützt zwei Arten von Indizes: gruppierte und nicht gruppierte. »Gruppiert« (engl. clustered) steht hier für die Bildung des Indizes auf der Basis der Werte der indizierten Spalten. Das stellt sicher, dass ähnliche Werte physisch benachbart sind. Gruppierte Indizes beschleunigen die Suche nach Wertebereichen.

Nicht gruppierte Indizes arbeiten intern mit einer Baumstruktur. Dabei legt der erste Indexwert fest, ob der korrespondierende Datensatz in der oberen oder unteren Hälfte der Datenbank liegt. Der nächste Indexwert für die obere Hälfte schränkt die Suche wieder ein usw. So lassen sich Werte in der Hälfte der Zeit finden.

Bedenken Sie, dass jeder Schritt in Abbildung 5.17 eine durchaus rechenintensive Bedingung beinhalten kann.

Einsatzzweck

Ob Sie gruppierte oder nicht gruppierte Indizes verwenden, hängt vom Einsatzzweck ab. Gruppiert entspricht eher dem Inhaltsverzeichnis eines Buches. Alle Einträge sind nach der Reihenfolge ihrer Werte sortiert. Nicht gruppiert entspricht dagegen dem Index eines Buches, eine eigene Sortierung (meist nach dem Alphabet)

wird verwendet und keine Rücksicht auf die physische Struktur genommen. Aus Sicht des Suchbaumes ist der gruppierte Index schneller. Kommt es auf Tempo an, nutzen Sie diese Form. Ein Index erstellt allgemein eine sortierte Tabelle mit Verweisen auf die Haupttabelle. Eine logische Schlussfolgerung aus der Art der Indizierung ergibt die Anzahl zulässiger Indizes: Einen gruppierte Index kann es pro Tabelle nur einmal geben. Sie können dagegen bis zu 249 nicht gruppierte Indizes pro Tabelle erzeugen. Das erinnert wieder an das Buch: Es wird in der Regel nur ein zentrales Inhaltsverzeichnis geben, aber einen Index können Sie nach verschiedenen Merkmalen mehrfach aufbauen.

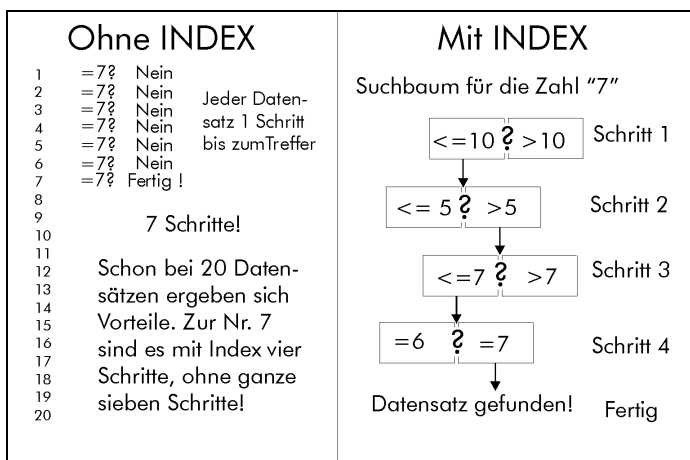


Abbildung 5.17: Vereinfachte Darstellung der Suchstrategie mit nicht gruppiertem Index

Performance-Tipp: Gehen Sie vorsichtig mit nicht gruppierten Indizes um. Sie brauchen mehr Speicherplatz. Befehle wie UPDATE oder INSERT werden deutlich langsamer, da jedes Mal der Index bedient werden muss. Beim gruppierten Index wird UPDATE und INSERT dagegen schneller, weil die Einordnung sehr einfach vorgenommen werden kann.



Jeder Index muss natürlich eine Spalte als Basis nutzen können. SQL kann sogar mehrere Spalten als Index heranziehen. So können Sie eine Tabelle mit Vor- und Zunamen erzeugen. Wenn Sie oft nach dem vollständigen Namen suchen, sollten Sie einen Index erstellen, der sich auf beide Felder bezieht. Es gibt außerdem die Möglichkeit, einen Index als eindeutig (engl. unique) zu bezeichnen. Damit verhindern Sie, dass Einträge doppelt vorkommen.

Indizes erzeugen und löschen

CREATE INDEX .. ON

Um einen Index zu erzeugen, muss schon eine Tabelle existieren. Dann nutzen Sie den folgenden Befehl, um in der Tabelle *Adressen* einen Index auf die Namen der Kunden zu erzeugen:

```
CREATE INDEX [IX_Adressen] ON [Adressen] ([name])
```

Sie können auch innerhalb des Befehls `CREATE TABLE` den Parameter `INDEX` verwenden. Manchmal macht die Erzeugung eines Index aber erst zu einem späteren Zeitpunkt Sinn. Befinden sich viele Daten in der Tabelle, kann der Vorgang einige Zeit in Anspruch nehmen.

DROP INDEX DROP INDEX

Sie können den Index auch wieder löschen. Da der Index selbst eine – wenn auch abhängige – Tabelle ist, eignet sich der Befehl `DROP`:

```
DROP INDEX [IX_Adressen].[name]
```



Wenn Sie einer Spalte einen Primärschlüssel mit `PRIMARY KEY` zuweisen, wird dies automatisch zu einem gruppierten Index auf Basis dieser Spalte führen.

Modifikation der Art des Index

CREATE INDEX CLUSTERED

Der erzeugte Index ist normalerweise nicht gruppiert. Sie können einen gruppierten Index erzeugen, indem das Schlüsselwort `CLUSTERED` benutzt wird:

```
CREATE CLUSTERED INDEX idx ON tabelle (spalte)
```

CREATE INDEX UNIQUE

Um einen eindeutigen Index zu erzeugen, der Doppelungen von Einträgen nicht zulässt, nutzen Sie das Schlüsselwort `UNIQUE`. Diese Einschränkung muss für eine Spalte getroffen werden, die nicht den Gruppierungsschlüssel enthält:

```
CREATE UNIQUE CLUSTERED INDEX idx ON tabelle (spalte)
```

Auch das Erzeugen eines Index mit zwei Spalten ist sehr einfach, hier verbietet sich natürlich der Modifizierer `CLUSTERED`:

```
CREATE INDEX idx ON tabelle (spalte1, spalte2)
```

5.5.4 Funktionen in T-SQL

MS Access verfügt über keine eigenen Funktionen für Berechnungen oder die Zeichenkettenverarbeitung, sondern nutzt hierfür

das Visual Basic-Modul. In T-SQL sind dagegen für alle Zwecke spezielle Funktionen definiert, die sich teilweise an SQL-92 anlehnen, oft jedoch darüber hinaus gehen.

Mathematische Funktionen

Die folgende Tabelle zeigt die mathematischen Funktionen, die Sie einsetzen können:

Funktion	Beschreibung
ABS(<i>n</i>)	Absoluter Betrag
ACOS(<i>f</i>)	Arcuskosinus
ASIN(<i>f</i>)	Arcussinus
ATAN(<i>f</i>)	Arcustangens
ATN2(<i>f1</i> , <i>f2</i>)	Arcustangens (Winkel) zwischen <i>f1</i> und <i>f2</i>
CEILING(<i>n</i>)	Die kleinste Ganzzahl größer oder gleich dem angegebenen numerischen Ausdruck.
COS(<i>f</i>)	Kosinus
COT(<i>f</i>)	Kotangens
DEGREES(<i>n</i>)	Umrechnung Radiant in Grad
EXP(<i>f</i>)	Potenz zur Basis e
FLOOR(<i>n</i>)	Die größte Ganzzahl kleiner oder gleich dem angegebenen numerischen Ausdruck.
LOG(<i>f</i>)	Natürlicher Logarithmus
LOG10(<i>f</i>)	Dekadischer Logarithmus
PI()	Konstante Pi
POWER(<i>n</i> , <i>p</i>)	Zahl <i>n</i> zur Potenz <i>p</i>
RADIANS(<i>f</i>)	Umrechnung in Radiant
RAND(<i>f</i>)	Zufallszahl
ROUND(<i>n</i> , <i>l</i> [, <i>e</i>])	Rundet Werte <i>n</i> mathematisch auf <i>l</i> Stellen. Wenn <i>e</i> 0 ist, wird gerundet, andernfalls gekürzt
SIGN(<i>n</i>)	Vorzeichen (gibt -1, 0 oder 1 zurück)
SIN(<i>f</i>)	Sinus
SQUARE(<i>f</i>)	Quadrat
SQRT(<i>f</i>)	Quadratwurzel
TAN(<i>f</i>)	Tangens

Tabelle 5.5: Mathematische Funktionen, die in Transact-SQL verwendet werden können

Anwendung Alle mathematischen Funktionen können mit der folgenden Syntax angewandt werden:

```
SELECT ergebnis = FUNCTION(parameters)
```

Ersetzen Sie dabei `FUNCTION(parameters)` durch den Namen der Funktionen und einen numerischen Ausdruck, der den folgenden Datentypen entsprechen muss: `decimal`, `numeric`, `integer`, `float`, `real`, `money`, `smallmoney`, `smallint`, `tinyint`.

Aggregat-Funktionen

Mit fünf einfachen Rechenoperationen kann auch spaltenweise gearbeitet werden. Die Auswahl beginnt wieder mit dem Befehl `SELECT`, dem eine Funktion nachgestellt wird. Die Anwendung ist nur mit numerischen Feldern sinnvoll.

Funktion	Beschreibung (Rückgabewert)
AVG	Average. Der Durchschnitt der Felder (INT, DECIMAL, MONEY oder FLOAT)
BINARY_CHECKSUM	Binärer Prüfsummenwert (INT)
COUNT	Die Anzahl der Felder (INT)
COUNT_BIG	Die Anzahl der Felder (BIGINT)
COUNT (*)	Repräsentiert die Anzahl aller Datensätze einer Tabelle (INT)
CHECKSUM	Prüfsummenwert (INT)
CHECKSUM_AGG	Prüfsumme einer Gruppe (INT)
GROUPING	Gruppierung im Zusammenhang mit CUBE oder ROLLUP
SUM	Summary. Die Summe der Felder (Addition).
MAX	Maximum. Das Feld mit dem größten Wert bestimmt das Ergebnis.
MIN	Minimum. Das Feld mit dem kleinsten Wert bestimmt das Ergebnis.
STDEV	Statistische Standardabweichung aller Werte der Liste
STDEVP	Auffüllung der statistischen Standardabweichung aller Werte
VAR	Statistische Varianz
VARP	Auffüllung der statistischen Varianz

Tabelle 5.6: Aggregat-Funktionen zum spaltenweisen Zusammenfassen

Systemfunktionen

Die Systemfunktionen liefern Angaben über Objekte und interne Zustände des SQL-Servers.

Funktion	Beschreibung (Rückgabetyt)
APP_NAME	Anwendungsname der aktuellen Sitzung (NVARCHAR(128))
COALESCE (<i>Ausdrucksliste</i>)	Liefert den ersten Nicht-NULL-Ausdruck. Siehe auch Befehl CASE.
COL_LENGTH (<i>"tabelle"</i> , <i>"spalte"</i>)	Die festgelegte Länge einer Spalte.
COL_NAME(<i>tabellen_id</i> , <i>spalten_id</i>)	Der Name der Spalte.
COLUMNPROPERTY (<i>id</i> , <i>spalten</i> , <i>eigenschaft</i>)	Erlaubt die Abfrage bestimmter Eigenschaften von Spalten.
CURRENT_TIMESTAMP	Liefert Datum und Uhrzeit (siehe GETDATE).
CURRENT_USER	Liefert den aktuellen Nutzer (siehe USER_NAME).
CURSOR_STATUS (<i>typ</i> , <i>name</i>)	Liefert den Typ der Rückgabe einer gespeicherten Prozedur zurück.
DATABASEPROPERTY	Gibt Eigenschaftswerte der Datenbank zurück
DATALength	Die tatsächliche Länge eines Ausdruck eines beliebigen Datentyps. Da die Datentypen varchar, varbinary, text und image Daten variabler Länge speichern können, ist DATALength in Verbindung mit diesen Datentypen besonders nützlich.
DB_ID, DB_NAME	Kennnummer und Name der Datenbank; Parameter ist jeweils der andere Wert
FORMATMESSAGE	Verknüpft eine Fehlermeldung mit Platzhaltern mit aktuellen Daten, um so die Ausgabe zu formatieren.
FILE_ID, FILE_NAME	Kennnummer und Name der Datei; Parameter ist jeweils der andere Wert
FILEGROUP_ID, FILEGROUP_NAME	Kennnummer und Name der Dateigruppe; Parameter ist jeweils der andere Wert
GETANSINULL	Die Standardeinstellung für die Zulässigkeit von NULL-Werten in der Datenbank.
HOST_ID, HOST_NAME	Kennnummer und Name des Host-Computer; Parameter ist jeweils der andere Wert

Tabelle 5.7: Systemfunktionen des SQL Server 2000

Funktion	Beschreibung (Rückgabetyt)
IDENT_CURRENT (<i>"tabelle"</i>)	Letzter Wert einer IDENTITY-Spalte
IDENT_INCR(<i>"tabelle"</i>)	Schrittweite einer IDENTITY-Spalte
IDENT_SEED(<i>"tabelle"</i>)	Startwert einer IDENTITY-Spalte
INDEX_COL(<i>"tabelle"</i> , <i>index_id</i> , <i>spalten_id</i>)	Name einer Indexspalte
IS_MEMBER(<i>gruppe</i> <i>rolle</i>)	Zeigt an, ob der aktuelle Nutzer Mitglied einer bestimmten Gruppe oder Rolle ist
ISDATE(<i>datumswert</i>)	Prüft, ob ein Ausdruck ein gültiges Datum ergibt
ISNULL(<i>Ausdruck</i> , <i>Wert</i>)	Ersetzt NULL-Werte durch den angegebenen Wert
ISNUMERIC	Prüft, ob ein gültiger numerischer Ausdruck vorliegt
NEWID	Erstellt einen eindeutigen Wert vom Typ UNIQUEIDENTIFIER
NULLIF(<i>Ausdruck1</i> , <i>Ausdruck2</i>)	Liefert NULL, wenn die Ausdrücke gleich sind
OBJECT_ID, OBJECT_ NAME	Kennnummer und Name eines Objekts
OBJECTPROPERTY	Gibt Eigenschaften aller möglichen Objekte zurück
PARSENAME	Gibt den Teil eines kompletten Objektnamens zurück
PERMISSIONS	Gibt die Zugriffsrechte auf Spalten zurück
SESSION_USER	Gibt den aktuellen Nutzer einer Tabelle zurück (meist »dbo«)
STATS_DATE	Datum der letzten Aktualisierung der Statustabellen (Systemtabellen) (DATETIME)
SYSTEM_USER	Gibt des Systemnutzer zurück (meist »sa«)
TYPEPROPERTY	Ermittelt Informationen zu einem Datentyp
USER_ID, USER_NAME	Kennnummer und Name des Datenbankbenutzers

Tabelle 5.7: Systemfunktionen des SQL Server 2000 (Forts.)

Sicherheitsfunktionen

Die folgenden Funktionen dienen der Kontrolle und Steuerung der Sicherheit des Datenbankservers oder der Datenbankanwendung:

Funktion	Beschreibung (Rückgabetyt)
IS_SRVROLEMEMBER(<i>rolle</i> , <i>name</i>)	Zeigt an, ob der aktuelle Nutzer oder ein bestimmter Benutzer Mitglied einer Serverrolle ist
SUSER_ID, SUSER_NAME	Kennnummer (SMALLINT) und Name (NVARCHAR(256)) des Serverbenutzers
USER_ID, USER_NAME	Kennnummer (SMALLINT) und Name (NVARCHAR(256)) des Datenbankbenutzers
USER	Ähnlich USER_NAME (CHAR)

Tabelle 5.8: T-SQL-Sicherheitsfunktionen

Text- und Bildfunktionen

Mit den Text- und Bildfunktionen werden große Datenmengen, beispielsweise für lange Texte oder Binärdaten, behandelt.

Funktion	Beschreibung (Rückgabetyt)
DATLENGTH(<i>Ausdruck</i>)	Liefert die Länge einer Spalte mit variabler Länge (INT)
READTEXT	Liest Text byteweise ein
PATINDEX(<i>"Muster"</i> , <i>Ausdruck</i>)	Liefert die Startposition einer bestimmten Zeichenfolge
SUBSTRING	Liefert einen Teil einer Textzeichenfolge
SET TEXTSIZE	Setzen der maximale Größe in Byte, die zurückgegeben wird.
TEXTPTR	Setzt einen Textzeiger
TEXTVALID	Prüft den Textzeiger auf Gültigkeit
UPDATETEXT	Überschreibt Text byteweise
WRITETEXT	Schreibt Text byteweise zurück

Tabelle 5.9: T-SQL-Text- und Bildfunktionen

Gruppierungen

Bei der Anwendung von Summenfunktionen werden Sie oft feststellen, dass Zwischensummen benötigt werden. SQL bietet dafür einige Ergänzungen des SELECT-Befehls. Wenn Sie sich beispielsweise die Artikel und die dazu passende Umsätze mit folgendem Befehl ansehen, erhalten Sie eine wenig hilfreiche Liste:

GROUP BY

```
SELECT [a-id], summe FROM bestellungen
```

	a-id	summe
1	4	99.0000
2	2	2850.0000
3	4	99.0000
4	1	199.9900

Es ist nahe liegend, für die Zusammenfassung der Umsätze der gehandelten Artikel die Summenfunktion heranzuziehen. Da Sie aber in der Bestellliste mehrere Artikel haben, ist eine Gruppierung sinnvoll. Dies wird mit `GROUP BY` erreicht:

```
SELECT [a-id], total = SUM(summe)
FROM bestellungen
GROUP BY [a-id]
```

	a-id	total
1	1	199.9900
2	2	2850.0000
3	4	198.0000

Das Ergebnis ist einfacher weiterzuverarbeiten. Wenn Sie diesen Befehl im Query Analyzer ausführen, erhalten Sie eventuell eine Warnung am Ende der Liste. Diese deutet darauf hin, dass weitere Reihen vorhanden waren, die wegen enthaltener `NULL`-Werte nicht beachtet wurden.

Gruppierungen im Detail

Gruppierungen können im Detail komplizierter sein als dieses Beispiel vermuten lässt. Wenn Sie beispielsweise die Namen der Artikel anzeigen möchten, wäre folgende Konstruktion möglich:

```
SELECT summe, [a-id], COUNT(*)
FROM Bestellungen
GROUP BY [a-id]
```

Dies führt zu einer Fehlermeldung! Der Fehler liegt in der Angabe der Spalte *Name*. Die Gruppierung bezieht sich auf die Spalte *id*. Jeder Datensatz, der in der Ausgabeliste angezeigt wird, muss eindeutig sein. Die Anzahl der Artikel ist aber uneindeutig, denn wenn mehrere Kunden denselben Artikel bestellen, kommt die Artikelnummer mehrfach vor. Welche Summe sollte SQL nun pro Datensatz der Ausgabeliste anzeigen?

Die folgende Konstruktion ist dagegen korrekt:

```
SELECT [a-id], SUM(summe), COUNT(*)
FROM Bestellungen
GROUP BY [a-id]
```


	(Kein Spaltenname)	a-id	(Kein Spaltenname)
1	199.9900	1	1
2	2850.0000	2	1
3	198.0000	4	2

Hier wird die Ausgabe der Namen auf einen bestimmten Namen (nämlich auf den ersten, nach dem Alphabet) eingeschränkt. Andersherum können Sie auch die Gruppierung auf mehrere Spalten ausdehnen. Stimmen alle Spalten mit allen Gruppen überein, ist die Ausgabe wieder ungruppiert. Hier die andere Variante:

COUNT(*)

```
SELECT [a-id], summe, COUNT(*)
FROM bestellungen
GROUP BY [a-id], summe
```

	a-id	summe	(Kein Spaltenname)
1	4	99.0000	2
2	1	199.9900	1
3	2	2850.0000	1

Die Funktion COUNT(*) bezieht sich bei Gruppen immer auf die Anzahl der Datensätze in der Gruppe. Die Ausgabe von 1 in der rechten Spalte im letzten Beispiel kann sinnvoll sein, wenn Sie die Anzahl pro Gruppe benötigen, nicht jedoch, wenn Sie parallel die Gesamtzahl der Datensätze ermitteln wollten.

Das Beispiel ist in einem anderen Zusammenhang dennoch sinnvoll. Lesen Sie weiter unten über den Parameter WITH CUBE, der sich gut mit GROUP BY kombinieren lässt und weitere Auswertungen ermöglicht.

Um mit Gruppen flexibel umgehen zu können, kann auch die Bildung von Gruppen Einschränkungen unterworfen werden. Ähnlich der WHERE-Bedingung können Sie hinter dem Schlüsselwort GROUP BY weitere Bedingungen folgen lassen, die durch HAVING eingeleitet werden. Die Bedingung wird gegen das Gruppenergebnis getestet:

HAVING

```
SELECT [a-id], SUM(summe) AS total, COUNT(*)
FROM bestellungen
GROUP BY [a-id]
HAVING COUNT(*) > 2
```

	a-id	total	(Kein Spaltenname)
1	1	748.9900	3
2	2	5898.0000	4

Hier werden nur die Artikel angezeigt, die mehr als zwei Bestellungen haben. Zugleich werden die Umsatzsummen ausgegeben. Damit das funktioniert, müssen Sie natürlich genug Bestelldatensätze erfasst haben.

Solche Auswahlmöglichkeiten können oft auch mit der `WHERE`-Bedingung erstellt werden. Der einzige (aber oft entscheidende) Unterschied besteht in der Performance. Das erste Beispiel ist deutlich schneller. Die Bildung von Gruppen kostet mehr Rechenleistung, als die Selektion nach Kriterien, denn sie basiert auf einer vorherigen Sortierung.

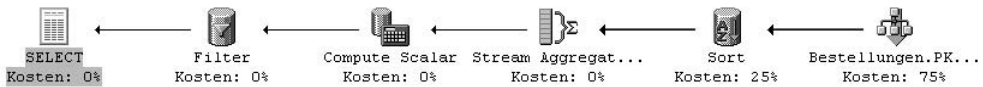


Abbildung 5.18: `GROUP BY` erzwingt eine Sortierung, die allein 25% der Leistung benötigt, die dieser Befehl verbraucht (siehe zweites Symbol von rechts).

Wenn Sie also zuerst selektieren (mit `WHERE`) und dann gruppieren (mit `GROUP BY`), steigern Sie die Leistung des Gesamtsystems. Der Flexibilität der `WHERE`-Bedingung kommt nicht zuletzt deshalb große Bedeutung zu. Der folgende Abschnitt zeigt, was Sie mit `WHERE` noch machen können.

WITH CUBE WITH ROLLUP

Oft werden Daten zu Berichten umgeformt und ausgegeben. Für gelegentliche Auswertung und die Nutzung über das Web sind die Operatoren `CUBE` und `ROLLUP` bestens geeignet.

Der `CUBE`-Operator erzeugt eine zusätzliche, berechnete Reihe. Die Berechnung selbst kann mit den Aggregat-Funktionen ausgeführt werden. In Kombination mit `GROUP BY` wird `CUBE` auf die Gruppen angewandt. Interessant wären nun neben der Ausgabe aller Artikel auch die Anzahl der Artikel pro Kunde und die Anzahl insgesamt. Statt mit drei `SELECT`-Befehlen schafft es `WITH CUBE` mit einem:

```
SELECT [a-id], SUM(summe) AS total, COUNT(*) ↵
FROM Bestellungen ↵
GROUP BY [a-id] WITH CUBE
```

	a-id	total	{Kein Spaltenname}
1	1	748.9900	3
2	2	5898.0000	4
3	3	490.0000	1
4	4	198.0000	2
5	NULL	7334.9900	10

Die Ausgabe wird nun um weitere Reihen ergänzt, deren Spalte *[a-id]* den Wert `NULL` und in den berechneten Spalten deren Summen enthält. Etwas komplexer fällt das Ergebnis aus, wenn man Zwischensummen benötigt. Dazu wird nach einer weiteren Spalte gruppiert, der `WITH CUBE`-Operator aber nur auf ein Selektionskriterium angewendet.

```
SELECT [a-id], SUM(summe) ←
FROM Bestellungen ↵
GROUP BY summe, [a-id] WITH CUBE
```

	a-id	{Kein Spaltenname}
1	1	99.0000
2	2	198.0000
3	4	198.0000
4	NULL	495.0000
5	1	199.9900
6	NULL	199.9900
7	1	450.0000
8	NULL	450.0000
9	3	490.0000
10	NULL	490.0000
11	2	5700.0000
12	NULL	5700.0000
13	NULL	7334.9900
14	1	748.9900
15	2	5898.0000
16	3	490.0000
17	4	198.0000

Rollt man die Liste der Ausgabewerte weiter nach unten, fällt auf, dass auch alle gegensätzlichen Kombinationen, mit `NULL` in der Spalte *a-id*, ausgegeben werden. Dies lässt sich mit `WITH ROLLUP` vermeiden:

```
SELECT [a-id], name, count(*) AS summe ←
FROM bestellungen ↵
GROUP BY [a-id], name WITH ROLLUP
```

	a-id	(Kein Spaltenname)
1	1	99.0000
2	2	198.0000
3	4	198.0000
4	NULL	495.0000
5	1	199.9900
6	NULL	199.9900
7	1	450.0000
8	NULL	450.0000
9	3	490.0000
10	NULL	490.0000
11	2	5700.0000
12	NULL	5700.0000
13	NULL	7334.9900

Wenn Sie statt zusätzlicher Reihen eine weitere berechnete Spalte benötigen, lohnt ein Blick auf den `COMPUTE`-Operator (weiter unten).

GROUPING Im Zusammenhang mit `CUBE` trifft man auch auf `GROUPING`. Denn die Unterscheidung der berechneten Spalten erfolgt anhand von `NULL`-Werten. Wie aber kann man nun Reihen identifizieren, die auf realen und nicht auf berechneten `NULL`-Werten basieren?

`GROUPING` ermittelt solche »realen« Nullwerte und gibt dann 1 zurück, ansonsten 0. Sie können die damit erzeugte Spalte zusätzlich auswerten, um die Zwischensummen korrekt zu erkennen. Das folgende Beispiel zeigt die Summen der eingegangenen Bestellungen, gruppiert nach Artikelnummern. Die Zwischensumme wird mit `WITH ROLLUP` gebildet. Nun gibt es eventuell Artikel, zu denen noch keine echten Bestellungen erfolgten. Das entsprechende Feld *summe* ist dann `NULL`¹. Um die Zwischensummen, die ebenfalls durch `NULL` gekennzeichnet sind, davon unterscheiden zu können, wird `GROUPING` auf die Spalte *summe* angewendet:

```
SELECT [a-id], SUM(summe), GROUPING([a-id]) 'grp'
FROM Bestellungen
GROUP BY summe, [a-id] WITH ROLLUP
```

¹ Das ist hier im Beispiel nicht der Fall, wenn Sie die Daten nicht von Hand ändern. In der Praxis wäre es denkbar, bereits den Warenkorb in der *Bestellungen*-Tabelle abzulegen und die Summe erst dann einzutragen, wenn der Kauf abgeschlossen wurde und eventuelle Rabatte berücksichtigt sind.

	a-id	(Kein Spaltenname)	grp
1	1	99.0000	0
2	2	198.0000	0
3	4	198.0000	0
4	NULL	495.0000	1
5	1	199.9900	0
6	NULL	199.9900	1
7	1	450.0000	0
8	NULL	450.0000	1
9	3	490.0000	0
10	NULL	490.0000	1
11	2	5700.0000	0
12	NULL	5700.0000	1
13	NULL	7334.9900	1

Im Ergebnis ist dann eine zusätzliche Spalte zu sehen, die bei den echten Zwischensummen eine »1« enthält.

Ein weiterer Operator für die Erstellung von Berichten ist TOP. TOP liefert eine Anzahl von Reihen aus einer größeren Auswahl zurück. So bietet der folgende Befehl nur die ersten sechs Datensätze an:

TOP

```
SELECT TOP 3 name FROM artikel
```

Sie können auch prozentual die Menge der auszugebenden Reihen angeben:

```
SELECT TOP 30 PERCENT name FROM artikel
```

Im Zusammenhang mit ORDER BY können Sie die Option WITH TIES angeben, die gleiche Datensätze nicht in die Berechnung der Anzahl der auszugebenden Datensätze einbezieht. Die Angabe von ORDER BY ist zwingend:

```
SELECT TOP 6 WITH TIES name
FROM Adressen
ORDER BY name
```

Eine Alternative dazu bietet die Voreinstellung für die Anzahl der Datensätze, SET ROWCOUNT:

SET ROWCOUNT

```
SET ROWCOUNT 3
SELECT name FROM Bestellungen
SET ROWCOUNT 0
```

Mit dem letzten Befehl `SET ROWCOUNT 0` wird die Ausgabe wieder für alle Datensätze freigegeben. Der Befehl setzt eine globale Variable, sodass die Beschränkung sich für alle nachfolgenden Ausgaben auswirkt. `TOP` wirkt dagegen nur innerhalb des aktuellen Befehls. Wenn Sie keine kompatible Software schreiben, sollten Sie `TOP` bevorzugen.

COMPUTE Der `COMPUTE`-Operator verwendet die bereits in Abschnitt »Aggregat-Funktionen« ab Seite 392 beschriebenen Aggregat-Funktionen. Das Ergebnis erscheint als weitere, praktisch virtuelle Tabelle nach der Ergebnisliste. Der Operator arbeitet, wie auch die Aggregat-Funktionen, reihenweise. `COMPUTE` kann mit dem Schlüsselwort `BY` ergänzt werden, um nach den Kriterien der angegebenen Spalte Gruppen zu bilden.

Das folgende Beispiel zeigt neben der gruppierten Artikel die Gesamtsumme an:

```
SELECT [a-id], summe ↵
FROM Bestellungen ↵
ORDER BY [a-id] ↵
COMPUTE SUM(summe)
```

	a-id	summe
1	1	199.9900
2	1	450.0000
3	1	99.0000
4	2	2850.0000
5	2	2850.0000
6	2	99.0000
7	2	99.0000
8	3	490.0000
9	4	99.0000
10	4	99.0000
sum		
1		7334.9900

Allein durch Anhängen von `BY [a-id]` ergibt sich folgendes Bild:

	a-id	summe
1	1	199.9900
2	1	450.0000
3	1	99.0000
sum		
1		748.9900
	a-id	summe
1	2	2850.0000
2	2	2850.0000
3	2	99.0000
4	2	99.0000
sum		
1		5898.0000
	a-id	summe
1	3	490.0000
sum		
1		490.0000
	a-id	summe
1	4	99.0000
2	4	99.0000
sum		
1		198.0000

Sie können hinter `COMPUTE` mehrere Aggregat-Funktionen angeben, getrennt durch Kommata. Bei der Anwendung gibt es einige Einschränkungen, die beachtet werden müssen:

Einschränkungen

- ▶ Mit `COMPUTE` berechnete Werte können nicht in `SELECT INTO` verwendet werden, denn das Ergebnis ist keine Tabelle.
- ▶ Die Spalten, auf die sich `COMPUTE` bezieht, müssen zuvor von einem `SELECT`-Befehl angesprochen werden.
- ▶ Wenn Sie mit `COMPUTE BY` gruppieren, muss dies dem `SELECT`-Befehl mit `ORDER BY` mitgeteilt werden. Die Reihenfolge der aufgeführten Spalten muss in beiden Befehlsteilen identisch sein.

Bei der Ausgabe werden Sie feststellen, dass die übliche Anordnung in einer Tabelle nicht mehr erfolgt.

Das Ergebnis ist nicht eine Ergebnisliste, sondern mehrere. Jede einzelne Liste enthält drei Spalten, dann folgt die Zwischensumme mit einer Spalte usw. Wenn Sie solche Listen extern abfragen möchten, müssen Sie sich also zwischen den Listen bewegen können. Das geht in ADO.NET beispielsweise mit dem *DataReader*. Informationen speziell dazu finden Sie in Abschnitt »Behandlung komplexer Abfragen mit DataReader« ab Seite 860.

Typumwandlungen

Wann immer es geht, wird der SQL Server 2000 Datentypen stillschweigend umwandeln und Sie nicht mit Fehlermeldungen traktieren. Sie können die korrekte Umsetzung aber gut unterstützen, wenn Sie zwei Befehle kennen, die zur Umwandlung herangezogen werden: `CONVERT` und `CAST`.

CONVERT Sie können die Datentypen eines Feldes nicht global ändern. Sie können aber mit `SELECT` die Umwandlung Feld für Feld vornehmen – auch und letztendlich für alle Felder.

```
SELECT id, CONVERT(CHAR(12), preis*1.16) + ' EURO' AS Preis
FROM Artikel
```

	id	Preis	
1	1	231.9884	EURO
2	2	3306.0000	EURO
3	3	162.3884	EURO
4	4	114.8400	EURO

Der Befehl `CONVERT` wandelt das betreffende Feld vom Typ `MONEY` in den Typ `CHAR` um. Die Länge, die `CHAR` nutzen soll, beträgt zwölf Zeichen. Der Platz muss ausreichen, um den Preis und das Wort »Euro« aufzunehmen, das bei dieser Gelegenheit gleich angehängt wurde. Außerdem wurde der Preis mit 1,16 multipliziert – aus dem Netto- wurde also ein Bruttopreis.

Manchmal müssen Sie explizit umwandeln, manchmal schafft es SQL intern. Die Umwandlung eines Feldes vom Typ `BIT` oder `MONEY` in eine Zeichenkette funktioniert nicht immer. Sie müssen in diesem Fall `CONVERT` verwenden. Dieser Befehl lässt sich so nicht ausführen:

```
SELECT "Der aktuelle Preis ist: " + preis FROM Artikel
```


Das Problem ist nicht der etwas »wilde« Einbau der Zeichenkette. Das ist durchaus erlaubt. Das Problem ist, dass das Feld price (Typ MONEY) nicht automatisch gewandelt wird. Es funktioniert folgendermaßen:

```
SELECT "Der aktuelle Preis ist: "+CONVERT(CHAR, preis) FROM Artikel
```

CAST entspricht in der Funktionsweise dem Befehl CONVERT. Mit CAST wird lediglich die ANSI-Kompatibilität gesichert. Wenn Sie in absehbarer Zeit eine Chance erkennen, einen anderen SQL-Server als den MS SQL Server 2000 zu verwenden, sollten Sie besser CAST verwenden. Die Anwendung ist ebenso einfach wie CONVERT:

CAST

```
SELECT name, 'Preis: ' + CAST(preis AS CHAR(12)) FROM Artikel
```

Zeichenketten-Funktionen

Die folgende Übersicht zeigt alle Zeichenketten-Funktionen. Die wichtigsten werden danach besprochen, eine exakte Auflistung finden Sie auch in der Referenz. Denken Sie beim Einsatz daran, dass die Abwicklung im SQL-Server unter Umständen schneller und effektiver ist, als durch eine Skriptsprache.

Funktion	Beschreibung (Rückgabetyt)
(Zkette1 + Zkette2)	Verknüpft zwei Zeichenketten miteinander
ASCII(Zeichenkette)	Zeigt den ASCII-Code des linken (ersten) Zeichens an.
CHAR(Ganzzahl)	Wandelt einen ASCII-Code-Wert in ein Zeichen um.
CHARINDEX(Muster, Ausdruck)	Liefert die Startposition der angegebenen Musterzeichenfolge <i>Muster</i> . Eine Musterzeichenfolge ist ein Zeichenausdruck. Der zweite Parameter ist ein Ausdruck, normalerweise ein Spaltenname, in dem SQL Server nach der Musterzeichenfolge sucht.
DIFFERENCE(Z1, Z2)	Zeigt den Unterschied zwischen den Werten von zwei Zeichenausdrücken, wie von der SOUNDEX-Funktion zurückgegeben. Siehe Beschreibung weiter unten.
LEFT(ZK, Position)	Gibt den linken Teil einer Zeichenkette zurück
LEN(ZK)	Gibt die Länge einer Zeichenkette zurück

Tabelle 5.10: T-SQL-Zeichenketten-Funktionen

Funktion	Beschreibung (Rückgabebetyp)
LOWER(<i>Zeichenkette</i>)	Wandelt die Zeichenkette in Kleinbuchstaben um
LTRIM(<i>Zeichenkette</i>)	Entfernt Leerzeichen vom linken Ende
NCHAR(<i>Zeichencode</i>)	Gibt ein Unicode-Zeichen mit dem angegebenen Wert zurück. Der <i>Zeichencode</i> kann zwischen 0 und 65.535 sein (16 Bit).
PATINDEX(<i>Muster, Ausdruck</i>)	Liefert die Startposition des ersten Auftretens der Musterzeichenkette im angegebenen Ausdruck bzw. NULL, wenn die Musterzeichenkette nicht gefunden wurde
QUOTENAME	Setzt einen Namen in Spaltenkennzeichen (»[]«) oder anderen Trennzeichen, beispielsweise Anführungszeichen.
REPLACE	Ersetzt eine Musterzeichenkette durch eine andere
REPLICATE(<i>Ausdruck, Ganzzahl</i>)	Wiederholt einen Zeichenausdruck so oft wie angegeben
REVERSE(<i>Zeichenkette</i>)	Dreht die Zeichenfolge um
RIGHT(<i>ZK, Position</i>)	Gibt den rechten Teil einer Zeichenkette zurück
RTRIM(<i>Zeichenkette</i>)	Entfernt Leerzeichen vom rechten Ende
SOUNDEX	Liefert einen vierstelligen (SOUNDEX) Code, um die Ähnlichkeit zweier Zeichenfolgen zu ermitteln. Siehe Beschreibung weiter unten.
SPACE(<i>Ganzzahl</i>)	Liefert eine Zeichenfolge mit wiederholten Leerstellen
STR(<i>Ausdruck, Länge, Dezimalstellen</i>)	Liefert eine Zeichenkette, die einem numerischen Wert entspricht; aus 17.5 wird die Zeichenkette »17.5«.
STUFF(<i>Ausdruck1, Start, Länge, Ausdruck2</i>)	Löscht die Anzahl der in <i>Länge</i> angegebenen Zeichen in <i>Ausdruck1</i> (bei <i>Start</i> beginnend) und fügt dann <i>Ausdruck2</i> in <i>Ausdruck1</i> bei <i>Start</i> ein
SUBSTRING(<i>Ausdruck, Start, Länge</i>)	Liefert einen Teil einer Zeichenkette oder Binärdaten. Der erste Parameter kann eine Zeichenkette oder Binärdaten, ein Spaltenname oder ein Ausdruck sein, der einen Spaltennamen enthält.

Tabelle 5.10: T-SQL-Zeichenketten-Funktionen (Forts.)

Funktion	Beschreibung (Rückgabetyt)
UPPER(<i>Zeichenkette</i>)	Wandelt alle Zeichen der Zeichenkette in Großbuchstaben um
UNICODE(<i>Zeichen</i>)	Wandelt ein Unicode-Zeichen in seinen dezimalen Zeichencode um

Tabelle 5.10: T-SQL-Zeichenketten-Funktionen (Forts.)

Einige Funktionen haben im .NET-Framework keine direkte Entsprechung, sodass sich der Einsatz unbedingt anbietet.

An dem Namen »Meier« sehen Sie eine häufige Schwierigkeit bei Datenbankabfragen – es gibt verschiedene Schreibweisen mit gleichem Klang. Die Dame im Call-Center, die einen Kunden nach dessen Namen suchen muss, wird für eine phonetische Suche dankbar sein. Ein wenig Komfort bieten die Funktionen DIFFERENCE und SOUNDEX, die eine solche phonetische Suche erlauben. Dabei ermittelt T-SQL einen fiktiven Schlüsselwert, der die phonetische Struktur des Wortes repräsentiert. Der Vergleich dieser Strukturen ergibt dann eine Ähnlichkeit. Folgendermaßen funktioniert es:

DIFFERENCE
SOUNDEX

```
SELECT name FROM partner WHERE DIFFERENCE(name, "Meier") > 2
```

Mit SOUNDEX geben Sie den Schlüsselwert aus, der zur Beurteilung verwendet wird. Das Schlüsselwort hat vier Stellen. Mit dem Vergleich DIFFERENCE > Wert oder DIFFERENCE < Wert wird die zulässige Abweichung festgelegt.

Hier ein paar Vergleiche:

```
Microsoft Mcrsft M262
Macrosoft Mcrsft M262 (stimmen immer überein)
Yahoo Yh Y000
Mahoo Mh M000 (stimmen bei DIFFERENCE>3 überein)
Nanosoft Nnsft N521
Megasoft Mgsft M221 (stimmen bei DIFFERENCE>2 überein)
```

Zuerst wird der erste Buchstabe beider Zeichenketten als Basis verwendet. Stimmt der überein, steht das erste Zeichen fest. Dann geht es mit den nächsten Konsonanten weiter. Vokale werden ignoriert (das Y zählt als Vokal!), es sei denn, der Vokal steht an erster Stelle, wie bei Yahoo.

SOUNDEX intern

Datum- und Zeit-Funktionen

**GETDATE
CURRENT_
TIMESTAMP**

In T-SQL fragen Sie das aktuelle Datum und die aktuelle Zeit mit GETDATE ab. Auch hier dient die Systemuhr des Servers als Basis. Sie können diese Funktion benutzen, um bei Datumsfeldern einen variablen Standardwert zu setzen:

```
CREATE TABLE logdata (
    name VARCHAR(50),
    date DATETIME DEFAULT GETDATE()
)
```

Sie können die Ausgabe des Datums vielfältig beeinflussen. Am einfachsten kann dazu wieder die Funktion CONVERT benutzt werden, die als dritten Parameter einen Code enthalten kann, der die Formatierung des Datums und der Zeit bestimmt.

Code	Standard	Ausgabeformat
0	Vorauswahl	mon dd yyyy hh:mi[AM PM]
1	USA I	mm/dd/yy
2	ANSI	yy.mm.dd
3	England	dd/mm/yy
4	Deutschland	dd.mm.yy
5	Italien	dd-mm-yy
6		dd mm yy
7	England, Brief	mon dd, yy
8	Zeit	hh:mi:ss
9	Millisekunde	mon dd yyyy hh:mi:ss:iii[AM PM]
10	USA II	mm-dd-yy
11	Japan	yy/mm/dd
12	ISO	yymmdd
13	Europa allgemein	dd mon yyyy hh:mi:ss:iii(24)
14	Zeit Europa	hh:mi:ss:iii(24)

Anmerkungen:
Die Codes 0, 9 und 13 geben das Jahr immer vierstellig zurück. Ansonsten können Sie zum Code 100 addieren, also statt 4 einfach 104 schreiben, um das Datum vierstellig zu machen.

Tabelle 5.11: Formate der Datums- und Zeitfunktionen

Nur die Codes 13 und 14 können die Uhr in unserem 24-Stunden-Format anzeigen. Die Zeichen bedeuten im Einzelnen:

- ▶ iii = Millisekunden, dreistellig
- ▶ ss = Sekunden
- ▶ mi = Minuten
- ▶ hh = Stunden[AM | PM] = entweder AM (vormittag) oder PM (nachmittag)
- ▶ (24) = 24-Stunden-Format
- ▶ dd = Tag
- ▶ mm = Monat in Ziffernform
- ▶ mon = Monat als Abkürzung (Jan, Feb, Nov usw.)
- ▶ yy = Jahr in zweistelliger Form
- ▶ yyyy = Jahr in vierstelliger Form

Die Funktion `CURRENT_TIMESTAMP` arbeitet identisch und bietet lediglich eine alternative Schreibweise.

Damit Sie gezielt Teile von Zeitangaben aus der Funktion `GETDATE` ziehen können, kennt T-SQL die Funktion `DATEPART` **DATEPART**

```
SELECT DATEPART(mm, GETDATE())
```

Statt »mm« können Sie einen der Codes in der folgenden Tabelle verwenden:

Code	Name	Beschreibung	Wertebereich
yy	Year	Jahr	1753 – 9999
qq	Quarter	Quartal	1 – 4
mm	Month	Monat	1 – 12
dy	Day of Year	Tag im Jahr	1 – 366
dd	Day	Tag	1 – 31
wk	Week	Woche	1 – 53
dw	Weekday	Wochentag	1 – 7
hh	Hour	Stunde	0 – 23
mi	Minute	Minute	0 – 59
ss	Second	Sekunde	0 – 59
ms	Millisecond	Millisekunde	0 – 999

Tabelle 5.12: Abkürzungen für die Steuerung der Funktion `DATEPART`

Ergänzend sei noch die Funktion `DATENAME` erwähnt, die genauso wie die Funktion `DATEPART` eingesetzt wird. Allerdings werden Zeichenketten statt Zahlen zurückgegeben. Für die Codes »mm« und **DATENAME**

»dw« ergibt sich die Ausgabe des Monatsnamens (January, December) und des Wochentages (Monday, Friday). Die Sprache ist vom Produkt abhängig, das heißt, beim deutschen SQL Server ist es deutsch. Sie können die Ausgabesprache im SQL Server aber auch einstellen. Verwenden Sie dazu das Kommando SET LANGUAGE:

```
SET LANGUAGE us_english
SELECT DATENAME(dw, GETDATE()) AS Tag
SET LANGUAGE german
SELECT DATENAME(dw, GETDATE()) AS Tag
```

	Tag
1	Friday
	Tag
1	Freitag

Die Ausgabe der beiden Ergebnisse stellt den Namen des Wochentags des aktuellen Datums dar.

**DATEDIFF
DATEADD**

Berechnungen mit Daten können recht aufwendig sein. Auch hier hilft SQL weiter und stellt zwei weitere Funktionen zur Verfügung: DATEDIFF und DATEADD. Damit werden Differenzen berechnet bzw. ein Zieldatum ermittelt. Die Tabelle *Bestellungen* enthält die Bestelltermine der Artikel in *bestelldatum*. Sie können einfach die Tage ermittelt, die die Bestellung eines Artikels zurück liegt:

```
SELECT [a-id],
       'Ist ' + CONVERT(VARCHAR(3),
                        DATEDIFF(dd, bestelldatum, GETDATE()))
       + ' Tage seit der letzten Bestellung'
       AS Zeitraum
FROM Bestellungen
```

	a-id	Zeitraum
1	4	Ist 31 Tage seit der letzten Bestellung
2	2	Ist 5 Tage seit der letzten Bestellung
3	4	Ist 100 Tage seit der letzten Bestellung
4	1	Ist 0 Tage seit der letzten Bestellung
5	1	Ist 1 Tage seit der letzten Bestellung
6	3	Ist 2 Tage seit der letzten Bestellung
7	1	Ist 3 Tage seit der letzten Bestellung
8	2	Ist 30 Tage seit der letzten Bestellung
9	2	Ist 25 Tage seit der letzten Bestellung
10	2	Ist 15 Tage seit der letzten Bestellung

Das Ergebnis leidet zwar noch an grammatikalischen Schwächen, ist aber schon recht brauchbar. Vor allem aber ist die so erzeugte Ergebnisliste schneller produziert als mit einer Programmiersprache.

5.5.5 Arbeiten mit Datenbankzeigern

Die Möglichkeit, Abfragen von Datensätzen über Zeiger zu steuern, hat mit dem zeiger- und verbindungslosen Konzept von ADO.NET an Bedeutung gewonnen. Zeiger erlauben es, eine umfangreiche und möglicherweise aufwändig berechnete Abfrage zeilenweise zu durchlaufen. Verschiedene Zeigertypen werden eingesetzt, um ein der Anwendung entsprechendes, optimiertes Laufzeitverhalten zu ermöglichen. Aus Sicht von ADO.NET ist der Einsatz von Zeigern nicht notwendig, weil die abgekoppelten Datensätze in *DataSet*-Objekten gehalten werden, die den direkten Zugriff auf alle Elemente erlauben. Die Optimierung des Zugriffs findet über Sichten (*DataView*) statt. Das funktioniert, solange die Datenmenge relativ klein ist. Gerade bei ASP.NET ist der Einsatz jedoch kritisch zu betrachten, da viele Benutzer parallel auf die Datenbank zugreifen und jeweils eigene *DataSet*-Objekte erzeugen. Wenn diese nun sehr große Datenmengen enthalten, multipliziert sich der Speicherverbrauch mit der Anzahl der Benutzer. In solchen Fällen kann ein leistungsfähiger Datenbankserver mit besserem Cache-Verhalten die bessere Wahl für den Aufbau der Abfragespeicher sein. An diesen Stellen kommen die Datenbankzeiger ins Spiel, denn damit können vorselektierte Daten gezielt durchlaufen werden.

**Zeiger und
ADO.NET**

Das Prinzip der Datensatzzeiger

Wenn Sie mit `SELECT` eine Tabelle (oder mehrere Tabellen) abfragen, erhalten Sie eine Sammlung von Datensätzen als Ergebnis – die Ergebnisliste. Zeiger (engl. Cursor) sind Erweiterungen dieser Ergebnisliste. Mit Zeigern können Sie sich praktisch im Anschluss an die erfolgte Abfrage innerhalb der Ergebnisliste bewegen und so gezielt einzelne Datensätze extrahieren. Datenbankzeiger haben folgende Aufgaben:

- ▶ Positionierung einer bestimmten Reihe
- ▶ Holen der Daten aus der aktuell angezeigten Position

- Änderung von Daten an der aktuellen Position
- Anzeige von Änderungen, die durch Zugriffe anderer Nutzer auf die Originaldaten erfolgten.

Deklaration und Verwendung in T-SQL

DECLARE CURSOR In T-SQL erstellen Sie einen Zeiger mit dem Befehl `DECLARE CURSOR`. Das folgende Beispiel zeigt die Erstellung eines Zeigers und seine Anwendung:

```
DECLARE MeinZeiger CURSOR FOR
    SELECT name FROM Artikel
        WHERE name LIKE '%Win%'
        ORDER BY name
OPEN MeinZeiger
FETCH NEXT FROM MeinZeiger
WHILE @@FETCH_STATUS = 0
    BEGIN
        FETCH NEXT FROM MeinZeiger
    END
CLOSE MeinZeiger
DEALLOCATE Meinzeiger
```

	name
1	Windows XP Home
	name
1	Windows XP Pro

Die Ausgabe liefert alle entsprechenden Artikel nicht in einer Ergebnisliste, wie bisher, sondern in vielen Ergebnislisten mit je einem Titel. Der Inhalt der `WHILE`-Schleife bestimmt, wie sich die Ausgabe verhält. Interessant sind die vielfältigen Berechnungsmöglichkeiten, die Ihnen bei einem einfachen `SELECT`-Befehl nicht zur Verfügung stehen.

Der Befehl `DECLARE CURSOR` kann mit verschiedenen Parametern kombiniert werden:

- `INSENSITIVE`

Wenn eine Deklaration mit diesem Parameter erfolgt, werden die Daten in einer temporären Tabelle *tempdb* abgelegt. Änderungen an den Basistabellen wirken sich nicht auf das Abfrageergebnis aus. Der Zeiger ist außerdem nicht änderbar, das heißt, die Daten können nur gelesen werden.

► SCROLL

Wenn angegeben, sind alle Bewegungsoperatoren (siehe `FETCH`) erlaubt, ohne Angabe nur `NEXT` (nur Vorwärtsbewegung).

► READ_ONLY

Verhindert, dass Änderungen an der Liste vorgenommen werden können.

► UPDATE [OF *spalten*]

Zeigt die Spalten an, die geändert werden dürfen.

► LOCAL

Sagt, dass der Zeiger nur lokal in der Prozedur oder dem Trigger gilt, wo er definiert wurde.

► GLOBAL

Dieser Zeiger ist immer verfügbar.

► FORWARD ONLY

Der Zeiger kann nicht rückwärts bewegt werden.

► STATIC

Änderungen an den Basistabellen, die ursprünglich die Ergebnisliste erzeugten, werden nicht weitergegeben und ändern die Liste nicht.

► KEYSET

Wenn der Zeiger geöffnet wurde, werden Änderungen von Schlüsselwerten nicht mehr berücksichtigt. Andere Änderungen werden aber weitergereicht. Mit `INSERT` eingefügte Reihen werden nicht angezeigt.

► DYNAMIC

Alle Änderungen werden sofort angezeigt.

► SCROLL_LOCKS

Alle Datensätze (Reihen), die sich in der Ergebnisliste befinden, werden in der Originaltabelle gesperrt, bis der Zugriff beendet wird. Änderungen und das Löschen von Datensätzen werden immer ausgeführt.

► OPTIMISTIC

Die Datensätze werden nicht gesperrt, Änderungen an Datensätzen, die zwischenzeitlich gelöscht wurden, gehen verloren.

Mit dem Befehl `FETCH` steuern Sie den Zeiger. Hinter `FETCH` kann eine der folgenden Anweisungen stehen:

- ▶ `NEXT`
Setzt den Zeiger auf den nächsten Eintrag.
- ▶ `PRIOR`
Setzt den Zeiger einen Eintrag zurück.
- ▶ `FIRST`
Setzt den Zeiger auf den ersten Eintrag.
- ▶ `LAST`
Setzt den Zeiger auf den letzten Eintrag.
- ▶ `ABSOLUTE`
Setzt den Zeiger auf einen bestimmten Eintrag.
- ▶ `RELATIVE`
Bewegt den Zeiger relativ zur aktuellen Position.

An die Anweisung zur Bewegung schließt sich das Schlüsselwort `FROM` an, das den Zeiger oder eine Zeigervariable definiert. Als letztes kann (optional) das Schlüsselwort `INTO` folgen, mit dem der Inhalt der Reihe in eine Variable übertragen wird (beispielsweise mit `FETCH NEXT FROM Zeiger INTO @wert`).

CLOSE DEALLOCATE

`CLOSE` schließt den Zugriff auf den Zeiger, `DEALLOCATE` zerstört den Zeiger endgültig.

Die erfolgreiche Arbeit mit Datensatzzeigern setzt einige Kenntnisse über die interne Arbeitsweise voraus. Grundsätzlich sollten Sie sich überlegen, wie mit den Ergebnislisten umgegangen wird. Wenn die Ergebnisliste relativ klein ist und keine Änderungen zurückgeschrieben werden müssen, nutzen Sie die Standardeinstellungen und geben keine der oben gezeigten Optionen an. Die Standardeinstellung ist auch ideal, wenn Sie die gesamte Ergebnisliste in die Zielapplikation übertragen und dort weiter verarbeiten. Dynamische Zeiger (`DYNAMIC`) arbeiten auch mit großen Listen gut, wenn der Nutzer nicht gleichzeitig sehr viele dieser Datensätze zur Anzeige bringt. Dynamische Zeiger sind schneller als statische (`STATIC`) oder schlüsselorientierte (`KEYSET`). Statische und schlüsselorientierte Zeiger verwenden die interne temporäre Tabelle *tempdb*; dies kostet Prozessorleistung, Festplattenzugriffe und nicht zuletzt zusätzlichen Speicherplatz. Absolute Bewegungen des Zeigers erzwingen ebenso diese Vorgehensweise.

5.5.6 Erweiterter Umgang mit SELECT

In der Einführung zu SQL wurde `SELECT` nur in seiner einfachen Abfrageform verwendet. Es gibt mehrere Techniken, komplexere Abfragen zu gestalten:

- ▶ Kombination von zwei `SELECT`-Befehlen, so genannte »Sub-selects«.
- ▶ Kombination von Abfrageergebnissen mit `UNION`
- ▶ Verknüpfung von Tabellen mit `JOIN` und verschiedenen Varianten von `JOIN`

SELECT verschachteln (Subselect)

Die Darstellung des `SELECT`-Befehls im letzten Abschnitt verschweigt, dass im Umgang mit komplexen Tabellen solche einfachen Abfragen kaum den Ansprüchen gerecht werden. Oft wird das Abfrageresultat einer Tabelle als Auswahlkriterium für eine weitere Abfrage verwendet. Solche praktisch zusammenhängenden Abfragen lassen sich mit einem verschachtelten `SELECT` erledigen.

Wenn Sie beispielsweise die Artikel abfragen, die einen Preis unter 1.000 € haben, ist der folgende Befehl ausreichend:

```
SELECT id, preis ↵  
FROM Artikel ↵  
WHERE preis < 1000
```

	id	preis
1	1	199.99
2	3	139.99
3	4	99.00

In diesem Zusammenhang wäre es interessant zu wissen, welche Bestellungen für diese Artikel eingegangen sind. Dazu schreiben Sie die ID-Nummern ab und fragen die Tabelle *Bestellungen* ab.

```
SELECT * ↵  
FROM bestellungen ↵  
WHERE [a--id] = 1 OR [a--id] = 3 [a--id] = 4
```

Sie können nun beide Befehle kombinieren, sodass nur die Bestellungen ausgegeben werden, bei denen Artikel bestellt wurden, die einen Preis unter 200 € haben:

```
SELECT * FROM bestellungen
  WHERE [a-id] IN
    (SELECT [id]
     FROM Artikel
     WHERE preis < 200)
ORDER BY [a--id] ASC
```

	id	k-id	a-id	menge	summe	bestelldatum
1	73	1	1	1.0	199.9900	2002-08-09 00:56:25.963
2	75	1	1	3.0	450.0000	2002-08-08 01:33:01.000
3	77	4	1	1.0	99.0000	2002-08-06 01:33:38.000
4	76	4	3	2.0	490.0000	2002-08-07 01:33:32.000
5	21	3	4	1.0	99.0000	2002-07-09 00:56:26.000
6	72	1	4	1.0	99.0000	2002-05-01 00:56:26.000

Subselects Diese Abfrage wird unabhängig von einer konkreten ID-Nummer des Titels ausgeführt und damit automatisiert. Verschachtelte SELECT-Befehle werden in Klammern verpackt:

```
(SELECT [a--id]
 FROM artikel
 WHERE preis < 200)
```

Zuerst erstellt diese Abfrage eine Liste von Artikeln, deren Preis kleiner als 200 € ist. SQL speichert diese Liste und führt dann den umgebenden SELECT-Befehl für jeden Datensatz der Liste aus. Dies ist auch die Ursache für die Wahl des IN-Schlüsselwortes. Das umgebende Kommando könnte auch so aussehen:

```
SELECT *
  FROM bestellungen
  WHERE [a--id] IN (1, 3, 4)
```

Mit den entsprechenden logischen Operatoren können Sie die Auswahl auch leicht negieren:

```
SELECT * FROM bestellungen
  WHERE [a--id] NOT IN
    (SELECT [id]
     FROM Artikel
     WHERE preis < 200)
ORDER BY [a--id] ASC
```

	id	k-id	a-id	menge	summe	bestelldatum
1	67	1	2	1.0	2850.0000	2002-08-04 00:56:26.000
2	78	4	2	1.0	2850.0000	2002-07-10 01:33:52.000
3	79	4	2	1.0	2850.0000	2002-07-15 01:34:32.000
4	80	4	2	1.0	2850.0000	2002-07-25 01:34:36.000

Dieser Befehl zeigt alle teuren und – trotzdem – bestellten Titel an.

Der verschachtelte `SELECT`-Befehl muss genau eine Spalte zurückgeben, denn SQL muss jedes Element der Liste auf den umgebenden Befehl anwenden. Bei mehreren Spalten ist keine eindeutige Auswahl mehr gegeben. Der folgende Befehl funktioniert nicht:

Einschränkungen

```
SELECT * FROM Bestellungen
WHERE [a--id] IN
( SELECT [id], preis
  FROM Artikel
  WHERE preis < 200)
```

Auch das folgende Beispiel enthält einen typischen Fehler. Da der verschachtelte Befehl mehrere Datensätze zurückgibt, können Sie in der umgebenden `WHERE`-Bedingung nicht den Operator `»=«` anwenden.

```
SELECT * FROM Bestellungen
WHERE [a--id] =
( SELECT [id]
  FROM Artikel
  WHERE preis < 200)
```

Nur der `IN`-Operator erlaubt mehrere Parameter. Alternativen zu `IN` haben Sie bereits kennen gelernt. Eine funktionierende Variante wäre die folgende:

IN

```
SELECT * FROM Bestellungen
WHERE [a-id] = ANY
( SELECT [id]
  FROM Artikel
  WHERE preis < 200)
```

	id	k-id	a-id	menge	summe	bestelldatum
1	21	3	4	1.0	99.0000	2002-07-09 00:56:26.000
2	72	1	4	1.0	99.0000	2002-05-01 00:56:26.000
3	73	1	1	1.0	199.9900	2002-08-09 00:56:25.963
4	75	1	1	3.0	450.0000	2002-08-08 01:33:01.000
5	76	4	3	2.0	490.0000	2002-08-07 01:33:32.000
6	77	4	1	1.0	99.0000	2002-08-06 01:33:38.000

ANY ANY liefert also einzeln die Datensätze zu einem einfachen Vergleich zurück. Als Alternative zu NOT IN könnte die folgende Konstruktion dienen:

```
SELECT * FROM artikel ↵
WHERE [a--id] <> ALL ↵
( SELECT [a--id] ↵
FROM artikel ↵
WHERE preis < 10 )
```

	id	k-id	a-id	menge	summe	bestelldatum
1	67	1	2	1.0	2850.0000	2002-08-04 00:56:26.000
2	78	4	2	1.0	2850.0000	2002-07-10 01:33:52.000
3	79	4	2	1.0	2850.0000	2002-07-15 01:34:32.000
4	80	4	2	1.0	2850.0000	2002-07-25 01:34:36.000

ALL Hier kommt als Auswahlkriterium für den Subselect ALL zum Einsatz.

JOIN

JOIN ... ON Verbindungen zwischen Tabellen werden typischerweise mit JOIN hergestellt. Auf JOIN wird weiter unten noch ausführlich eingegangen. Das folgende Beispiel ermittelt zu jeder Bestellung auch gleich den Namen des Bestellers:

```
SELECT b.[a-id], b.bestelldatum, a.name ↵
FROM Bestellungen b ↵
JOIN Adressen a ON a.[id] = b.[k-id] ↵
WHERE b.summe < 200 ↵
ORDER BY bestelldatum
```

	a-id	bestelldatum	name
1	4	2002-05-01 00:56:26.000	Jörg Krause
2	4	2002-07-09 00:56:26.000	Lukas Werlich
3	1	2002-08-06 01:33:38.000	Carolin Scholz
4	1	2002-08-09 00:56:25.963	Jörg Krause

Einen ähnlichen Effekt erzielen Sie mit einem verschachtelten SELECT-Befehl, diesmal in der Liste der Ausgabespalten:

```
SELECT b.[a-id], b.bestelldatum, ␣
      (SELECT name FROM Adressen a ␣
       WHERE a.id = b.[k-id]) AS name ␣
FROM Bestellungen b ␣
WHERE summe < 200 ␣
ORDER BY bestelldatum
```

Sie können verschachtelte SELECT-Befehle anwenden, wenn nur auf eine Spalte zugegriffen werden soll. JOIN funktioniert auch mit mehreren Spalten. Die verschachtelte Version würde außerdem NULL-Spalten mitführen, was JOIN verhindert. Solche Verknüpfungen mit einer äußeren Referenz werden korrelierende Unterabfragen genannt. Die Ergebnisse der beiden Beispiele zeigen die Vorteile der Verknüpfung mit JOIN.

Im folgenden Beispiel werden die Artikelnamen aus der *Artikel*-Tabelle zurückgegeben, zu denen noch keine Bestellung erfolgte. Ein einfacher JOIN würde kein Ergebnis liefern, weil NULL-Zeilen keine Übereinstimmung bringen können.

FULL OUTER JOIN

```
SELECT b.[a-id], a.name ␣
FROM Artikel a FULL OUTER JOIN Bestellungen b ␣
ON b.[a-id] = a.id ␣
WHERE b.[a-id] IS NULL ␣
OR a.id IS NULL
```

	a-id	name
1	NULL	SharpTemple Handbuch

Entfernen Sie die WHERE-Bedingung, sehen Sie alle Reihen der Tabelle *Bestellungen*. Um nun steuern zu können, ob die linke oder rechte Tabelle unbedingt angezeigt werden soll – also auch dann, wenn NULL enthalten ist –, können Sie die Schlüsselworte LEFT und RIGHT verwenden.

**LEFT OUTER JOIN
RIGHT OUTER JOIN**

Tauschen Sie das Schlüsselwort LEFT gegen RIGHT aus, wird der Prozess entsprechend zuerst auf die rechte Tabelle angewandt. Die entsprechenden Reihen aus der linken Tabelle sind dann NULL. Je nach Umfang der Daten sieht das dann folgendermaßen aus:

**LEFT JOIN
RIGHT JOIN**

	a-id	name		a-id	name
1	4	Big Color Keyboard	1	1	Windows XP Pro
2	2	MSDN Abo Professional	2	1	Windows XP Pro
3	4	Big Color Keyboard	3	1	Windows XP Pro
4	1	Windows XP Pro	4	2	MSDN Abo Professional
5	1	Windows XP Pro	5	2	MSDN Abo Professional
6	3	Windows XP Home	6	2	MSDN Abo Professional
7	1	Windows XP Pro	7	2	MSDN Abo Professional
8	2	MSDN Abo Professional	8	3	Windows XP Home
9	2	MSDN Abo Professional	9	4	Big Color Keyboard
10	2	MSDN Abo Professional	10	4	Big Color Keyboard
			11	NULL	SharpTemple Handbuch

JOIN
INNER JOIN
RIGHT JOIN

FULL OUTER JOIN
LEFT JOIN

Bei inneren Verbindungen (INNER JOIN) ist die Verbindung äquivalent zur Angabe der Bedingung innerhalb von WHERE. Bei einer äußeren Verbindung (OUTER JOIN) wird dagegen die Verknüpfung mit der JOIN-Bedingung ausgeführt.



Anwendungen des Schlüsselworts OUTER JOIN können sehr kompliziert sein. Wenn die Abfrage nicht wie erwartet funktioniert, entfernen Sie zuerst die WHERE-Bedingungen, um die »Rohdaten« beurteilen zu können.

Korrelierende Namen

Komplexe Abfragen mit JOIN sind oft sinnvoll und vereinfachen den Code. Sie sind aber kein Allheilmittel gegen haarsträubende WHERE-Bedingungen. Oft verliert man den Überblick allein durch Nennung mehrerer Tabellennamen, die zur eindeutigen Adressierung der Felder geschrieben werden müssen. Hilfreich ist ein weiteres Schlüsselwort: CROSS JOIN.

CROSS JOIN

Bei der Verbindung von zwei Tabellen kann auch ein Zugriff auf beide Spaltensätze der Tabellen (oder mehrerer Tabellen) erforderlich sein. CROSS JOIN erledigt dies:

```
SELECT * FROM Artikel CROSS JOIN Adressen
```

Im Ergebnis erhalten Sie alle Spalten von beiden Tabellen und eine Mischung aus jeder mit jeder Spalte (die man mit WHERE sinnvoll einschränken kann). Gleiche Spaltennamen werden übernommen. Wenn Sie gezielt darauf zugreifen möchten, sollten numerische Spaltenindizes verwendet werden.

	id	name	preis	nummer	id	name	strasse	plz	ort
1	1	Windows XP Pro	199.99	1325	1	Jörg Krause	Dornenweg 33	12683	Berlin
2	2	MSDN Abo Professional	2850.00	6668	1	Jörg Krause	Dornenweg 33	12683	Berlin
3	3	Windows XP Home	139.99	1326	1	Jörg Krause	Dornenweg 33	12683	Berlin
4	4	Big Color Keyboard	99.00	7738	1	Jörg Krause	Dornenweg 33	12683	Berlin
5	5	SharpTemple Handbuch	499.00	8837	1	Jörg Krause	Dornenweg 33	12683	Berlin
6	1	Windows XP Pro	199.99	1325	3	Lukas Werlich	Spielallee 11	89487	München
7	2	MSDN Abo Professional	2850.00	6668	3	Lukas Werlich	Spielallee 11	89487	München
8	3	Windows XP Home	139.99	1326	3	Lukas Werlich	Spielallee 11	89487	München
9	4	Big Color Keyboard	99.00	7738	3	Lukas Werlich	Spielallee 11	89487	München
10	5	SharpTemple Handbuch	499.00	8837	3	Lukas Werlich	Spielallee 11	89487	München
11	1	Windows XP Pro	199.99	1325	4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf
12	2	MSDN Abo Professional	2850.00	6668	4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf
13	3	Windows XP Home	139.99	1326	4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf
14	4	Big Color Keyboard	99.00	7738	4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf
15	5	SharpTemple Handbuch	499.00	8837	4	Carolin Scholz	Eichenwall 4	40748	Düsseldorf

UNION

Manchmal gibt es keine Möglichkeit, in einer einzigen Abfrage alle Daten zu ermitteln. Dann müssten Sie die Abfragen einzeln erstellen, die Daten in eine Programmier- oder Skriptumgebung überführen und dort Datensatzweise kombinieren. Alternativ können Sie SELECT-Befehle mit dem Schlüsselwort UNION kombinieren. Das Ergebnis entspricht dann der Summe oder Kombination der Abfragen. Oft kann UNION durch eine WHERE-Bedingung und den Operator OR ersetzt werden. Kombinieren lassen sich nur die Ergebnisse von zwei zueinander kompatiblen Spalten. So sind beispielsweise SMALLINT und INT kompatibel, DATETIME und CHAR dagegen nicht.

UNION

	name
1	Big Color Keyboard
2	Carolin Scholz
3	Jörg Krause
4	Lukas Werlich
5	MSDN Abo Professional
6	SharpTemple Handbuch
7	Windows XP Home
8	Windows XP Pro

Hier eine Anwendung des UNION-Operators:

```
SELECT name FROM Artikel
UNION
SELECT name FROM Adressen
```

Den letzten SELECT-Befehl können Sie mit ORDER BY kombinieren. Für die Kombination als Kreuzprodukt (statt der Summe) bietet sich CROSS JOIN an, das zuvor bereits beschrieben wurde.

Abruf von Daten im XML-Format

FOR XML Wenn Daten im XML-Format benötigt werden, können diese über ADO.NET leicht umgewandelt werden. ADO.NET verwendet generell XML als internes Speicherformat. Wenn die Weiterverarbeitung aber nicht Gegenstand Ihrer Applikation ist, sondern nur ein Abruf von Daten stattfindet, bietet sich der Einsatz der Klausel **FOR XML** an. Der Einsatz erfolgt im Zusammenhang mit **SELECT**. Ergänzend ist einer der folgenden drei Parameter anzugeben:

► **RAW**

Mit dieser Option wird das Abfrageergebnis in Zeilen mit dem generischen Tag `<row/>` festgelegt. Eine dreispaltige Tabelle sähe in XML dann folgendermaßen aus:

```
<row CustomerID="ALFKI" OrderID="10643"
      OrderDate="1997-08-25T00:00:00"/>
<row CustomerID="ANATR" OrderID="10308"
      OrderDate="1996-09-18T00:00:00"/>
```

► **AUTO**

Diese Methode gibt einen einfachen XML-Baum zurück. Das ist sinnvoll, wenn mehrere Tabellen abgefragt werden.

```
<Customers CustomerID="ALFKI" ContactName="Maria Anders">
  <Orders OrderID="10643"/>
  <Orders OrderID="10692"/>
  <Orders OrderID="10702"/>
  <Orders OrderID="10835"/>
  <Orders OrderID="10952"/>
  <Orders OrderID="11011"/>
</Customers>
```

► **EXPLICIT**

Diese Option definiert die Art und Weise der Gestaltung des XML-Baumes. Die Bildung der Tag-Struktur basiert auf der Angabe von Alias-Namen mit einer sehr speziellen Syntax.

Alle drei Parameter lassen sich noch mit einem Modifikator verknüpfen, der die Ausgabe steuert:

► **ELEMENTS**

Hiermit wird erzwungen, dass alle Spalten als Elemente (Tags) dargestellt werden, nicht als Attribute.

► BINARY BASE64

Binärdaten werden mit dieser Option im Base64-Format dargestellt. Beim Parameter `AUTO` ist dies die Standardeinstellung, bei `EXPLICIT` und `RAW` muss die Option explizit gesetzt werden.

► XMLDATA

Hiermit wird ein XML Schema mit der Beschreibung der Datentypen erzeugt und dem Dokument vorangestellt.

Eine Anwendung dieser Abfrageform finden Sie unter anderem im Abschnitt 9.2.4, »SQL-Befehle an die Datenbank senden« ab Seite 842.



5.5.7 Sichten (Views)

Sichten (engl. Views) sind treffend benannt. Die Einrichtung einer Sicht auf dem SQL-Server bietet eine »Sicht« auf die Datenbank. Sie können praktisch jede Form eines SQL-Befehls abbilden und damit den Zugriff auf solche komplexen Befehle, wie sie in diesem Kapitel behandelt wurden. Das allein kann jedoch der Sinn nicht sein, denn die Ersparnis von Schreibarbeit ist nicht Gegenstand der Abfragesprache SQL. Sichten bieten eine Art festen Zusammenhang des Zugriffs auf Tabellen. Sie können davon ausgehen, dass die durch eine Sicht definierte Abfrage immer aktuell ist. Änderungen an den Quelldaten führen notwendigerweise zu Änderungen der Ergebnisse, die beim Aufruf der Sicht produziert werden. Wenn sich also häufiger die Quelldaten ändern, die eigentliche Abfrage jedoch kaum, ist der Einsatz einer Sicht sinnvoll.

Einsatz

Sichten können auch aus Sicherheitsgründen zum Einsatz kommen. Wenn in Programmen Sichten zum Einsatz kommen, können die dahinter liegenden Tabellen vor einem direktem Zugriff geschützt werden. Damit könnten auch sicherheitsrelevante Daten in der Datenbank gehalten werden. Der Zugriff erfolgt über Sichten, die nur öffentlich zugängliche Daten replizieren.

Im SQL Server-Enterprise-Manager können Sie Sichten selbst einfach definieren. Sie werden dann Bestandteil der Datenbank und können von allen Systemen benutzt werden, die vollen Zugriff auf die Datenbank haben. Bei der Nutzung von SQL-Server 2000 im Webpace eines Providers kann ein solcher Zugriff möglicherweise nur über ASP.NET erfolgen. Sie sollten deshalb die entsprechenden Befehle kennen.

Definition

Sichten im Detail

Sichten bilden letztlich nur mehr oder weniger komplexe `SELECT`-Abfragen ab. Vor der Definition einer Sicht sollte Ihre Applikation mit entsprechenden `SELECT`-Befehlen einwandfrei funktionieren.

Das folgende Kommando zeigt eine typische Abfrage, die die Namen der Artikel mit Bestellungen der letzten fünf Tage anzeigt:

```
SELECT DISTINCT a.id, a.name
FROM Bestellungen b JOIN Artikel a
ON b.[a-id] = a.id
AND DATEDIFF(dd, bestelldatum, GETDATE()) <= 5
```

CREATE VIEW Möchten Sie diese Abfrage als Sicht verfügbar machen, definieren Sie zuerst eine Sicht mit dem Befehl `CREATE VIEW`:

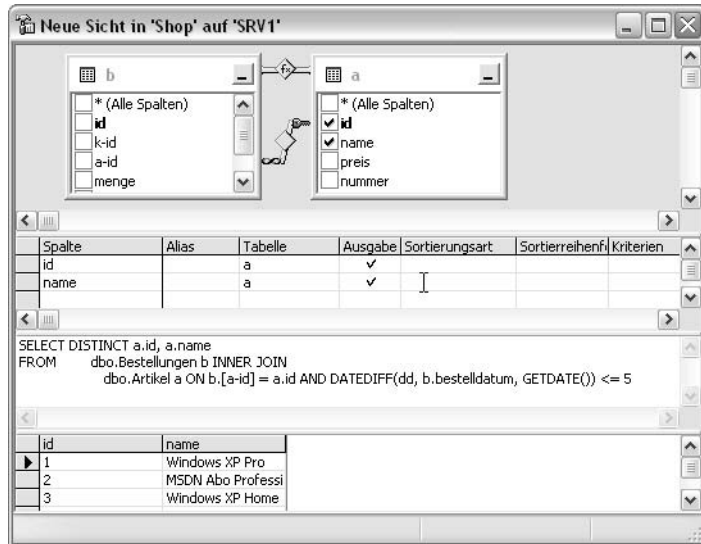


Abbildung 5.19: Definition einer Sicht im Sichteneditor des Enterprise Managers

```
CREATE VIEW LetzteBestellung
AS
SELECT DISTINCT a.id, a.name
FROM Bestellungen b INNER JOIN
    Artikel a
ON b.[a-id] = a.id
AND DATEDIFF(dd, b.bestelldatum, GETDATE()) <= 5
```

Technisch erzeugt eine Sicht eine temporäre, interne Tabelle; im eben gezeigten Beispiel mit dem Namen *actives*. Es liegt nahe, auf diese Tabelle wiederum mit einem normalen `SELECT`-Befehl zuzugreifen:

```
SELECT * FROM LetzteBestellung
```

	id	name
1	1	Windows XP Pro
2	2	MSDN Abo Professional
3	3	Windows XP Home

Diese Abfrage entspricht in der Wirkung exakt der oben beschriebenen Befehlsfolge. Ein Zugriff durch den anonymen Webnutzer erfolgt hier nur auf die durch die Sicht erzeugte Tabelle. Eine praktische Anwendung wäre ein Informationssystem eines Providers, bei dem sich jeder Nutzer des Servers seine persönlichen Daten extrahieren kann, jedoch nicht direkt auf die Tabellen, in denen die Daten aller Nutzer zu finden sind.

Es liegt nahe, dass sich Sichten mit einem `DROP`-Befehl wieder löschen lassen:

DROP VIEW

```
DROP VIEW LetzteBestellung
```

Um eine Sicht neu zu definieren, wenn sich etwas ändert, verwenden Sie den Befehl `ALTER VIEW`. Die Anwendung entspricht exakt `CREATE VIEW`; der angegebene Name der Sicht muss bereits existieren.

ALTER VIEW

In den zuvor gezeigten Beispielen sind die originalen Spaltennamen erhalten geblieben. Für die Anwendung einer Sicht kann es sinnvoll sein, der temporären Tabelle eigene Spaltennamen zuzuweisen. Die Definition der Sicht kann dann folgendermaßen erfolgen:

```
CREATE VIEW LetzteBestellung2 (ArtikelNummer, ArtikelName)
AS
SELECT DISTINCT a.id, a.name ←
    FROM Bestellungen b INNER JOIN ←
        Artikel a ←
    ON b.[a-id] = a.id ←
    AND DATEDIFF(dd, b.bestelldatum, GETDATE()) <= 5
```

Der Abfragebefehl für die nachfolgend gezeigte Ergebnisliste lautet ebenfalls folgendermaßen:

```
SELECT * FROM LetzteBestellung2
```

	Artikelnummer	Artikelname
1	1	Windows XP Pro
2	2	MSDN Abo Professional
3	3	Windows XP Home

Die in der Sicht definierten Spaltennamen sind nach außen hin verbindlich. Die tatsächliche Tabellenstruktur der zugrunde liegenden Tabellen kann damit wirkungsvoll verborgen werden.



Sichten stellen technisch berechnete Tabellen dar. SQL-Datenbanken speichern Daten in Tabellen in keiner spezifischen Reihenfolge. Erst bei der Abfrage der Tabelle können Berechnungen oder Sortierungen durchgeführt werden. Das Sortieren einer Sicht ist deshalb auch nicht möglich. Sie können den Operator `ORDER BY` nicht in einer Sicht-Definition angeben. Stattdessen können Sie die Abfrage einer Sicht mit `ORDER BY` kombinieren.

5.5.8 SQL-Programmierung mit Transact-SQL

T-SQL ist leistungsstark genug, kleinere Abläufe in eigenen Programmen zu erledigen. Der Vorteil ist ein deutlicher Performance-zuwachs gegenüber der Aufteilung der Prozesse zwischen externem Programm und Datenbank.

Es gibt darüber hinaus einen noch wesentlicheren Vorteil für den Programmierer. Wenn Sie bestimmte Merkmale, Einschränkungen, Funktionen und Aktionen in der Datenbank hinterlegen, wird diese selbst für die benötigte Integrität und Stabilität der Daten sorgen. Da die Befehle in T-SQL weitaus mächtiger sind als die in VB.NET (oder auch C#) verfügbaren, ist die Programmierung der Datenbank oft einfacher.

Für die praktische Programmierung gibt es zwei Wege. Zum einen können Sie einfach Befehlsserien als Kommando aus ASP.NET-Applikationen heraus übergeben. Dies wird unter dem Stichwort Batchprogrammierung behandelt. Sie können solche Befehlsfolgen aber auch unter einem Namen im SQL Server speichern. Dies bezeichnet man dann als gespeicherte Prozeduren. Die Batchprogrammierung wird nachfolgend behandelt. Information zu Prozeduren finden Sie in Abschnitt 5.5.11, »Gespeicherte Prozeduren und Funktionen« ab Seite 434.

Batchprogrammierung

Es ist in T-SQL möglich, kleine Programme zu schreiben, die sich von außen oder automatisch starten lassen. Damit vereinfacht sich die Programmierung der Skripte und der SQL Server 2000 ist im Regelfall schneller als bei der Auswertung von Daten durch Skript- oder Programmiersprachen. Wenn Sie SQL-Befehle direkt an der Konsole eingeben, können Sie mehrere Befehle untereinander schreiben und dann die Abarbeitung auslösen. Tritt ein Fehler in einem der Befehle auf, stoppt der SQL Server 2000 die Ausführung ganz. Wenn Sie das verhindern möchten, können Sie Batchprogramme mit dem Befehl `GO` trennen. Jeder Teilabschnitt wird abgearbeitet, bis in einem Abschnitt ein Fehler auftritt. Erst dann wird abgebrochen. Die bis dahin abgearbeiteten Befehle bleiben aber gültig. **GO**

```
SELECT name FROM Artikel  
GO  
SELECT lname FROM Artikel
```

Der Schreibfehler im zweiten Befehl würde eigentlich sofort erkannt werden; und T-SQL würde dann nichts weiter ausführen. Durch den `GO`-Befehl wird zuerst der erste Teil erfolgreich ausgeführt, dann der zweite, der mit einer Fehlermeldung endet. Ein Blick auf die Meldung des Query Analyzers zeigt dies:

```
{5 row(s) affected}  
  
Server: Nachr.-Nr. 207, Schweregrad 16, Status 3, Zeile 1  
Ungültiger Spaltenname 'lname'.
```

Abbildung 5.20: Fehler in einer Stapelverarbeitungsdatei

Variablen

T-SQL kennt zwei Typen von Variablen: globale und lokale. Die globalen werden vom SQL Server selbst erzeugt und können nicht geändert werden. Sie sind in allen Batchprogrammen verfügbar. Auch von außen – über ASP.NET-Programme – können Sie darauf zugreifen. Die lokalen sind so wie alle Variablen selbst zu definieren und können zur Übertragung von Daten benutzt werden.

@@IDENTITY
@@ROWCOUNT

Globale Variablen sind sehr wichtig, denn Sie können damit die am Anfang schon angesprochenen Verknüpfungen steuern. Praktisch wird im folgenden Beispiel ein neuer Datensatz in die Tabelle *items* geschrieben und der Wert des *IDENTITY*-Feldes *item_id* anschließend zurückgegeben:

```
INSERT INTO Artikel (name, preis, nummer) ↵
VALUES ('Werbegag', 9.99, '3456');
SELECT @@IDENTITY AS itemid
```

	itemid
1	6

@@IDENTITY enthält also immer den Schlüsselwert des letzten Befehls. Wenn Sie nun die Artikel in der Tabelle *Artikel* suchen, bringt der Befehl **SELECT** alles zusammen. Damit die frisch erzeugte ID nicht verloren geht, wird sie in einer lokalen Variablen gespeichert:

```
INSERT INTO Artikel (name, preis, nummer) ↵
VALUES ('Werbegag2', 9.99, '1234');
DECLARE @ItemId BIGINT;
SET @ItemId = @@IDENTITY;
SELECT * FROM Artikel WHERE id = @ItemId;
```



*Dieses Beispiel funktioniert nur, wenn es im selben Programm wie der vorhergehende **INSERT**-Befehl ausgeführt wird.*

In der Praxis entspricht *itemid* einer Variablen und *id* der Spalte. Enthält die Tabelle kein Feld, das mit dem Parameter *IDENTITY* erstellt wurde, ist der Inhalt der Variable **@@IDENTITY** gleich **NULL**.

Die andere wichtige Variable ist **@@ROWCOUNT**. Sie gibt die Anzahl der behandelten Datensätze der letzten Aktion zurück. Damit können Sie prüfen, wie viele Reihen von der Auswahl betroffen waren.

```
UPDATE Artikel SET preis = preis *1.2 WHERE preis < 100
SELECT @@ROWCOUNT
```

	{Kein Spaltenname}
1	5

Zum Speichern von Daten gibt es lokale Variablen, die nur während des Batchlaufes aktiv sind. Hier eine einfache Anwendung:

**Lokale Variable
mit DECLARE
erklären**

```
DECLARE @zahl INT  
SELECT @zahl = 4+6  
SELECT @zahl
```

Als Ergebnis wird 10 ausgegeben, was sicher nicht überrascht. Zur Deklaration können Sie alle Datentypen, die T-SQL kennt, verwenden. Die Deklaration ist immer erforderlich.

Die Ausgaben einzelner Werte ist mit ADO.NET sehr einfach möglich. Mehr Informationen darüber finden Sie im Abschnitt 9.2.4, »SQL-Befehle an die Datenbank senden« ab Seite 842. Suchen Sie dort nach der Methode `ExecuteScalar`.



Kommentare

Sie können auch in SQL-Programmen Kommentare anbringen. Innerhalb der Zeile beginnt der Kommentar mit zwei Minuszeichen »--« und außerhalb mit `/*`. Ebenso endet der Kommentar mit `*/`. Sie können natürlich auch jede Zeile immer wieder mit den zwei Minus-Zeichen beginnen lassen.

```
SELECT name FROM Artikel -- alle Namen ermitteln  
/* jetzt werden die folgenden  
   Variablen benutzt:  
*/
```

Eine gute, ausführliche Kommentierung gehört auch in der SQL-Programmierung zu einem guten Programmierstil. Beschreiben Sie in Kommentaren immer, was eine Funktion erledigt, nicht wie sie es tut. Denken Sie auch an die Beschreibung der Übergabeparameter. Es ist sinnvoll, für die Kommentierung dieselben Prinzipien zu verwenden, wie Sie sie für Ihre ASP.NET-Programme einsetzen.



5.5.9 Programmsteuerung in T-SQL-Programmen

Komplexere T-SQL-Programme benötigen auch die bedingte Ausführung von Abschnitten. Mehrere Befehle können hier eingesetzt werden.

Einfache Bedingungen mit IF ... ELSE

IF Mit `WHERE` können Sie nur den aktuellen Befehl von einer Bedingung abhängig machen. Wenn ein Teil des Batchprogramms bedingt ausgeführt werden soll, gibt es zwei mögliche Befehle, die Sie bereits aus anderen Programmiersprachen kennen: `IF` und `ELSE`. Im Gegensatz dazu ist die Syntax anders:

```
DECLARE @viele AS INT
IF (SELECT COUNT(*) FROM Artikel) > 3 SELECT @viele=1
SELECT @viele
```

ELSE Wenn Sie statt des Vergleichswerts 3 die Zahl 100 eingeben, wird `NULL` zurückgegeben, wenn die Tabelle weniger als 100 Datensätze enthält. Mit `ELSE` könnte die Konstruktion so aussehen:

```
DECLARE @viele AS int
IF (SELECT COUNT(*) FROM Artikel) > 3
    SELECT @viele=1
ELSE
    SELECT @viele=0
SELECT @viele
```

Das Schlüsselwort `ELSE` ist optional und wird nur eingesetzt, wenn es tatsächlich um eine alternative Verzweigung geht.

BEGIN ... END Bemerkenswert ist, dass es für die Zusammenfassung von Blöcken keine Klammern wie in `C#` oder ein *Then*-Schlüsselwort wie in `VB.NET` gibt. Stattdessen deklarieren Sie Blöcke mit `BEGIN ... END`. Der nach der Bedingung stehende Befehl wird ausgeführt, wenn die Bedingung wahr ist:

```
DECLARE @viele AS INT
DECLARE @text AS varchar(80)
IF (SELECT COUNT(*) FROM Artikel) > 3
BEGIN
    SELECT @viele=1
    SELECT @text = 'Es wurden mehr als 10 Artikel gefunden'
END
SELECT @viele, @text
```

	{Kein Spaltenname}	{Kein Spaltenname}
1	1	Es wurden mehr als 10 Artikel gefunden

Selbstverständlich ist es auch hier möglich, Spaltennamen mit dem Alias-Operator `AS` zu bilden.

Mehrfache Verzweigungen mit CASE ... WHEN ... THEN

Mit dem CASE-Befehl sind mehrfache Verzweigungen möglich. Hinter jedem THEN kann wieder ein Block mit BEGIN ... END stehen. Die folgende Befehlssequenz definiert zwei Variablen, übernimmt den ersten Artikel aus der Tabelle *Artikel* in die Variable *@name* und weist der Variablen *@wert* entsprechend der in CASE gegebenen Bedingungen die Werte 0, 1, 2 oder 3 zu:

**CASE
WHEN
THEN**

```
DECLARE @name AS varchar(100)
DECLARE @wert AS INT
SELECT @name = name FROM Artikel
SELECT @name
SELECT @wert=
    CASE
        WHEN @name LIKE '%XP%' THEN 1
        WHEN @name LIKE '%Home%' THEN 2
        WHEN @name LIKE '%gag%' THEN 3
        ELSE 0
    END
SELECT @wert
```

	{Kein Spaltenname}
1	Werbegag2
	{Kein Spaltenname}
1	3

Der letzte SELECT-Befehl gibt das Ergebnis aus. Beachten Sie das Gleichheitszeichen vor dem CASE. Es gehört zum SELECT-Befehl. Das folgende Beispiel zeigt, wie man mit CASE komfortable Auswahlmöglichkeiten realisiert.

```
DECLARE @name AS varchar(80)
SELECT cast(nummer as char(4)),
    (CASE nummer
        WHEN '1325' THEN 'Dies ist das neue ↵
                        Windows XP Prodeessional'
        WHEN '6668' THEN 'Das beste Abo für ↵
                        Entwickler: MSDN Pro'
        ELSE 'Unbekannter Artikel'
    END) 'Beschreibung', id 'ID', preis 'Nettopreis'
FROM Artikel ORDER BY 'Beschreibung'
```

	(Kein Spaltenname)	Beschreibung	ID	Nettopreis
1	6668	Das beste Abo für Entwickler: MSDN Pro	2	2850.00
2	1325	Dies ist das neue Windows XP Prodeessional	1	199.99
3	1326	Unbekannter Artikel	3	139.99
4	7738	Unbekannter Artikel	4	118.80
5	8837	Unbekannter Artikel	5	499.00
6	3456	Unbekannter Artikel	6	11.99
7	1234	Unbekannter Artikel	7	11.99
8	1234	Unbekannter Artikel	8	11.99
9	1234	Unbekannter Artikel	9	11.99

Damit wird eine Tabelle ausgegeben, die für einige Artikel eine weiterführende Beschreibung enthält. Sie können statt der Ausgabe auf dem Bildschirm ebensogut die neuen Werte in eine andere Tabelle kopieren. Natürlich lassen sich auch die zugewiesenen Werte in Variablen übertragen und von dort aus weiterverarbeiten, ebenso werden in der Praxis auch die Übersetzungen aus einer anderen Tabelle stammen.

RETURN

RETURN Um ein Batchprogramm zu verlassen, gibt es den Befehl `RETURN`. Bauen Sie diesen Befehl in eine Bedingung ein:

```
IF DATENAME(mm,GETDATE())="Feb" RETURN
```

Schleifen mit WHILE, BREAK und CONTINUE

WHILE
BREAK
CONTINUE Beim Umgang mit Datenbankzeigern ist es oft notwendig, die Datensätze zu durchlaufen. Dazu dient der Schleifenbefehl `WHILE`. Die Schleife kann mit `BREAK` verlassen werden, wozu `IF` eingesetzt wird. Um die Prüfung der Schleifenbedingung zu erzwingen, kann `CONTINUE` verwendet werden.

5.5.10 Transaktionen

Viele Prozesse, die im Internet mit Datenbanken bearbeitet werden, sind kritische Vorgänge mit Kreditkartentransaktionen, Zugriffen auf Gateways der Geldinstitute oder Abgleich kostbarer Kundendaten. Auch Webserver bestehen nur aus normalen Computern und unterliegen Störungen und technischen Problemen. Es besteht so immer die Möglichkeit, dass Sie eine Tabelle öffnen, etwas schreiben und dann die korrespondierende Tabelle öffnen und auch etwas schreiben. Genau da stürzt der Computer ab. Jetzt

stimmen die Tabellen nicht mehr überein, das Programm läuft nicht mehr. Oder das Zahlungsgateway weigert sich neue Prozesse anzunehmen, weil der alte Prozess nicht beendet wurde.

BEGIN, COMMIT, ROLLBACK und TRANSACTION

Um sich vor solchen Problemen zu schützen, können Sie Batchprogramme mit dem Schlüsselwort `TRANSACTION` in Module zusammenfassen. Der Trick: T-SQL wird das Modul als Einheit betrachten und – wenn die Abarbeitung unterbrochen wurde – alles rückgängig machen, was schon begonnen wurde. Alle Daten werden erst dann gültig, wenn das Ende des Moduls erreicht wurde. Das Prinzip: Entweder alles oder nichts. Das sieht folgendermaßen aus:

**BEGIN
COMMIT
ROLLBACK
TRANSACTION**

```
BEGIN TRANSACTION
INSERT Artikel (name, nummer, preis)
    VALUES ('Sonderanfertigung', '9999', 19.99)
INSERT Bestellungen ([k-id], [a-id], menge, summe)
    VALUES (4, @@IDENTITY, 1, 19.99)
COMMIT TRANSACTION
```

Nachdem die Kreditkartennummer gespeichert wurde, stürzt der Webserver ab. Da das zweite `INSERT`-Kommando noch nicht ausgeführt werden konnte, wird auch die erste »Transaktion« keinen Effekt haben. Sie können dieses Zurückholen (engl. roll back) oder Ungültigmachen eines Befehls auch von Hand auslösen:

```
ROLLBACK TRANSACTION
```

Anders sieht die Anwendung aus, wenn auch die Fehler beachtet werden sollen, die bei laufendem Betrieb auftreten können. Hierzu werten Sie den Fehlercode aus:

```
BEGIN TRANSACTION
INSERT Artikel (name, nummer, preis)
    VALUES ('Sonderanfertigung', '9999', 19.99)
INSERT Bestellungen ([k-id], [a-id], menge, summe)
    VALUES (2, @@IDENTITY, 1, 19.99)
IF @@ERROR = 0
    COMMIT TRANSACTION
ELSE
    ROLLBACK TRANSACTION
SELECT @@ERROR AS Fehlermeldung
```

Die im Beispiel verwendete Systemvariable @@ERROR enthält den Fehlercode, der Code »0« bezeichnet »keine Fehler«. Die Auswertung des Fehlers ist sinnvoll, um dem Nutzer einen kontrollierten Hinweis zu geben. In ADO.NET stehen entsprechende Fehlerobjekte zur Verfügung, die detaillierte T-SQL-Fehlermeldungen beinhalten.

5.5.11 Gespeicherte Prozeduren und Funktionen

Prozeduren sind unter einem bestimmten Namen gespeicherte Befehlsfolgen, die sich gezielt und immer wieder starten lassen. Im Gegensatz zu Batchprogrammen werden Prozeduren kompiliert und dadurch insgesamt schneller ausgeführt. Sie sind dann natürlich auch unter einem bestimmten Namen jederzeit wieder verwendbar. Batchprogramme müssen Sie dagegen in Ihrem ASP.NET-Code ablegen.

Funktionen werden vergleichbar verwendet, haben jedoch immer einen definierten Rückgabewert.

Einführung in die Technik der gespeicherten Prozeduren

T-SQL kennt auch Prozeduren. Diese Prozeduren sind unter einem eigenen Namen in der Datenbank gespeichert und stehen so immer wieder zur Verfügung. Die Verwendung ist durchaus sinnvoll. Einige der mögliche Anwendungen finden Sie nachfolgend:

- ▶ Wenn Sie in Ihren Programmen wiederholt umfangreiche T-SQL-Strukturen nutzen, kann es sinnvoll sein, diese als Prozedur zu speichern.
- ▶ Wenn Sie intensiv Stapeldateien nutzen, denken Sie daran, dass der Server jedes Mal den Stapel durchsucht, übersetzt und ausführt. Prozeduren sind schneller. Die Ausführung von zehn Befehlen in einer Prozedur von einer Skriptsprache aus zu starten ist schneller, als zehn Befehle einzeln von außen auszuführen.
- ▶ Prozeduren können Parameter entgegennehmen und zurückgeben.
- ▶ Haben Sie den Datenbankserver in einer anderen Maschine, reduzieren Sie den Netzwerkverkehr zwischen den Computern, wenn Sie statt einzelner Befehle Prozeduren nutzen.

- ▶ Sie können die Rechte innerhalb der Tabellen von Prozeduren aus so beeinflussen wie direkt im SQL Enterprise-Manager. Das erhöht die Sicherheit der gesamten Datenbankanwendung.
- ▶ Prozeduren können einander aufrufen. So sind komplexe Applikationen möglich. Sie können kleinere Prozeduren immer wieder verwenden und so schnell zu umfangreichen Datenbankprogrammen gelangen.

Prozeduren sind auch dann sinnvoll, wenn komplexere Berechnungen mit großen Datenmengen ausgeführt werden sollen, und dafür der Zugriff auf ADO.NET nicht performant genug ist. ADO.NET speichert Daten verbindungslos in eigenen Datensatzobjekten. Das ist meist effizient, führt aber zu Problemen, wenn für Manipulationen an den Daten tatsächlich sehr große Teile der Datenbank benötigt werden. In solchen Fällen ist es besser, die Arbeit in der Datenbank erledigen zu lassen. Dafür sind gespeicherte Prozeduren der einzige Weg.

Gespeicherte Prozeduren erzeugen, ändern und verwenden

Sie können jeden einfachen T-SQL-Befehl als Prozedur ablegen. Die folgende Prozedur gibt alle Felder der Tabelle *Adressen* zurück:

**CREATE
PROCEDURE**

```
CREATE PROCEDURE AlleNamen  
AS  
SELECT * FROM Adressen
```

Um die Prozedur gleich auszuprobieren, nutzen Sie den Befehl EXECUTE:

```
EXECUTE AlleNamen
```

Der Name der Prozedur muss eindeutig sein. Sie können natürlich mehrere Befehle in der Prozedur speichern. Sind Sie fertig, speichern Sie die Prozedur. Änderungen sind mit ALTER PROCEDURE möglich. Wollen Sie unbedingt etwas ändern, löschen Sie die Prozedur, und legen Sie sie neu an:

DROP PROCEDURE

```
DROP PROCEDURE EinigeNamen
```

Prozeduren können Parameter übernehmen. Die übergebenen Werte werden mit den bekannten Datentypen deklariert:

```

CREATE PROCEDURE EinigeNamen (@key VARCHAR(20))
AS
IF EXISTS
    (SELECT name FROM Adressen
     WHERE id LIKE '%' + @key + '%')
    PRINT "Gefunden"
ELSE
    PRINT "Nicht gefunden"

```

Sie können die so definierte Prozedur nun mit einem Parameter aufrufen:

```
EXECUTE EinigeNamen 3
```

Sie können maximal 255 Parameter an jede Prozedur übergeben.

ALTER PROCEDURE Um eine Prozedur ändern zu können, verwenden Sie den Befehl ALTER PROCEDURE. Die Syntax entspricht exakt CREATE; geben Sie einfach eine neue Definition an.

EXECUTE Beim Aufruf mit EXECUTE werden die Parameter einfach hinter den Namen geschrieben:

```
EXECUTE EinigeName 3
```

Die Anwendung ist meist optional, bei bestimmten Situationen aber zu empfehlen (Lesbarkeit) oder erzwungen (Schlüsselwort erwartet). EXECUTE kann aber auch Zeichenfolgen als T-SQL interpretieren und ausführen. Sie können damit die Bezeichner (Tabellen, Spalten) dynamisch zusammenstellen:

```

DECLARE @spalte AS VARCHAR(20)
SET @spalte = 'name'
EXECUTE ('SELECT ' + @spalte + ' FROM Adressen')

```

Parameter flexibel definieren

Parameter können von Prozeduren auch wieder zurückgegeben werden. In einigen SQL-Umgebungen werden sie dann Funktionen genannt. Die Deklaration derartiger Prozeduren ist wichtig, denn Sie können die Rückgabewerte in ADO.NET auswerten.

```

CREATE PROCEDURE EinigeNamen2
    (@key VARCHAR(10),
     @message VARCHAR(50) OUTPUT)
AS
IF EXISTS
    (SELECT name FROM Adressen

```



```

        WHERE id LIKE '%'+@key+'%')
    SELECT @message = 'Gefunden'
ELSE
    SELECT @message = 'Nicht gefunden'
RETURN

```

Der Aufruf ist ähnlich der Prozedur ohne Rückgabewert, nur das Schlüsselwort **OUTPUT** muss mit angegeben werden.

```

DECLARE @ergebnis VARCHAR(20)
EXECUTE EinigeNamen2 3, @ergebnis OUTPUT
SELECT @ergebnis

```

	(Kein Spaltenname)
1	Gefunden

Der Befehl *PRINT* ist nur sinnvoll, wenn der SQL Server eine Ausgabekonsolle hat, beispielsweise wenn Sie den Query Analyzer verwenden. In anderen Umgebungen gehen die mit *PRINT* erzeugten Ausgaben verloren. Sie können die Ergebnisse auf diesem Wege nicht nach ASP.NET transportieren.



Auch Prozeduren verstehen das Schlüsselwort **RETURN**. Sie können es genauso verwenden wie in den Batchprogrammen. Mit Erreichen des Wortes **RETURN** wird die Prozedur sofort beendet. Hinter **RETURN** können Sie Werte angeben, die zurückgegeben werden. Die Angabe muss vom Datentyp **INT** sein.

RETURN

```

CREATE PROCEDURE EinigeAutoren3 (@key VARCHAR(10))
AS
IF EXISTS
(SELECT name FROM Adressen WHERE id LIKE '%'+@key+'%')
    RETURN 0
ELSE
    RETURN 1

```

Solche Statuswerte sind manchmal einfacher und leichter zu behandeln als irgendwelche Wortmeldungen. Beachten Sie aber, dass T-SQL selbst Statuswerte kennt. Benutzen Sie eigene Werte nur größer als 0 oder kleiner als -99. Die Werte zwischen -99 und 0 sind von T-SQL belegt.

Funktionen

CREATE FUNCTION Funktionen werden mit `CREATE FUNCTION` erzeugt. Der enthaltene Code entspricht den Prozeduren, ebenso die Verwendung. Der Rückgabewert wird mit der Klausel `RETURNS` festgelegt. Im Gegensatz zu Prozeduren kann mit `RETURNS TABLE` auch die Rückgabe einer Tabelle erfolgen.

5.5.12 Trigger

Als kleine Automaten in T-SQL könnte man Trigger bezeichnen. Mit diesen Funktionen automatisieren Sie Prozesse auf einfachste Art und Weise. Vor allem für globale Einschränkungen sind Trigger prädestiniert. Trigger arbeiten ähnlich wie die Ereignisse in .NET, sind aber vergleichsweise primitiv.

Einführung in Trigger

TRIGGER Unter dem Stichwort Trigger (dt. Auslöser) werden Prozeduren verstanden, deren Start nicht von der Eingabe oder der Ausführung eines Befehles abhängt, sondern von Ereignissen. Trigger werden also nicht von einem Skript aus gestartet, sondern mit bestimmten Zuständen und Bedingungen verbunden und starten dann bei Bedarf direkt im SQL-Server.

Auslösende Merkmale können alle Arten von Änderungen an der Datenbank sein – also die Anwendung der Befehle `UPDATE`, `INSERT` und `DELETE`. Auslösen können Trigger eine gespeicherte Prozedur oder ein Batchprogramm. Jeder Trigger besteht aus drei elementaren Angaben:

1. Dem Namen des Triggers
2. Der Definition des Auslösers
3. Der bei Auslösung auszuführenden Befehlsfolge

Folgende Merkmale zeichnen Trigger aus:

- ▶ Trigger werden einer Tabelle zugeordnet.
- ▶ Löschen Sie die Tabelle mit `DROP`, werden auch alle dieser Tabelle zugeordneten Trigger gelöscht.
- ▶ Jede Aktion kann mehr als nur einen Trigger haben.
- ▶ Normalerweise haben Sie drei Trigger: `UPDATE`, `DELETE`, `INSERT`.

Wenn Sie Trigger so programmieren, dass ein Trigger durch die Ausführung seiner Befehlsfolge in einer anderen Tabelle einen weiteren Trigger startet, spricht man von verschachtelten Triggern. Sie können bis zu 32 Stufen von Verschachtelungen mit ausführen. Jede weitere Stufe wird ignoriert.

CREATE TRIGGER ... ON ... FOR ... AS

Die Erzeugung eines neuen Triggers ist sehr einfach:

CREATE TRIGGER

```
CREATE TRIGGER new_webuser ON Shop FOR INSERT
AS
EXECUTE master..xp_sendmail "admin", "Neuer Nutzer angemeldet"
```

Hinter dem Schlüsselwort `TRIGGER` steht der Name des Triggers, hinter `ON` die Tabelle, auf die sich der Trigger bezieht. Trigger arbeiten immer tabellenorientiert. `FOR INSERT` bedeutet, dass der Befehl `INSERT`, auf die Tabelle *admins* angewendet, den Trigger auslöst. Hinter `AS` beginnt dann der Code des Triggers. In dem Beispiel wird eine E-Mail versendet, wenn ein SMTP-Server erreichbar ist.

DROP TRIGGER und ALTER TRIGGER

Löschen können Sie Trigger mit dem `DROP`-Befehl:

DROP TRIGGER

```
DROP TRIGGER name
```

T-SQL kennt auch den Befehl `ALTER TRIGGER`, mit dem Sie eine Definition wiederholen können, ohne den `DROP TRIGGER`-Befehl zuvor einsetzen zu müssen.

Trigger-Praxis

Ein praktisches Beispiel soll zeigen, wie Sie Trigger in realen Datenbankprojekten einsetzen können und damit die Programmierung der umgebenden Skripte vereinfachen. Der folgende Trigger berechnet die Gesamtmenge der Artikel bei einer neuen Bestellung:

```
CREATE TRIGGER SumMenge ON Bestellungen
FOR INSERT
AS
UPDATE Bestellungen
```

```
SET menge = menge +
  (SELECT menge FROM Bestellungen
   WHERE id = @@IDENTITY)
WHERE id = (SELECT [a-id]
  FROM Bestellungen
  WHERE id = @@IDENTITY)
DELETE FROM Bestellungen WHERE id = @@IDENTITY
```

Der Trigger startet nur einmal, auch wenn mit einem INSERT-Befehl mehrere Datensätze betroffen sind. Der Trigger startet aufgrund des Auftretens des INSERT-Befehls, nicht aufgrund dessen Effektes.

Triggertabellen
nutzen

Um feststellen zu können, welche Aktion den Trigger wirklich ausgelöst hat, reichen die bisherigen Informationen nicht aus. DELETE- oder UPDATE-Befehle werden bei großen Programmen hundertfach benutzt, und immer wieder wird der Trigger ausgelöst. Der folgende Trigger speichert beispielsweise jeden gelöschten Ansprechpartner in einer Sicherheitstabelle:

```
CREATE TRIGGER name_weg ON Adressen
FOR DELETE
AS
INSERT logdata (name) SELECT name FROM deleted
```

Der Trick ist die Verwendung der Tabelle *deleted*. Diese wird automatisch mit jedem DELETE angelegt, wenn ein Trigger definiert ist. Ebenso gibt es eine Tabelle *inserted*, die nur dann vorhanden ist, wenn der Befehl INSERT arbeitet und ein assoziierter Trigger existiert. Wann immer Sie die gesamte Tabelle mit DELETE partner löschen, wird die Spalte *name* komplett nach Tabelle *logdata* kopiert. Grundsätzlich können Sie damit leicht Protokolldateien für Ihren SQL Server erzeugen. Den Zustand der beiden Tabellen finden Sie in der folgenden Liste:

Tabelle	INSERT	DELETE	UPDATE
inserted	Eingefügte Reihen	Leer	Reihen vor UPDATE
deleted	Leer	Gelöschte Reihen	Reihen nach UPDATE

Tabelle 5.13: Auslöser für Trigger

Tricks mit Triggern

Raffiniert ist die Kombination aus Trigger und Transaktion. Der folgende Trigger verhindert Änderungen an der Datenbank am Wochenende:

```
CREATE TRIGGER weekend_prevent ON Adressen
FOR INSERT,DELETE,UPDATE
AS
IF DATENAME(dw,GETDATE())="Sunday") OR
   DATENAME(dw,GETDATE())="Saturday")
ROLLBACK TRANSACTION
```

Ohne eine Zeile Code in Ihrem ASP.NET-Programm wird damit ein sehr globaler Effekt erzielt. Das ist auch der Sinn der Trigger. Denken Sie bei der Planung einer Applikation dann an Trigger, wenn Sie grundsätzliche Prozesse automatisieren möchten, die sich oft wiederholen würden.

5.6 Praktische Einführung in XML

XML hat sich inzwischen fest in der Softwarewelt etabliert. Das hat gute Gründe und ist nicht nur einem aktuellen Hype zuzuschreiben. Denn die Einsatzmöglichkeiten sind recht umfassend, auch wenn XML keinesfalls ein Allheilmittel für alle Arten von Daten darstellt. Die umfassende und direkte Unterstützung in .NET erlaubt jedoch die Anwendung auch dann, wenn es sich nur um eine mögliche Form des Einsatzes handelt.

5.6.1 Anwendung und Verarbeitung

Wenn Sie ASP.NET-Applikationen entwickeln und nicht direkt mit XML in Berührung kommen, mag die Nutzung an den Haaren herbeigezogen erscheinen. Dem ist jedoch nicht so, denn es gibt viele Verwendungen:

- ▶ Einsatz zum Speichern von Konfigurationen
- ▶ Einsatz als statische Datenquelle
- ▶ Einsatz als »Nur-Lese«-Datenquelle

Trotz dem Fokus auf »Lesen« kann mit .NET XML auch erzeugt werden. Dies geht aber eher in den Bereich des Umgangs mit Daten allgemein hinein und soll später noch genauer betrachtet werden.

Zum Speichern von Konfigurationen wird XML bereits eingesetzt; die Dateien *web.config* und *machine.config* basieren bereits auf derartigen Tags. Wenn Sie Pfade zu einer Datenbankdatei, E-Mail-Adressen oder Versionsnummer für Ihr Projekt speichern, bietet

**Speichern von
Konfigurationen**

sich XML dafür an. Ändern sich die Daten, laden Sie die aktualisierte Konfigurationsdatei per FTP oder WebDAV auf den Server und die Applikationen berücksichtigt das entsprechend.

Datenquelle für Steuerelemente

Geht man etwas weiter in den Überlegungen, können auch HTML-Fragmente konfiguriert werden. So sind relativ häufig Listfelder (`<select>`-Tags) im Einsatz. Diese lassen sich gut per XML füllen. Da XML als Datenquelle dienen kann – analog zu einem relationalen Datenbankmanagementsystem – ist der Einsatz mit Webserver-Steuerelementen sehr einfach.

Denken Sie an einen täglichen Nachrichtendienst, der aktuelle Informationen auf der Website anzeigt. Auch hier werden die Daten nur unregelmäßig, und durch externe Quellen gesteuert verändert. Die Applikation selbst greift auf den Pool an Informationen nur lesend zu. Bedenkt man weiter, dass sich Quelle und Darstellformat häufig ändern können, ist XML ideal geeignet. Wenn Ihre Site komplexer wird, könnten Sie XML-verwandte Technologien einsetzen, um mit den Daten flexibel zu arbeiten. Aus einer größeren Anzahl Daten käme XPath in Betracht, um die Auswahl zu ermöglichen. Für die Darstellung in verschiedenen Medien ist XSLT die geeignete Technologie.

Datenaustausch

Nicht zuletzt wird XML natürlich bereits in seiner eigentlichen Domäne eingesetzt, beim Austausch von Daten. Diese und weitere Möglichkeiten werden in den folgenden Abschnitten besprochen. Informationen zu XPath finden Sie in Abschnitt 5.8, »Daten suchen mit XPath« ab Seite 470, zu XSLT werden in Abschnitt 5.9, »Daten transformieren mit XSLT« ab Seite 478 weitere Ausführungen folgen.

Verarbeitungsprinzipien

Aus der Aufgabenstellung ergeben sich verschiedene Strategien, wie die Daten eines XML-Dokumentes verarbeitet werden können. Vereinfacht kann das auf zwei Varianten reduziert werden:

- ▶ Sequenzielles Lesen eines Dokuments
- ▶ Aufbau eines Objektmodells zum wahlfreien Lesen

Push: SAX

Das sequenzielle Lesen wird von zwei Mechanismen verwendet. Einer wird als SAX(Simple API for XML) bezeichnet. Dies ist eine Schnittstelle zu XML-Dokumenten, die Programme implementieren können. SAX ist ein so genanntes Push-Modell. Der Parser

liest dabei Knoten sequenziell ein und sendet ein entsprechendes Ereignis an das verarbeitende Programm immer dann, wenn ein Knoten auftritt. Das Programm kann dann auf die Daten angemessen reagieren. SAX ist nicht direkt in .NET implementiert, sondern wird im Rahmen des MSXML-Moduls bereit gestellt. Man kann aber den *XmlReader* einsetzen, um eine SAX-Schnittstelle in .NET zu programmieren, wenn die Integration von MSXML nicht ratsam ist oder nicht praktikabel erscheint.

Alternativ kann ein Pull-Modell verwendet werden. Dabei fordert das Programm den nächsten Knoten explizit vom Parser an. Die *XmlReader*-Implementierung in .NET basiert auf diesem Prinzip. Wie SAX ist auch dieses Modell lediglich zum Vorwärtslesen in der Lage. Vorteilhaft ist die hohe Geschwindigkeit und der gute Umgang mit sehr großen Datenmengen, denn es ist nicht notwendig, mehr als einen Knoten im Speicher zu halten. Wenn nur kleine Datenmengen aus großen Dokumenten benötigt werden oder der gesamte Inhalt gelesen werden muss, ist *XmlReader* die beste Wahl. Die Implementierung des allgemeinen Modells in .NET ist um einige Methoden erweitert worden, mit denen sich der Einsatz von DOM oft vermeiden lässt.

Pull: *XmlReader*

Völlig anders arbeitet das Objektmodell, als DOM (*Document Object Model*) bezeichnet. Dabei wird das gesamte Dokument gelesen und die Abhängigkeiten und Strukturen als Hierarchie dargestellt. Nun kann wahlfrei auf jeden Knoten zugegriffen werden. Dazu wird als Abfragesprache XPath eingesetzt. Den Weg zu einem Knoten kann man sich als Pfad vorstellen, daher der Name. Mit der flexibleren Verarbeitung des Dokuments verbunden ist der Nachteil hohen Speicherverbrauchs. DOM eignet sich also nur für relativ kleine Dokumente. Im Framework dienen die Klassen *XmlDocument* und *XmlDataDocument* dem Zugriff auf XML-Dokumente mit Hilfe ihres Objektmodells.

**DOM:
*XmlDocument***

5.6.2 XML-Grundlagen

Die Theorie hinter XML ist weder ungewöhnlich komplex noch unverständlich. Dennoch sind ein paar Grundlagen für den praktischen Einsatz zu beachten, wenn Sie nicht mit Leistungseinbrüchen oder unhandlichem Code konfrontiert werden möchten.

Sie sollten sich über den grundsätzlichen Aufbau eines XML-Dokuments im Klaren sein. Anfänger sollten gegebenenfalls weitere

Literatur speziell zu diesem Thema konsultieren. An dieser Stelle werden nur die notwendigsten Eigenschaften aus Aussagen vermittelt.

Aufbau und Merkmale eines XML-Dokuments

XML-Dokumente gehorchen in ihrem Aufbau festen Regeln. Erst durch diese Regeln kann eine automatisierte Verarbeitung in derart umfassender Form stattfinden, wie sie erforderlich ist. Auf der anderen Seite sollte der Einsatz so universell wie möglich sein, wie der Name »Extensible Markup Language« suggeriert.

Wohlgeformtheit

Die Regeln erlauben die Prüfung der Wohlgeformtheit von Dokumenten durch das verarbeitende Programm – Parser genannt – auch ohne Kenntnis der Grammatik der von Ihnen konkret eingesetzten Sprache. Der Begriff »Wohlgeformtheit«, im englischen »well formed« genannt, ist ein Basismerkmal von Markup-Sprachen. Als wohlgeformt gilt ein Dokument, wenn folgende Merkmale zutreffen:

► *Alle Tags sind syntaktisch korrekt*

Dazu gehört, dass Anfangs- und Endtag übereinstimmen, wobei auch Groß- und Kleinschreibung zu beachten ist (`<tag></Tag>` wäre unzulässig). Zu jedem öffnenden Tag muss ein schließendes existieren (`<tag></tag>`). Alternativ kann ein Tag auch direkt geschlossen werden (`<tag/>`).

► *Alle Attribute sind syntaktisch korrekt*

Parameter der Attribute müssen immer in Anführungszeichen stehen (`<tag attr="parm">`). Außerdem sind verkürzte Attribute – also solche ohne Parameter – nicht erlaubt. Das HTML-Tag `<hr noshade>` würde in XHTML – der Entsprechung von HTML in XML – folgendermaßen aussehen:

```
<hr noshade="noshade" />.
```

► *Korrekte Verschachtelung erforderlich*

Tags müssen korrekt verschachtelt sein. Die folgende Sequenz ist falsch:

```
<b><i></b></i>>
```

Stattdessen müsste es folgendermaßen geschrieben werden:

```
<b><i></i></b>>
```


Bei der Verarbeitung der Daten gibt es verschiedene Strategien. Eine davon bildet das Dokument als Objektmodell in einer Hierarchie ab. Falsch verschachtelte Tags lassen die Darstellung als Hierarchie nicht zu.

In einem zweiten Verarbeitungsschritt muss dann natürlich auch die Grammatik geprüft werden. Dabei geht es um erlaubte Tags, deren Beziehungen zueinander und die möglicherweise einsetzbaren Attribute. Dieser Schritt wird als Validierung bezeichnet. Ein Dokument muss vor der Verarbeitung als »valid« oder gültig erkennbar sein.

Validierung

Nun muss zur Validierung natürlich eine Vorschrift existieren, aus der der Parser die Gültigkeit ableiten kann. Dafür gibt es mehrere Verfahren.

Hinweise zum Entwurf einer eigenen Grammatik

Für kleinere Applikationen mag es durchaus praktikabel erscheinen, eine eigene Grammatik zu entwerfen. Vor allem im Hinblick auf die spätere Transformation können Sie sich viel Arbeit ersparen, wenn der Entwurf geradlinig und elegant ist.

Zwei Philosophien zum Einsatz XML

Es gibt zwei widerstreitende Lager, was den typischen Einsatz von XML betrifft. Auf der einen Seite wird eine Hauptanwendung in der Dokumentenverarbeitung gesehen, wo textuelle Abschnitte durch Markierung computerlesbar werden. Im engeren Sinne entspricht dies dem Einbau von Formatanweisungen in Text und damit der Variante XHTML. Auch Bücher, wie das vorliegende, entstehen teilweise auf diesem Wege. Hier finden spezielle Grammatiken wie DocBook Verwendung. Dies ist sinnvoll und für eine wenigstens ansatzweise Automatisierung notwendig. Die Vertreter dieser Philosophie gehen aber auch davon aus, dass derart erstelltes XML primär von Menschen gelesen wird, sei es unbehandelt oder schön formatiert. Solche XML-Dokumente sind oft sehr groß und verlangen spezielle Software zur Verarbeitung.

Auf der anderen Seite steht ein Lager, dessen Anhänger einen datenzentrierten Ansatz als Haupteinsatzgebiet von XML sehen. Hier wird davon ausgegangen, dass künftig die überragende Mehrheit der XML-Dokumente automatisch erzeugt und auch automatisch gelesen wird; der Mensch also nur selten reines XML zu sehen bekommt. Daten in XML sind nicht immer so groß wie reine Texte und die Software zur Verarbeitung ist Bestandteil der meisten modernen Parser.

Es hat den Anschein, dass der datenzentrierte Ansatz von größerer Bedeutung ist. Es gibt mehr Software, eine direktere und zügigere (aus Sicht der Verfügbarkeit von Anwendungsprogrammen) Unterstützung und ein breiteres Einsatzspektrum. Die mit .NET gelieferten Klassen zur Verarbeitung von XML zeigen ebenfalls einen eindeutig datenorientierten Ansatz. Vermutlich werden sich die XML-Versionen zukünftig spezialisieren, um beiden Einsatzspektren optimal genügen zu können.

Struktur entwerfen

XML-Dokumente beschreiben Struktur und Inhalt von Daten. Dabei wird der Inhalt durch die Namen der Tags beschrieben, die Struktur dagegen durch deren Anordnung. Da die Strukturbeschreibung zu den signifikanten Merkmalen gehört, sollten sie ihr zuerst Ihre Aufmerksamkeit schenken. Generell sollte jede Datenebene eine eigene Struktur enthalten. Betrachten Sie beispielsweise ein Telefonbuch. Nahe liegend wäre folgender Entwurf für einen Eintrag:

```
<eintrag>
  Lieschen Müller
  <telefon>030--12345678</telefon>
  <handy>0177--9876543</handy>
</eintrag>
```

Diese Variante kann man bestenfalls als unglücklich bezeichnen. Warum? Als Erstes ist das Vorhandensein von zwei verschiedenen Knotentypen als Unterknoten nicht leicht zu verarbeiten. Dies kommt, wie im Exkurs »Zwei Philosophien zum Einsatz XML« erklärt, bei der Erstellung von Textdokumenten häufig vor. Beim datenorientierten Ansatz ist diese Methode dagegen störend. Unterhalb von `<eintrag>` finden Sie sowohl den Textknoten »Lieschen Müller« als auch zwei normale Knoten `<telefon>` und `<handy>`. Die

Auswahl mit XPath bereitet hier Schwierigkeiten, weil Textknoten immer Endpunkte eines Pfades sein sollten. Konsequenter wäre die folgende Variante:

```
<eintrag>
  <name>Lieschen Müller</name>
  <telefon>030--12345678</telefon>
  <handy>0177--9876543</handy>
</eintrag>
```

Je finer Sie die Struktur beschreiben, umso einfacher wird die Auswahl von Teilen bei der Transformation und Darstellung. »Übertriebene« Feinheit kann jedoch zu unübersichtlichen und schwer handhabbaren Dokumenten führen.



Nun ist auch dies noch nicht perfekt, denn eine Abfrage bestimmter Teilmerkmale bereitet immer noch Probleme. Die Auswahl des Nachnamens ist kaum zuverlässig möglich und eine Liste aller Telefontypen, unabhängig davon, ob es sich um Festnetz- oder Mobiltelefone handelt, ist nur schwer möglich. Durchaus denkbar ist, dass sich die Anzahl an Typen später erhöht, was massive Änderungen in Transformationsanweisungen zur Folge hätte. Eine bessere Basis für den nächsten Schritt wäre folgende Version:

Inhalt beschreiben

```
<eintrag>
  <name>
    <vorname>Lieschen</vorname>
    <nachname>Müller</nachname>
  </name>
  <telefon>030--12345678</telefon>
  <telefon>0177--9876543</telefon>
</eintrag>
```

Eine exakte Beschreibung des Inhalts vermeidet Fehler und verbessert die automatisierte Verarbeitung. Der Computer »lernt« anhand der Auszeichnung der Inhalte mit Tags deren Bedeutung kennen. Je finer die Inhaltsbeschreibung, umso häufiger können Informationen direkt abgerufen werden. Übertriebene Markierung kann zu komplizierteren Transformationsprogrammen führen.



Für die Unterscheidung von Merkmalen vergleichbarer Inhalte werden Attribute verwendet. Der Inhaltstyp »Telefon« kann nach der Zugehörigkeit zu einem Netz eingeteilt werden. Hier ist der Einsatz von Attributen angeraten, was zu folgendem Code führt:

Attribute verwenden

```

<eintrag>
  <name>
    <vorname>Lieschen</vorname>
    <nachname>Müller</nachname>
  </name>
  <telefon type="telekom">030--12345678</telefon>
  <telefon type="eplus">0177--9876543</telefon>
</eintrag>

```

Jetzt kann jedes Element direkt mittels XPath – später folgt mehr dazu – aus dem Datenstrom extrahiert werden. Durch die Attribute lassen sich Daten auch gut filtern. Attribute beschreiben Zustände oder definieren Modifikationen. Sie sollten eindeutig, einmalig und in geringer Zahl vorhanden sein. Folgende Definition ist nicht zu empfehlen:

```
<tag a1="n1" a2="n2" a3="n3" .../>
```

Es ist absehbar, dass es weitere Attribute »a4« usw. geben wird. Vermutlich sind dann auch Kombinationen daraus zulässig. Das ist besser mit Tags zu erledigen:

```

<tag>
  <a1>n1</a1>
  ...

```

Dies erscheint zwar umständlicher, ist aber einfacher zu verarbeiten. Die konkrete Größe eines XML-Dokuments spielt meist keine Rolle. Ob es am Ende 50 KByte oder 80 KByte sind, ist uninteressant. Viel schwerer wiegt der Aufwand, der zur Transformation betrieben werden muss.



Attribute sind kein Ersatz für Struktur- und Inhaltsbeschreibung, sondern dienen der Modifikation einer bereits aufgelösten Struktur. Sie beschreiben Zustände des Inhalts, nicht der Struktur. Attribute verhalten sich bei der Auswahl mit XPath wie Knoten. Wenn eine größere Anzahl Attribute pro Tag notwendig erscheint, sind jedoch reguläre Subknoten – mit eigenen Tags – besser lesbar und meist auch einfacher zu verarbeiten.

Das, was beim Entwurf einer XML-Definition herauskommt, wird auch als »Grammatik« bezeichnet. Was auf dem Papier lesbar ist, muss nun auch für den Computer verständlich definiert werden. Der folgende Abschnitt beschreibt, wie das erfolgt.

Die Grammatik eines XML-Dokuments definieren

Es gibt drei Wege, die Grammatik eines XML-Dokuments zu definieren. Alle Möglichkeiten werden in .NET unterstützt (dies ist keine .NET-spezifische Angelegenheit, XML ist ein offener Standard):

- ▶ Document Type Definition (DTD)
- ▶ Microsoft XML Data Reduced (XDR)-Schema
- ▶ XML Schema Definition Language (XSD)

Der älteste und verbreitetste Standard ist die DTD (Document Type Definition), anerkannt durch das W3C und sehr häufig im Einsatz. Viele Programme unterstützen DTDs und immer noch werden XML-Applikationen mit DTDs geliefert. Eine DTD definiert Elemente, Attribute, Standardwerte und Abhängigkeiten. Sie hat aber einige prinzipielle Nachteile. So ist die DTD-Sprache selbst kein XML-Dialekt, was eigentlich unverständlich ist, denn damit wird die eigene Universalität in Frage gestellt. Die Ursache ist in der Geschichte von XML zu finden. XML entstand aus dem älteren und komplexeren SGML, wo auch verschiedene Darstellformen für die Grammatikbeschreibung und Formatierungen zum Einsatz kamen. Erst mit der anerkannten Universalität von XML als Beschreibungssprache kam die Idee auf, auch Hilfssprachen und Definitionen in XML selbst zu definieren. Weiterhin kennt die DTD keine Datentypen, was den Einsatz von XML in Datenbank-anwendungen erschwert.

DTD

Der Schwerpunkt des Einsatzes von DTDs liegt bei textorientierten XML-Dokumenten.

Relativ früh entwickelte Microsoft vor diesem Hintergrund die Sprache XDR – XML Data Reduced. XDR ist kein offizieller Standard und wird heute nur noch selten eingesetzt. Am bekanntesten dürfte wohl die Implementierung im Biztalk-Server sein. XDR ist selbst konsequent in XML entwickelt.

XDR

Als Antwort auf XDR und den Bedarf nach einem offenen Standard, der die Nachteile von DTDs vermeidet und in der Zukunft erweiterbar ist, wurde XSD entwickelt. XSD steht für XML Schema Definition Language, oft auch kurz als »Schema« bezeichnet. Diese Form der Grammatikbeschreibung ist selbst in XML geschrieben. Der Standard ist beim W3C verabschiedet.

XSD

Der Schwerpunkt des Einsatzes von XSD liegt bei datenorientierten XML-Dokumenten.

Die Unterstützung der Standards in .NET

In .NET werden prinzipiell alle drei Standards unterstützt. Dabei liegt der Fokus eindeutig auf XSD. Es gibt nur sehr selten handfeste Gründe, DTDs oder gar XDR zu verwenden. Der Weg, den VS.NET vorgibt, ist dann auch eindeutig. Direkt – auf der grafischen Oberfläche – wird nur XSD angeboten. Es gibt ein Kommandozeilenwerkzeug, mit dem alte XDR-Schemas nach XSD übersetzt werden. Mit DTDs ist das nicht ganz so einfach, weil eine DTD viele Informationen vermissen lässt, die XSD benötigt. Auf der anderen Seite enthält es solche (beispielsweise Entitäten), die mit XSD nicht mehr definiert werden, weil sie nun Bestandteil des Dokuments sind.



In diesem Buch wird deshalb als Definitionssprache konsequent XSD verwendet, da es im Zusammenhang mit ADO.NET fast immer um die Verarbeitung datenorientierter Dokumente geht.

Eine sehr kurze Einführung in XSD

Die Beschäftigung mit XSD hat praktische Vorteile. Sie können damit die Struktur eines XML-Dokuments beschreiben, wenn Sie eine eigene Grammatik entwerfen und einsetzen. .NET kann diese Schematas lesen und als Tabellen in DataSet-Objekten abbilden. Damit können Sie mit den von ADO.NET bekannten Techniken auf die Daten zugreifen. XSD kann also Abhängigkeiten definieren, Datentypen festlegen und damit auch komplexe Grammatiken vollkommen beschreiben.

Grundaufbau

Prinzipiell ist ein Schema ein XML-Dokument, das folgenden Grundaufbau hat:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
</xs:schema>
```

Darin eingebettet sind die Element- und Typdefinitionen. Der Namensraum, der verwendet wird, heißt bei VS.NET »xs«, bei der W3C-Empfehlung ist »xsd« im Einsatz. Allerdings ist der Namensraum ohnehin wählbar – im Unterschied zu XDR.

Namensraum

Elemente werden folgendermaßen definiert:

```
<xs:element name="elementname"></xs:element>
```

Innerhalb des Elements können weitere Elemente folgen – wenn Ihre Grammatik dies erfordert – oder eine komplexere Typdefinition. Allerdings verfügt XSD bereits über eine hervorragende Liste von Standardtypen. Zeichenketten definieren Sie folgendermaßen:

```
<xs:element name="elementname" type="xs:string">
```

Komplexe Typen setzen sich aus Standardtypen zusammen. Wenn Sie innerhalb eines Tags ganz bestimmte andere Elemente erlauben möchten, definieren Sie einen entsprechenden Typ:

```
<xs:complexType name="newscontenttype">
  <xs:sequence>
    <xs:element name="title" type="xs:string" />
    <xs:element name="text" type="xs:string" />
  </xs:sequence>
  <xs:attribute name="date" type="xs:date" />
</xs:complexType>
```

Dieser Typ definiert einen Datentyp mit dem Namen »newscontenttype«. Enthalten darf ein Tag, das diesen Datentyp trägt, eine unbegrenzte Folge (xs:sequence) der Elemente <title> und <text>. Außerdem trägt das Tag ein Attribut mit dem Namen date und dem Datentyp xs:date. Ein XML-Dokument, das dieser Grammatik entspricht, kann also folgende Strukturen enthalten:

**Element- und
Typdefinitionen**

```
<newscontent date="13.4.2002">
  <title>Titel steht hier</title>
  <text>Hier folgt der Text</text>
</newscontent>
```

Die Nutzung des komplexen Typs ist auf zwei Wegen möglich. Zum einen kann die Definition abstrakt erfolgen, außerhalb aller Elementdefinitionen. Dann wird auf den Typ an der Stelle verwiesen, an der der Datentyp festgelegt wird:

```
<xs:element name="news" type="newscontenttype">
```

Alternativ wird die Definition direkt eingebettet. Das ist vor allem dann sinnvoll, wenn der Typ nur an dieser einen Stelle verwendet wird:

```
<xs:element name="news">
  <xs:complexType name="newscontenttype">
    <xs:sequence>
      <xs:element name="title" type="xs:string" />
      <xs:element name="text" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="date" type="xs:date" />
  </xs:complexType>
</xs:element>
```

Damit ist die Reihenfolge und das Auftreten der Elemente starr festgelegt. Möchten Sie eine beliebige Folge definieren, hilft `xs:choice`:

```
<xs:complexType name="newscontenttype">
  <xs:choice>
    <xs:element name="title" type="xs:string" />
    <xs:element name="text" type="xs:string" />
  </xs:choice>
</xs:complexType>
```

Anzahl der Elemente

Die Anzahl der Elemente kann ebenso einfach festgelegt werden. Dazu wird die Elementdefinition um die Attribute `minOccurs` und `maxOccurs` ergänzt:

```
<xs:element name="title" type="xs:string"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="text" type="xs:string"
  minOccurs="0" maxOccurs="unbounded"/>
```

Hier wird festgelegt, dass der Titel (`<title>`) genau ein Mal auftreten darf, während der Textblock `<text>` optional ist und unbegrenzt wiederholt werden kann.

Der Einsatz kann auch in `xs:choice` erfolgen. Da die Elemente in genau der Folge auftreten müssen, wie sie definiert wurden, ist dies ein häufig eingesetztes Mittel, um die Reihenfolge wieder freizustellen. Denn wenn Sie `minOccurs` auf 0 setzen, kann ein Element auch entfallen. Damit rückt aber das nachstehende in der Folge nach vorn, was einer Aufgabe der Reihenfolge gleichkommt.

Soll ein Tag in einer Sequenz von Tags genau ein Mal auftreten und ist die Reihenfolge nicht von Belang, ist dies mit `xs:all` einfacher zu bewerkstelligen:


```
<xs:all>
  <xs:element name="text" type="xs:string"/>
  <xs:element name="picture" type="xs:href"/>
</xs:all>
```

Attribute wurden bereits kurz angesprochen. Sie werden wie Elemente definiert, stehen aber in komplexen Definitionen immer nach allen `xs:sequence`-, `xs:choice`- oder `xs:all`-Definitionen.

Attribute

In Attributen kommen häufig feste Parameterlisten zum Einsatz, beispielsweise "on" und "off". Diese werden als einfache, selbst-definierte Datentypen deklariert. Die folgende Definition zeigt ein solches Attribut:

**Attribute und
Datentypen**

```
<xs:attribute name="encoding" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="on"/>
      <xs:enumeration value="off"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

Das Attribut `use` wird hier verwendet, um die Angabe eines Attributes »encoding« im Tag zu erzwingen. Mit `base` wird der Basistyp festgelegt, auf dem die zulässigen Parameter basieren. Bei "on" und "off" handelt es sich zwangsläufig um Zeichenketten.

`xs:restriction` ist das Mittel, den Wertebereich von Datentypen allgemein einzuschränken. Dazu dienen folgende Definitionen:

XSD-Datentypen

► `<xs:minInclusive value="1">`

Diese Definition ist für Zahlenwerte interessant und definiert den kleinsten Wert eines Ganzzahlbereiches.

► `<xs:maxInclusive value="100">`

Mit dieser Definition wird der maximale Wert eines Ganzzahlbereiches festgelegt.

► `<xs:length value="5"/>`

Bei Zeichenketten kann die Länge begrenzt werden, die die Zeichenkette annehmen darf.

► `<xs:pattern value="[a-z._]+@[a-z.]\.[a-z]{2,4}"/>`

Komplexere Bedingungen lassen sich mit regulären Ausdrücken festlegen.

Entitäten sind mit DTDs leicht zu definieren. Dabei werden Folgen der Art € durch feste Zeichen oder Zeichenketten ersetzt. Das steht prinzipiell im Widerspruch zur Art der XML-Darstellung. Im Zusammenhang mit XSD sieht eine solche Definition dann so aus:

```
<xsd:element name="euro" type="xsd:token" fixed="€"/>
```



Beachten Sie, dass dies nur eine sehr kompakte und stark vereinfachende Darstellung von XSD war. Es sollte aber ausreichend sein, um die mit VS.NET erzeugten Dokumente zu interpretieren oder für einfache Anwendungen ein Schema zu schreiben, wenn VS.NET nicht zur Verfügung steht.



Eine Warnung ist wichtig, wenn Sie mit VS.NET arbeiten: XSD wurde von Microsoft etwas erweitert, das heißt, es stehen mehr Typen und Elemente zur Verfügung, als der Standard verlangt. Dies ist darin begründet, dass mit XML-Dokumenten Daten aus relationalen Datenbanken abgebildet werden sollten. DataSet verwenden in der internen Darstellung immer XML, egal wie Ihre Datenquelle aussieht. XSD kann aber nicht alles abbilden, was in SQL Server oder Access möglich ist. Die Erweiterungen lösen das Problem. Für den Import fremder XSD-Dateien stellt dies kein Problem dar. Umgekehrt können Informationen aus der Definition verloren gehen.

5.6.3 XML im praktischen Einsatz

Als erstes Beispiel für den praktischen Einsatz soll der Aufbau eines Nachrichtendienstes auf einer Website dienen. Dabei sind folgende Schritte zu erledigen:

1. Wahl eines geeigneten Formats für die Datenstruktur
2. Entwicklung des Schemas und der XML-Datenquelle
3. Festlegung eines passenden Steuerelementes für die Anzeige
4. Bindung der XML-Datenquelle an das Steuerelement

Am Anfang soll ein einfaches Datenformat genügen. Nachrichten bestehen mindestens aus einem Titel und einem Text. Zusätzlich könnte ein Datum sinnvoll sein, beispielsweise um alte Nachrichten automatisch aussortieren zu können.

Format der Datenstruktur

Beim Design eines XML-Dialekts sind einige prinzipielle Überlegungen anzustellen. So kann man mit Elementen direkt verbun-

dene und kurze Daten als Attribute definieren. Längere Inhalte und natürlich alle wiederholenden Knoten werden als Elemente definiert. Im Beispiel wäre zuerst ein globales Element zu erstellen, das alle Nachrichten enthält. Es soll den Namen »news« erhalten. Diesem sind dann Informationen zugeordnet. Diese Elemente werden als »newscontent« bezeichnet. Jedes Element kann zwei weitere Elemente enthalten, »title« und »text«. Das Datum wird dem Element »newscontent« als Attribut »date« zugeordnet.

Damit dürfte das Format hinreichend flexibel sein. Den Titel als Attribut zu definieren, erscheint nicht günstig. Spätere Versionen könnten mit Untertiteln arbeiten, dann wäre auf jeden Fall ein Element erforderlich – Attribute sind Endpunktknoten. Beim Datum werden kaum Teile von Interesse sein. Weitere Datumsangaben – wie ein in der Zukunft liegendes Datum, ab dem die Information aktiv wird – lassen sich gut als weitere Attribute definieren.

Im letzten Abschnitt wurden bereits theoretische Ausführungen zu XSD gezeigt. Sie sollten die starke Unterstützung in Visual Studio .NET für XML und Schema nutzen, um gleich ein professionelles Datenregime aufzubauen. Legen Sie also zuerst eine XSD-Datei an:

**XML und XSD im
Visual Studio .NET**

1. Fügen Sie Ihrem Projekt ein neues XML-Schema hinzu. Dazu wählen Sie im Kontextmenü HINZUFÜGEN | NEUES ELEMENT HINZUFÜGEN.
2. Im folgenden Dialog suchen Sie das Symbol XML-SCHEMA.
3. Nennen Sie die Datei *news.xsd*.

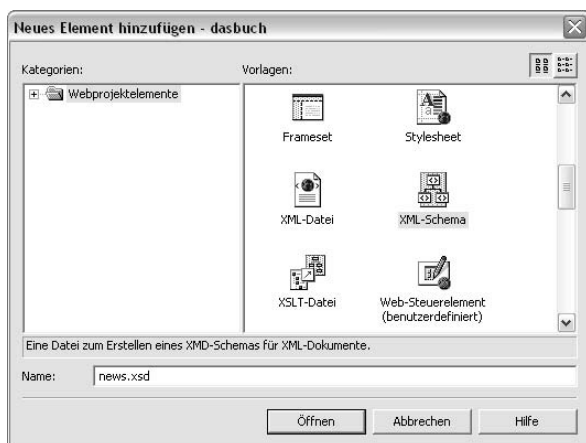


Abbildung 5.21: Erzeugen einer neuen Schema-Datei in Visual Studio .NET

Es erscheint eine leere Fläche, auf der Sie nun die Struktur der XML-Daten bestimmen können. Wenn Sie später Daten hinzufügen, wird die Validierung des Dokuments auf der Grundlage dieser Schema-Datei erfolgen.

Zuerst ziehen Sie ein Element aus der Toolbox am linken Rand auf die Fläche. Benennen Sie es mit »news« im linken Feld der Anzeige und schreiben Sie »NewsType« auf die rechte Seite, wo der Datentyp zugeordnet wird. Da dieses oberste Element nur weitere eigene Elemente enthält, ist dies ein benutzerdefiniertes Format.

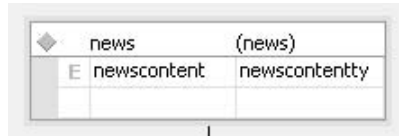


Abbildung 5.22: Element mit benutzerdefiniertem Format

E element

Ziehen Sie nun ein Element vom Typ »element« auf das Rechteck, das den neuen komplexen Typ bildet: »newscontent«. Schreiben Sie als Datentyp nun »newscontenttype« hinein. Der Name ist wahlfrei – solange er eindeutig ist – und wird nachfolgend zur Definitionen eines »ComplexType« verwendet.

CT complexType

Ziehen Sie jetzt ein Element vom Typ »ComplexType« aus der Toolbox auf die freie Fläche und benennen Sie es »NewsType«. Dieser komplexe Typ enthält die eigentlichen Elemente. Dazu erzeugen Sie in der Liste, entweder durch Auswahl eines Typs am linken Rand der Definitionsfläche oder durch Drag&Drop aus der Toolbox die benötigten Elemente »title« und »text« sowie das Attribut »date«. Dem Attribut »date« geben Sie als Datentyp date, den anderen Elementen string.

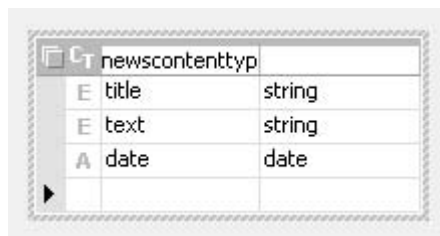


Abbildung 5.23: Die gesamte Definition des komplexen Typs

XML-Dateien erzeugen und das Schema zuordnen

Der nächste Schritt besteht nun darin, die XML-Datei zu erzeugen und mit dem Schema zu verbinden. Auf dieser Basis kann dann die Dateneingabe erfolgen, verbunden mit den nun möglichen Gültigkeitsprüfungen. Mit der XSD-Datei erfolgt die Prüfung sowohl zum Entwurfszeitpunkt in VS.NET, als auch später beim Verarbeiten der Daten in der Applikation.

Eine neue XML-Datei fügen Sie wie jede andere Datei im Projekt-Explorer hinzu. Vergeben Sie hier den Namen *news.xml*. Ist die Datei erzeugt, wird sie angezeigt. Bis auf den XML-Kopf ist nichts enthalten. Mit **[F4]** gelangen Sie zum Eigenschaften-Dialog. Wählen Sie dort die Gruppe DOCUMENT aus, falls sie nicht ohnehin bereits vorgegeben ist. In der Zeile TARGETSCHEMA können Sie nun die XSD-Datei mitsamt dem Standardpräfix *http://tempuri.org* auswählen.

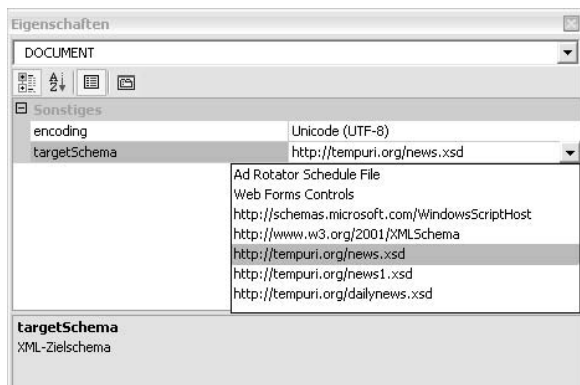


Abbildung 5.24: Zuordnung der Schema- zur XML-Datei

Daten in einer XML-Datei erfassen

Die XML-Datei ist zwar immer noch leer, aufgrund des Schemas können Sie dennoch sofort mit der Datenerfassung beginnen. Wechseln Sie dazu in die Datenansicht, entweder mit **[Strg]-[Bild ↓]** oder über das Menü **ANSICHT | DATEN**. Sie erhalten eine Tabellenansicht, wie Sie es von SQL Server oder Access gewohnt sind. Die Datenerfassung ist nun kein Problem mehr, im Übrigen auch nicht mit anderen Werkzeugen, die XSD lesen können, was inzwischen für fast alle professionellen XML-Editoren zutrifft.



Abbildung 5.25: Bearbeitungsansicht der XML-Datei in VS.NET

5.6.4 Verarbeitung einfacher XML-Daten

Es bietet sich an, die Verarbeitung der XML-Datei mit einem Daten-Steuerelement zu erledigen. Damit es wiederverwendbar ist, käme außerdem die Auslagerung in ein Benutzer-Steuerelement in Betracht. Um die Anzeige universell zu gestalten, ist ein Repeater-Steuerelement die beste Wahl. Selbstverständlich sind Sie nicht darauf beschränkt; die folgenden Beispiele ließen sich ähnlich auch mit DataGrid oder DataList umsetzen.

Platzierung der Anzeige

Die Nachrichten sollen möglicherweise an mehreren Stellen verwendet werden. Ein ausreichend »intelligentes« Steuerelement ist die ideale Lösung dazu. Die Deklaration und Verwendung sieht folgendermaßen aus:

```
<%@ Register TagPrefix="uc" TagName="news" Src="newsControl.ascx"%>
<html>
  <head>
  </head>
  <body>
    <uc:news runat="server"/>
  </body>
</html>
```

Listing 5.1: So wird die Ausgabe der Nachrichten eingebunden (newsControl.aspx)

Im nächsten Schritt muss das Benutzer-Steuerelement selbst entworfen werden.

Gestaltung des Benutzer-Steuerelements

Das Benutzer-Steuerelement enthält ein Repeater-Steuerelement, indem mit der Datenbindungssyntax auf die beiden entscheidenden Tags »title« und »text« zugegriffen wird.

```
<%@ Control Language="vb" AutoEventWireup="false"
    Codebehind="newsControl.ascx.vb"
    Inherits="newsControl"%>
<asp:Repeater ID="newsRepeater" Runat="server">
    <HeaderTemplate>
        <ul><B>Neuigkeiten</B>
    </HeaderTemplate>
    <ItemTemplate>
        <li>
            <b><%=# DataBinder.Eval(Container.DataItem, "title")
%></b><br/>
            <%=# DataBinder.Eval(Container.DataItem, "text") %>
        </li>
    </ItemTemplate>
    <FooterTemplate>
        </ul>
    </FooterTemplate>
</asp:Repeater>
```

Listing 5.2: Der HTML-Teil des Benutzer-Steuerelements (newsControl.ascx)

Der Zugriff auf die XML-Daten erfolgt in der Code-Datei *newsControl.ascx.vb*. Diese Datei enthält einige private Variablen:

Wie es funktioniert

```
Private sCurrentXml As String
Private sCurrentSchema As String
Private dsNews As DataSet
Public newsRepeater As Repeater
```

Abgesehen von den beiden Hilfsvariablen am Anfang ist der Typ *DataSet* von Interesse. Die Ablage der XML-Daten erfolgt dort. Außerdem muss natürlich der öffentliche (*public*) Zugriff auf den Repeater ermöglicht werden.

Der Zugriff auf die im Web abgelegten Datendateien – dies betrifft sowohl *news.xsd* als auch *news.xml* – erfolgt über die Berechnung des physikalischen Pfades. Die folgende Methode ermittelt diesen Pfad jeweils. Im Beispiel sind die Datendateien in einem Unterverzeichnis mit dem Namen *data* zu finden.

Zugriff auf Datendateien

```
Private Function getConfig(newsfile As String) As String
    Dim s As String
```

```

s = Server.MapPath("data")
s = s & "\" & newsfile
Return s
End Function

```

Die eigentliche Arbeit wird dann in der `Page_Load`-Methode erledigt. Falls Sie die Nachrichtenausgabe innerhalb eines Formulars verwenden, sollte die Zuweisung beim *PostBack* nicht erneut erfolgen, der aktuelle Zustand wird dann im Anzeigestatus (*ViewState*) erhalten. Ändern sich die Daten in der XML-Datei schnell und sollen sich diese Änderungen sofort widerspiegeln, weisen Sie den aktuellen Zustand immer wieder zu, so wie es im Beispiel gemacht wird.

```

Private Sub Page_Load(sender As Object, e As System.EventArgs)
    sCurrentSchema = getConfig("news.xsd")
    sCurrentXml = getConfig("news.xml")

```

Innerhalb eines `try`-Zweiges wird dann das `DataSet`-Objekt mit den Daten aus der XML-Datei gefüllt. Dazu wird zuerst ein leeres `DataSet` erzeugt:

```
dsNews = New DataSet()
```

Dann wird das Schema zur Festlegung der Datenstruktur des `DataSet` zugewiesen:

```
dsNews.ReadXmlSchema(sCurrentSchema)
```

Dann wird die XML-Datei unter Verwendung des Schemas eingelesen:

```
dsNews.ReadXml(sCurrentXml, XmlReadMode.ReadSchema)
```

Die Daten sind nun schon sehr universell verwendbar, lediglich die Bindung an das `Repeater`-Steuerelement ist noch zu erledigen:

```

newsRepeater.DataSource = dsNews
newsRepeater.DataMember = "newscontent"
newsRepeater.DataBind()

```

Nun ist es an der Zeit, die XML-Datei mit ein paar Zeilen zu füllen und das Programm auszuprobieren, was zu etwa folgendem Ergebnis führt:

Neuigkeiten

- **SharpTemplate veröffentlicht**
Heute wurde die erste öffentliche Version der Entwicklungsumgebung SharpTemplate vorgestellt.
- **ASP-Kongress**
Burghausen beginnt, erstmals auch mit einer eigenen Veranstaltung zu ASP.NET
- **QualityHosting**
preiswert hosten auf den W2000-Systemen von Qualityhosting.de

Abbildung 5.26: Die News-Ausgabe mit den Daten aus der XML-Datei in Aktion

Daten filtern und sortieren

Ab dem Punkt, wo die Daten im `DataSet`-Objekt sind, können Sie das Thema XML eigentlich vergessen. Alle Optionen, die bei ADO.NET verwendet werden können, stehen hier zur Verfügung. Tatsächlich können Sie die Daten auch filtern und sortieren – ohne das eine Datenbank zum Einsatz kommt. Vor allem bei kleineren Datenmengen, die sehr effizient in `DataSet`-Objekten verwaltet werden können, ist XML gegenüber Datenbanken klar im Vorteil. Bei komplexen Relationen, Berechnungen oder großen Datenmengen sieht das freilich anders aus.

5.7 XML mit .NET verarbeiten

Das Beispiel im letzten Abschnitt zeigte eine einfache Anwendung von XML, die auf die speziellen XML-Klassen in .NET offensichtlich verzichten konnte. Im Prinzip kam hier nur ADO.NET zum Zuge². Für größere Projekte wird dies nicht ausreichend sein, zumal die Verarbeitung mit `DataSet`-Objekten nicht immer die einfachste und beste Lösung ist.

Wie nicht anders zu erwarten war, hat Microsoft eine sehr umfassende direkte Unterstützung für XML ins .NET-Framework implementiert. Die Basis bildet der Namensraum `System.Xml`. Für den Zugriff via DOM dienen die Klassen aus `System.Xml.XPath` und für Transformationen `System.Xml.Xsl`. Informationen zu XPath finden Sie in Abschnitt 5.8, »Daten suchen mit XPath« ab Seite 470, während Transformationen mit .NET in Abschnitt 5.9, »Daten transformieren mit XSLT« ab Seite 478 behandelt werden.

² Das ist so nicht ganz korrekt. Die XML-Klassen werden tatsächlich verwendet, allerdings erledigte dies das Framework intern. Diese »heimliche« Nutzung von XML ist sehr umfassend, entsprechende Kenntnisse darüber erleichtern das Verständnis für viele alltägliche Vorgänge.

5.7.1 XML-Dokumente direkt lesen

XmlReader Für das Lesen von XML-Dokumenten bietet die Klasse `XmlReader` den schnellsten Weg. Der Parser prüft dabei jedoch nur auf Wohlgeformtheit. Eine Gültigkeitsprüfung mit Hilfe eines Schemas oder einer DTD erfolgt nicht. Es obliegt dem weiterverarbeitenden Programm, die Daten als gültig oder nicht gültig zu interpretieren. `XmlReader` ist eine abstrakte Klasse, die noch implementiert werden muss. Von `XmlReader` werden bereits drei Klassen abgeleitet, die eine konkrete Implementierung vornehmen:

► `XmlTextReader`

Dies ist die einfachste Implementierung, die nicht mehr bereit stellt als die Basisklasse selbst.

► `XmlValidatingReader`

Mit dieser Klasse kann zusätzlich die Gültigkeitsprüfung auf der Basis eines Schemas vorgenommen werden. Die Verarbeitung ist deshalb geringfügig langsamer.

► `XmlNodeReader`

Diese Klasse erlaubt das Lesen eines Knotens auch dann, wenn es sich um eine Teilstruktur eines Dokumentes handelt, wie sie vom DOM des Dokuments abgeleitet werden kann. Eine Gültigkeitsprüfung wird nicht unterstützt.

Einen guten Einblick in die Arbeitsweise der `XmlTextReader`-Klasse bietet folgender Code. Die Ausgabe erfolgt an ein `Label`-Steuerelement.

```
Private Sub Page_Load(ByVal sender As Object,   
                        ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
    Dim xreader As XmlTextReader   
        = New XmlTextReader(Server.MapPath("data/news.xml"))  
    While xreader.Read()  
        XmlOutput.Text += String.Format("<b{0}</b>&lt;{1}&gt; ",   
                                         = {2:<br>",   
                                         xreader.NodeType.ToString(),   
                                         xreader.Name, xreader.Value)  
    End While  
End Sub
```

*Listing 5.3: Sequenzielles Lesen und Ausgeben eines XML-Dokuments
(Ausschnitt aus `XmlReader1.aspx.vb`)*

Der Konstruktor der Klasse erlaubt die Angabe eines Pfades zur XML-Datei. Er besitzt viele Überladungen, sodass der Umgang mit anderen Datenquellen ohne weiteres möglich ist. Von Interesse ist die Angabe eines Objekts einer der `Stream`-Klassen, wie beispielsweise `FileStream` oder `TextReader`.

Wie es funktioniert

Nach dem Zugriff kann das XML-Dokument sequenziell durchlaufen werden, wozu die Methode `Read` eingesetzt wird. Sind keine Daten mehr vorhanden, wird `false` zurückgegeben. Mit Hilfe der Eigenschaften `NodeType`, `Name` und `Value` kann dann untersucht werden, welche Knotentypen gelesen wurden, wie sie heißen und was sie enthalten. Die Ausgabe ist zumindest im Hinblick auf die Arbeitsweise des Parsers aufschlussreich, wenn auch zugegebenermaßen völlig unpraktisch.

```
XmlDeclaration<xml> = "version="1.0" encoding="utf-8"
Whitespace<> = " "
Element<news> = ""
Whitespace<> = " "
Element<newscontent> = ""
Whitespace<> = " "
Element<title> = ""
Text<> = "Sharp Temple veröffentlicht"
EndElement<title> = ""
Whitespace<> = " "
Element<text> = ""
Text<> = "Heute wurde die erste öffentliche Version der Entwicklungsumgebung Sharp Temple vorgestellt."
EndElement<text> = ""
```

Abbildung 5.27: Erster Teil des XML-Dokuments *news.xml* aus dem letzten Abschnitt

Das sequenzielle Lesen erlaubt den Zugriff auf alle Knoten. Dabei ist an der Ausgabe des letzten Beispiels zu erkennen, dass auch die Leerräume (allgemein als »Whitespaces« bezeichnet) zwischen Tags als Elemente erkannt werden. Prinzipiell ist es nicht notwendig, bei der Verarbeitung von XML auf die Leerräume zu achten, denn sie sind zur Trennung nicht zwingend notwendig. Die weitere Verarbeitung kann aber darauf Rücksicht nehmen. Wenn die Ausgabe von XML künftig auch für Menschen lesbar sein soll, ohne dass eine explizite Transformation in ein typischerweise für Menschen geeignetes Format stattfindet, ist der Erhalt von Einrückungen, Trennfeldern und Zeilenumbrüchen durchaus angeraten.

Umgang mit Whitespaces

Ansonsten trennt der Parser bei Container-Tags noch zwischen dem öffnenden Teil, dem Inhalt des Elementes und dem schließenden Teil. Beim Inhalt führt die Eigenschaft `NodeType` den Wert `Text`.

5.7.2 XML selbst erzeugen

Ähnlich einfach wie der Zugriff auf `XmlTextReader` erfolgt, kann auch `XmlTextWriter` verwendet werden. Auch hier wird die Verknüpfung mit einem `Stream` erwartet, ein Ziel also, dass die erzeugten Daten aufnimmt. In der Webprogrammierung kann der Einsatz häufig erfolgen, wenn Sie damit rechnen, dass die Clients XML direkt verarbeiten können. Der Internet Explorer kann dies ebenso wie der Netscape Navigator ab Version 6.0. Das folgende Beispiel liest eine Dateistruktur von der Festplatte und gibt sie als XML-Datei direkt an den Browser aus. XML bietet sich hierfür geradezu an, denn die Struktur der Dateien im Dateisystem ist hierarchisch.

Die direkte Ausgabe der Daten an den Browser spart die Definition von HTML. Deshalb ist die *aspx*-Datei bis auf den Aufruf der Code-Datei leer:

```
<%@ Page language="vb" Codebehind="XmlWriter1.aspx.vb"
      AutoEventWireup="false"
      Inherits="dasbuch.XmlWriter1" %>
```

Listing 5.4: Die XML-Daten werden ohne HTML-Anteil versendet (XmlWriter1.aspx)

Die Klasse *XmlWriter1* muss nun zwei Dinge erledigen: Lesen einer Dateihierarchie und Versenden als XML. Der Startpunkt im Dateisystem kann mit einem GET-Parameter festgelegt werden, sodass die Lösung auch in dieser Hinsicht universell ist. Zuerst ein Blick auf die `Page_Load`-Methode, in der die Verarbeitung beginnt:

```
Private Sub Page_Load(ByVal sender As Object,
                    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Dim path As String
    If Request.QueryString("path") Is Nothing Then
        path = "."
    Else
        path = Request.QueryString("path")
    End If
    xWrite = New XmlTextWriter(Response.Output)
    xWrite.WriteStartDocument()
    xWrite.WriteStartElement("dirlist")
    xWrite.WriteAttributeString("path", Server.MapPath(path))
    ReadDir(Server.MapPath(path))
    xWrite.WriteEndDocument()
End Sub
```

*Listing 5.5: Start und Ausgabe von XML-Daten direkt an den Browser
(Ausschnitt aus XmlWriter1.aspx.vb)*

Wie bereits am Anfang angedeutet, wird die Ausgabe über Stream-Objekte gesteuert. Eines davon ist `TextStream`, das auch von `Response.Output` implementiert wird. Die gesamte Ausgabe wird deshalb allein durch die folgenden Zeile ausgelöst; ein Verfahren, das auch bei der Ausgabe von Bilddaten Verwendung findet:

Wie es funktioniert

```
xWrite = New XmlTextWriter(Response.Output)
```

Dann wird der Kopf eines XML-Dokuments geschrieben, also die erste Zeile mit »<?xml ...« am Anfang:

```
xWrite.WriteStartDocument()
```

Dann wird – konform zur XML-Spezifikation – ein Wurzelement geschrieben, dass die gesamte Datei umschließt. Es trägt hier den Namen `<dirlist>`:

```
xWrite.WriteStartElement("dirlist")
```

Dieses Element erhält ein Attribut, das den Pfad enthält und folgendermaßen erzeugt wird:

```
xWrite.WriteAttributeString("path", Server.MapPath(path))
```

Dann wird die Methode `ReadDir` aufgerufen, die es noch zu entwickeln gilt und die für die Ausgabe der Dateien und Unterverzeichnisse verantwortlich ist. Anschließend wird das Dokument geschlossen.

In `ReadDir` steckt die eigentliche Funktionalität. Zuerst das Listing dieser Methode auf einen Blick:

Das Dokument erzeugen

```
Private Sub ReadDir(ByVal current As String)
    Dim di As DirectoryInfo = New DirectoryInfo(current)
    xWrite.WriteStartElement("directory")
    Try
        If di.GetFiles().Length > 0 Then
            xWrite.WriteAttributeString("hasfiles", "true")
        End If
        xWrite.WriteAttributeString("name", di.Name)
        Dim sdi As DirectoryInfo
        For Each sdi In di.GetDirectories()
            ReadDir(sdi.FullName)
        Next
        Dim fi As FileInfo
        For Each fi In di.GetFiles()
            xWrite.WriteStartElement("file")
            xWrite.WriteString(fi.Name)
            xWrite.WriteEndElement()
        Next
    Catch
    End Try
    xWrite.WriteEndElement()
End Sub
```

```

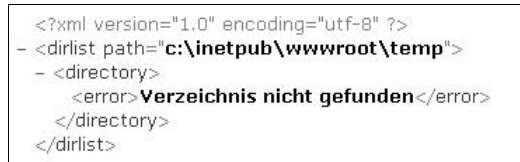
        Next
    Catch
        xWrite.WriteStartElement("error")
        xWrite.WriteString("Verzeichnis nicht gefunden")
        xWrite.WriteEndElement()
    Finally
        xWrite.WriteEndElement()
    End Try
End Sub

```

*Listing 5.6: Lesen einer Verzeichnishierarchie und Schreiben der XML-Daten
(Fortsetzung des Ausschnitts aus XmlWriter1.aspx.vb)*

Wie es funktioniert

Die Methode arbeitet rekursiv, wobei das Erzeugen der XML-Tags jeweils am Anfang erfolgt; am Ende wird das Tag dann geschlossen (`WriteEndElement`). Bei Dateizugriffen kommt es häufig vor, dass Pfadangaben nicht stimmen. Deshalb wird noch eine *Try-Catch-Finally*-Anweisung verwendet, die sicherstellt, dass auch im Fehlerfall gültiges XML erzeugt wird. Die folgende Abbildung zeigt, wie der Internet Explorer diesen Fehlerzustand anzeigt.



```

<?xml version="1.0" encoding="utf-8" ?>
- <dirlist path="c:\inetpub\wwwroot\temp">
  - <directory>
    <error>Verzeichnis nicht gefunden</error>
  </directory>
</dirlist>

```

Abbildung 5.28: Ausgabe eines »Fehlerdokuments« bei einem nicht vorhandenen Pfad

Läuft alles nach Plan, wird zuerst ermittelt, ob das Verzeichnis Dateien enthält:

```
If di.GetFiles().Length > 0 Then
```

Ist das der Fall, wird ein Attribut `hasfile="true"` erzeugt:

```
xWrite.WriteAttributeString("hasfiles", "true")
```

Dann wird im Tag `<directory>` der Name des Verzeichnisses als weiteres Attribut ausgegeben:

```
xWrite.WriteAttributeString("name", di.Name)
```

Sind weitere Verzeichnisse vorhanden, erfolgt nun der rekursive Aufruf der Methode, die zu weiteren verschachtelten `<directory>`-Tags führen, wenn entsprechende Daten vorliegen:

```

For Each sdi In di.GetDirectories()
    ReadDir(sdi.FullName)
Next

```

Anschließend werden, soweit vorhanden, die Dateien durchlaufen und Tags vom Typ `<file>` erzeugt:

```

For Each fi In di.GetFiles()
    xWrite.WriteStartElement("file")
    xWrite.WriteString(fi.Name)
    xWrite.WriteEndElement()
Next

```

Mit der passenden Pfadangabe kann nun schon eine ansprechende Ausgabe erzeugt werden. Bevor Sie eigene Versuche starten, sollten Sie daran denken, dass der Abruf eines Verzeichnisses mit umfangreichem Inhalt zum einen erheblich Zeit in Anspruch nimmt, zum anderen aber auch zu sehr großen XML-Dateien führt.

```

<?xml version="1.0" encoding="utf-8" ?>
- <dirlist path="c:\inetpub\wwwroot\dasbuch">
- <directory hasfiles="true" name="dasbuch">
  - <directory hasfiles="true" name="bin">
    <file>Buch.dll</file>
    <file>Buch.pdb</file>
  </directory>
  - <directory hasfiles="true" name="data">
    <file>news.xml</file>
    <file>news.xsd</file>
    <file>news.xsx</file>
  </directory>
  - <directory name="mssamples">
    - <directory name="downloads">
      - <directory name="samples">
        - <directory name="internet">
          - <directory name="xml">
            - <directory hasfiles="true" name="xsl_debugger">
              <file>copy.xsl</file>
              <file>dbg-buttons.htm</file>
              <file>dbg-copy-xml.xsl</file>
              <file>dbg-copy-xsl.xsl</file>
              <file>dbg-format-xsl.xsl</file>
              <file>dbg-format.xsl</file>
              <file>Readme.txt</file>
              <file>test.xml</file>
              <file>xsl-debugger.htm</file>
            </directory>
          </directory>
        </directory>
      </directory>
    </directory>
  </directory>
  - <directory hasfiles="true" name="License">
    <file>License.txt</file>
  </directory>
</directory>
+ <directory hasfiles="true" name="wrox">

```

Abbildung 5.29: Teil einer umfassenderen Ausgabe, die das gezeigte Programm erzeugt

Nun kann mit dieser Ausgabe vermutlich kein »menschlicher« Rezipient etwas anfangen. Es ist also noch eine Überlegung notwendig, dies zu gestalten. Dafür bietet sich XSLT an. An dieser Stelle soll der Internet Explorer, der mit MSXML ein clientseitiges Transformationsprogramm enthält, die Arbeit übernehmen. Im Abschnitt 5.9, »Daten transformieren mit XSLT« ab Seite 478 finden Sie eine Erweiterung dieses Beispiels, die die Arbeit von .NET erledigen lässt.

Mit XSLT spielen



XSLT transformiert XML-Daten, sodass die Darstellung von der Struktur der ursprünglichen Daten abweicht. In VS.NET wählen Sie im Kontextmenü des Projekts die Option HINZUFÜGEN | NEUES ELEMENT HINZUFÜGEN und in der folgenden Auswahl XSLT-DATEI. Das folgende Listing zeigt eine solche Datei, abgestimmt auf die zuvor mit `XmlTextWriter` erzeugte Verzeichnisausgabe:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="dirlist">
    <html>
      <head>
        <title>Verzeichnisliste</title>
      </head>
      <body>
        <basefont face="Verdana"/>
        <h1><xsl:value-of select="./@path"/></h1>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="error">
    <div style="color:red; font-weight:bold">
      <xsl:value-of select="."/>
    </div>
  </xsl:template>
  <xsl:template match="directory">
    <ul><b><xsl:value-of select="./@name"/></b><br/>
      <xsl:apply-templates/>
    </ul>
  </xsl:template>
  <xsl:template match="file">
    <li>
      <xsl:apply-templates />
    </li>
  </xsl:template>
</xsl:stylesheet>
```



```

    </li>
  </xsl:template>
</xsl:stylesheet>

```

Listing 5.7: XSLT-Datei zur Transformation der automatisch erzeugten Ergebnisse (ShowFiles.xslt, im Stammordner des Projekts)

Die Transformationsanweisungen – jeweils als `<xsl:template>` bezeichnet – behandeln jedes Tag der XML-Daten einzeln. Dieses einfache Beispiel nutzt die Verschachtelungsfähigkeit des HTML-Tags `<ul>`, um den Verzeichnisbaum darzustellen. Das Tag `<dirlist>` erzeugt eine HTML-Seite, der Titel (`<h1>`) wird dabei dem Attribut `path` entnommen:

```
<h1><xsl:value-of select="./@path"/></h1>
```

Ansonsten sind drei Templates notwendig, denn es gibt drei Tags, die erzeugt werden (`<error>`, `<directory>` und `<file>`). Für jedes neue Verzeichnis tritt `<directory>` auf, das zum Erzeugen eines `<ul>`-Tags führt. Da der Name von Interesse ist, wird dieser zuerst dem Attribut `name` entnommen:

```
<ul><b><xsl:value-of select="./@name"/></b><br/>
```

Der Umgang mit XSLT ist aufgrund des funktionalen Programmierstils, der sich völlig vom imperativen oder objektorientierten unterscheidet, nicht trivial. Vor allem ist beim Entwurf einer XML-Grammatik die spätere Verwendungsweise zu beachten. Denn nicht alle Fälle, die der Internet Explorer in seiner XML-Ansicht problemlos anzeigt, lassen sich mit XSLT ebenso einfach transformieren. Konsultieren Sie ggf. die Tipps am Anfang des Kapitels.



Die Ausgabe, mit etwas CSS verfeinert, zeigt Abbildung 5.30. Interessant am Zusammenspiel der erzeugten XML-Daten und des XSLT-Stylesheets ist die Tatsache, dass bei der Transformation offensichtlich rekursiv gearbeitet wird. Gerade hierarchische Strukturen, die beliebig verschachtelt sein können, sind mit prozeduralen Programmen nicht ganz so einfach zu verarbeiten.

Dies soll vor allem als Anregung dafür dienen, XML als Transport- oder Speichermedium in bestimmten Fällen bevorzugt als Alternative zur konventionellen Programmierung zu erkennen.

Wie es funktioniert

c:\inetpub\wwwroot\dasbuch



Abbildung 5.30: Der Verzeichnisbaum im Internet Explorer, formatiert mit XSLT und CSS

5.8 Daten suchen mit XPath

Mit Hilfe einer Transformation wurde im letzten Abschnitt bereits eine Auswahl aus XML-Daten vorgenommen. Immer dann, wenn auf Knoten oder Inhalte mit `match=""` oder `select=""` zugegriffen wurde, kam bereits XPath zum Einsatz. Diese einfache Form reicht in der Praxis nur selten aus. Der vollständige Name für XPath lautet *XML Path Language 1.0*.

5.8.1 Eine Einführung in XPath

Um in der hierarchischen Struktur eines XML-Dokuments gezielt Knoten adressieren zu können, wird XPath eingesetzt. Ohne eine solche Sprache würden Dokumente immer sequenziell durchlaufen werden, was wenig praxistauglich ist. Gerade bei Webanwendungen, die oft vielen Benutzern einen Ausschnitt aus einer großen Datenmenge zur Verfügung stellen, ist die schnelle Auswahl eminent wichtig.

XPath realisiert die Abfrage durch Beschreibung eines Pfades und verzichtet dabei auf Schleifen oder andere zyklische Elemente. Damit ist die Konstruktion zur Laufzeit und in Abhängigkeit vom aktuellen Auftreten von Knoten zueinander möglich. Wie der Name der Sprache andeutet, ähnelt die Auswahl von Knoten den Pfadangaben im Dateisystem eines Betriebssystems. Das ist nahe liegend, weil auch dort Daten hierarchisch angeordnet sind. Eine typische XPath-Anweisung könnte also folgendermaßen aussehen:

Pfade zum Ziel

```
eintrag/name/vorname
```

Sie adressiert einen Knoten `<vorname>`, der Kind von `<name>` ist, was wiederum Kind von `<eintrag>` sein muss:

```
<eintrag>
  <name>
    <vorname>
```

Das Beispiel in Abbildung 5.27 zeigte, dass es verschiedene Knotentypen in XML gibt. XPath muss diese adressieren können. Konkret unterschieden werden die in der folgenden Tabelle dargestellten Typen:

XPath-Knoten	Darstellung
Wurzelknoten	/
Elementknoten	ElementName
Attributknoten	@Attribut
Textknoten	., text()
Prozessinformation	Nicht darstellbar
Namensraum	alias:

Tabelle 5.14: Auswahl von Knotentypen mit der XPath-Syntax

Grundlagen für die Entwicklung von Ausdrücken

XPath basiert auf Ausdrücken, die den Weg zu einem Knoten beschreiben. Der Weg kann – ebenso wie beim Dateisystem – durch absolute oder relative Pfadangaben beschrieben werden. Absolute Angaben beginnen immer an der Dokumentenwurzel. Wenn der Ausdruck einen Pfad über mehrere Knoten hinweg beschreibt, werden die Elemente durch Schrägstriche getrennt:

```
dirlist/directory/file
```

Lokalisierungsschritte

Jedes dieser Elemente wird allgemein als Lokalisierungsschritt bezeichnet. Die eben gezeigte und häufig verwendete Darstellung durch einen Knotennamen ist eine verkürzte Form. Tatsächlich kann jeder Schritt aus drei Teilen bestehen:

1. Achsenbezeichner
2. Knotentest
3. Prädikate

Der Achsenbezeichner modifiziert die Auswahl des Knotens auf der Grundlage seiner Position im Baum. Als Trennzeichen zwischen dem Achsenbezeichner und dem nächsten Teil des Ausdrucks werden zwei Doppelpunkte geschrieben »::«. Dies wird im nächsten Abschnitt noch weiter erläutert. Der Knotentest beschreibt den Knoten selbst, beispielsweise eine direkte Auswahl durch Nennung des Tagnamens. Die Prädikate stehen in eckigen Klammern und werden meist zur Auswahl von Attributen verwendet.

Insgesamt ergibt sich beispielsweise folgender Ausdruck:

```
child::directory[attribute::hasfiles='true']
```

`child::` ist der Achsenbezeichner, hier wird also beginnend von der aktuellen Position das nächste Kindelement gesucht. Dann wird das Element selbst benannt: `directory`. Es wird also das nächste Kindelement mit dem Namen »directory« gesucht. Das Prädikat schränkt die Suche weiter ein; hier auf das Vorhandensein eines Attributes `hasfile` mit dem Parameter `'true'`³.

Um solche Ausdrücke nun entwickeln zu können, ist in erster Linie eine Kenntnis der Achsenbezeichner notwendig.

Die XPath-Achsenbezeichner

Achsenbezeichner

Achsenbezeichner können einen oder mehrere Knoten auswählen. Die konkrete Auswahl hängt vom aktuellen Knoten ab. Dies ist zutreffend, wenn ein Dokument sequenziell durchlaufen wird. Die folgende Tabelle zeigt alle Achsenbezeichner für Elemente auf einen Blick. Elemente sind Tagnamen, also keine Attribute und keine Namensraumbezeichnungen:

³ Die einfachen Anführungszeichen innerhalb der doppelten gehören bei XPath dazu.

Achsenname	Such- richtung	Hinweise
self	–	Aktueller Knoten
child	vor	Kinder des Knotens
parent	–	Eltern des Knotens
descendant	vor	Alle Nachfahren (Kinder und Kindeskindern)
descendant-or-self	vor	Alle Nachfahren und der Knoten selbst
ancestor	rück	Alle Vorfahren (Eltern und deren Eltern)
ancestor-or-self	rück	Alle Vorfahren und der Knoten selbst
following	vor	Alle folgenden Knoten im Dokument, die nicht direkte Nachfahren sind
following-sibling	vor	Alle folgenden Geschwister
preceding	rück	Alle vorhergehenden Knoten, die nicht Eltern sind
preceding-sibling	rück	Alle vorhergehenden Geschwister

Tabelle 5.15: Achsenbezeichner für Element-Knoten

Neben den Achsenbezeichnern für Elemente gibt es noch zwei spezielle: `attribute` zur Auswahl von Attributen und `namespace` zur Lokalisierung von Namensräumen.

```
<?xml version="1.0" encoding="utf-8" ?>
- <A>
  <B />
  - <C>
    <D />
    - <E>
      <H />
      <I />
    </E>
    <F />
    <G />
  </C>
  <X />
- <Y>
  <Z />
</Y>
</A>
```

Abbildung 5.31: Eine Muster-XML-Datei zum Testen von XPath-Ausdrücken (`data/axischeck.xml`)

Wirkung der Achsenbezeichner

Es bietet sich an dieser Stelle an, die Wirkung der Achsenbezeichner mit einem Testprogramm zu lernen. Damit alle erdenklichen Kombinationen auch getestet werden können, wird eine XML-Datei entworfen, die entsprechende Achsen auch aufweist (siehe Abbildung 5.31).

Diese Datei dient im folgenden Beispiel als Basis für die ersten Programmiersuche mit .NET.

5.8.2 Mit .NET XPath-Ausdrücke verarbeiten

Die bereits besprochene Klasse `XmlTextReader` kann mit XPath-Ausdrücken nicht umgehen. Die hohe Zugriffsgeschwindigkeit geht mit eingeschränkter Funktionalität einher. Universeller ist die Klasse `XmlDocument`, von der weitere Klassen abgeleitet werden können, die Zugriff auf XPath und die Resultate einer Abfrage erlauben.

Zugriff auf XML mit XmlDocument

`XmlDocument` erlaubt den Zugriff auf XML über dessen DOM-Struktur. Damit das Dokument entsprechend bereit gestellt wird, schreiben Sie im Code folgende Zeilen:

```
Dim xDoc As XmlDocument = New XmlDocument ()
xDoc.Load(Server.MapPath("data/axischeck.xml"))
```

Die Datei *axischeck.xml* ist die in Abbildung 5.31 gezeigte XML-Datei. Nun kann mit der Methode `SelectNodes` eine Liste von Knoten mit Hilfe eines XPath-Ausdrucks ermittelt werden. Ist der Ausdruck erfolglos, wird `Nothing` zurückgegeben. Die folgende Methode aus *AxisCheck.aspx.vb* zeigt die Ausgabe einer Knotenhierarchie als kommaseparierte Liste:

```
Private Sub.GetAxis(ByVal xNodes As XmlNodeList,   
                  ByVal currentLabel As Label)   
    Dim iNodes As Integer = xNodes.Count   
    Dim i As Integer = 0   
    currentLabel.Text += iNodes.ToString() + " Knoten: "   
    Dim sa() As String = New String(iNodes) {}   
    Dim xNode As XmlNode   
    For Each xNode In xNodes   
        i = i + 1   
        sa.SetValue(xNode.LocalName, i)
```

```

Next
currentLabel.Text += String.Join(", ", sa)
End Sub

```

Eine solche Liste ist ein Objekt der Klasse `XmlNodeList`. Hier wird zuerst die Anzahl der Elemente mit `Count` ermittelt:

```
Dim iNodes As Integer = xNodes.Count
```

Die Methode zeigt die Ausdrücke jeweils gezielt in einem bestimmten `<asp:Label>`-Steuerelement auf der Seite an. Die Elemente werden dann mit einer *For Each*-Schleife durchlaufen und einem Array hinzugefügt:

```
sa.SetValue(xNode.LocalName, i)
```

Dieses Array wird dann mit `Join` in die Liste umgewandelt und dem `Label`-Steuerelement hinzugefügt. Nun ist natürlich noch der Aufruf interessant, der `SelectNodes` anwendet und die eben beschriebene Methode aufruft:

```
GetAxis(xDoc.SelectNodes(sPath + "child:*"), child)
```

In der `WebForm` ist die Ausgabe in eine Tabelle eingebettet, die die `Label`-Steuerelemente entsprechend enthält (vollständig finden Sie diese in der Datei `AxisCheck.aspx`):

```
<asp:Label Runat="server" ID="child"/>
```

Führt man diese Schritte für alle XPath-Standardanweisungen aus, bezogen auf den Knoten »C« als Startknoten, ergibt sich folgende Abbildung:

Achsenauswahl mit XPath

self	1 Knoten: C
parent	1 Knoten: A
child	4 Knoten: D, E, F, G
descendant	6 Knoten: D, E, H, I, F, G
descendant-or-self	7 Knoten: C, D, E, H, I, F, G
ancestor	1 Knoten: A
ancestor-or-self	2 Knoten: A, C
preceding	1 Knoten: B
preceding-sibling	1 Knoten: B
following	3 Knoten: X, Y, Z
following-sibling	2 Knoten: X, Y

Abbildung 5.32: Knotenmengen, die verschiedene XPath-Ausdrücke zurückgeben

Das Feld `sPath` enthielt die Zeichenfolge `»/A/C«`, also einen einfachen XPath-Ausdruck, der zum Element `<C>` hinführt, damit dieses als Ausgangspunkt dienen kann. Der Ausdruck selektiert lediglich die Achse (`achsenbezeichner::*`), wobei das `*` zur Auswahl aller Elemente der Achse führt. In der Praxis könnten sich hier Elementnamen anschließen, um die Auswahl einzuschränken, sowie eines oder mehrere Attribute, die auftreten oder bestimmte Parameter aufweisen müssen.

Wichtige Methoden und Eigenschaften von XmlDocument

`XmlDocument` erlaubt den Zugriff auf das gesamte Dokument. Die enthaltenen Daten können mit Hilfe der `Create`-Methode zur Laufzeit erzeugt werden. Liegen die Daten bereits vor – wie im letzten Beispiel – werden diese mit `Load` auf einer Datei oder mit `LoadXml` aus einer Zeichenkette in das Objekt kopiert. Das Lesen der Daten kann dann mit zwei Methoden erfolgen:

► `SelectNodes`

Diese Methode gibt Knotenlisten aufgrund eines XPath-Ausdrucks zurück. Der Rückgabetyt ist `XmlNodeList`.

► `SelectSingleNode`

Diese Methode gibt nur den ersten Knoten der durch XPath ermittelten Knotenmenge zurück. Der Rückgabetyt für einen Knoten ist `XmlNode`.

Alternativ zu XPath kann die Auswahl eines Elements auch über die ID erfolgen – wenn die Daten entsprechend enthalten sind. Dazu nutzen Sie die Methode `GetElementById`. Sind die Namen ein sicheres Auswahlkriterium, kann `GetElementsByTagName` verwendet werden. Diese Methode gibt ein Objekt vom Typ `XmlNodeList` zurück.

Navigieren in einem Dokument ohne XPath

Da `XmlDocument` bereits XPath kennt, sind einige elementare Operationen bereits durch fertige Eigenschaften hinterlegt. So ist `»child::*«` äquivalent zur Eigenschaft `ChildNodes`. Wenn Sie diese Eigenschaften direkt nach dem Laden des Dokuments anwenden, startet die Auswahl immer an der Wurzel. Sie können jedoch in je-

der Situation Knotenlisten mit `ChildNodes` abtrennen und diese dann mit denselben Eigenschaften durchsuchen. Dies liegt an der Vererbung der XML-Klassen. Denn als Basis für Knotenlisten dient die Klasse `XmlNodeList`, deren Instanz mehrere Knoten enthalten kann und `XmlNode`, die Repräsentanz eines Knotens. Die letzte Klasse liefert auch die Navigationseigenschaften an `XmlDocument`.

Neben `ChildNodes` kann mit `FirstChild` und `LastChild` sowie `NextSibling` und `PreviousSibling` operiert werden. Unter Umständen sparen Sie damit XPath-Anweisungen.

Das folgende Beispiel verwendet `ChildNodes`, um ein Dokument rekursiv zu durchlaufen.

```
Public Class XmlDocumentNodes
    Inherits System.Web.UI.Page
    Protected name As Label
    Shared indent As Integer

    Private Sub GetNode(ByVal xNodes As XmlNodeList)
        Dim xn As XmlNode
        For Each xn In xNodes
            indent = indent + 1
            name.Text += (xn.Name).PadLeft(indent * 3, " " &
                Convert.ToChar(&H9A0)) + "<br />"
            If (xn.HasChildNodes) Then
                GetNode(xn.ChildNodes)
            End If
            indent = indent - 1
        Next
    End Sub

    Private Sub Page_Load(ByVal sender As Object, " "
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim xDoc As XmlDocument = New XmlDocument()
        xDoc.Load(Server.MapPath("data/axischeck.xml"))
        indent = 0
        GetNode(xDoc.ChildNodes)
    End Sub

End Class
```

Listing 5.8: Rekursive Ausgabe des gesamten XML-Dokuments
(*XmlDocumentNodes.aspx.vb*)

Um den rekursiven Aufruf zu steuern, wird die Eigenschaft `HasChildNodes` für jeden einzelnen Knoten untersucht. Außerdem wird für die übersichtliche Ausgabe die Einrückung gesteuert, in-

dem entsprechend dem Inhalt des statischen Feldes `indent` eine Anzahl Leerzeichen geschrieben wird. Der hexadezimale Code »&HA0« entspricht der HTML-Entität ` `; im Unicode-Zeichenraum.

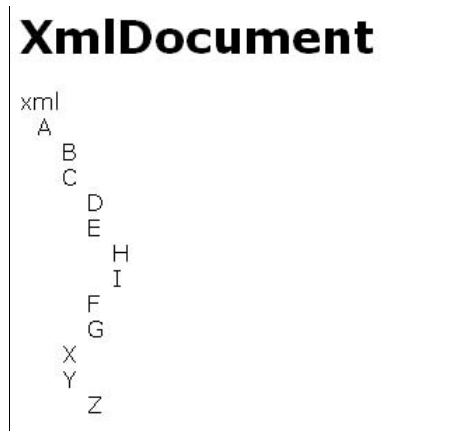


Abbildung 5.33: Baumdarstellung eines XML-Dokuments

5.9 Daten transformieren mit XSLT

Im ersten Teil des XML-Abschnitts wurde bereits XSLT verwendet. XSLT (XML Stylesheet Language for Transformation) dient der Umwandlung von XML in andere textbasierte Darstellungen von Daten – natürlich auch wieder in XML. Der Ansatz, die Transformation durch irgendein geeignetes Programm vornehmen zu lassen, ist dank der guten Standardisierung praktikabel. Wenn Sie sich jedoch darauf beim Nutzer nicht verlassen möchten, kann mit .NET jede Transformation auf dem Server ausgeführt werden. Diese Methode hat weitere Vorteile. So kann die Transformationsanweisung selbst durch Parameter modifiziert werden, die gegebenenfalls erst zur Laufzeit entstehen.

Dieser Abschnitt zeigt, wie Sie mit XSLT in .NET transformieren und die erzeugten Daten in jedem beliebigen Browser zur Anzeige bringen.

5.9.1 Eine kompakte Einführung in XSLT

XSLT ist eine so genannte funktionale Programmiersprache. Das Prinzip unterscheidet sich grundlegend von den imperativen oder objektorientierten Sprachen, wie beispielsweise VB.NET. Der Programmfluss selbst wird in erster Linie durch Automatismen initiiert, in zweiter Linie dann durch Regeln. Regeln definieren Sie, um bestimmte Effekte beim Auftreten von bestimmten Daten zu erreichen. Vorteil derartiger Systeme ist die weitgehende – bei XSLT per Definition die vollkommene – Befreiung von Seiteneffekten. Wenn eine Regel gilt, dann wird diese und nur diese ausgeführt und dies in immer der gleichen Art und Weise. Dazu gehört auch, dass Variablen zwar verfügbar sind, beispielsweise um einer Regel einen Wert zu übergeben, ihren Inhalt aber nachträglich nicht ändern können. Sie verhalten sich also eher wie Konstanten in imperativen Sprachen. Nachträgliche Änderungen könnten Seiteneffekte erzeugen, was nicht erlaubt ist.

Eine funktionale Sprache

Dennoch kann man damit erstaunlich effektiv programmieren und verblüffende Resultate erzielen. Nicht immer ist XSLT die perfekte Sprache. Richtig leistungsfähig wird sie erst in Kombination mit einer modernen objektorientierten Sprache, die hinreichende imperative Merkmale aufweist. Es ist nahliegend, Transformation und Programm mit .NET in einen Kontext zu überführen. Zuvor sind jedoch wenigstens elementare Kenntnisse von XSLT notwendig.

Verblüffende Resultate

Die Basisregeln in XSLT

XSLT basiert auf XML, weshalb jede Datei durch die entsprechende Deklaration eingeleitet wird. Dann folgt das Wurzelement `stylesheet`. Das W3C empfiehlt als Standardnamensraum `xsl`; diese Angabe ist aber im Prinzip freiwillig. Visual Studio .NET erstellt neue XSLT-Dateien ohne eigenen Namensraum. Solange Sie diese nur in eigenen Projekten verwenden und nicht weitergeben, mag das akzeptabel sein. Es ist jedoch empfehlenswert, generell den Standardnamensraum zu verwenden. Daraus ergibt sich folgendes Grundgerüst für XSLT:

Standard-namensraum

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Namensraumalias Durch die Erweiterung des Attributes `xmlns` wird der Namensraumalias festgelegt.



Namensräume in XML sollen hier nicht ausführlich diskutiert werden. Sie finden dazu in der XML-Literatur entsprechende Informationen. Generell kapseln Namensräume die Elementnamen, sodass Namenskonflikte vermieden werden. Der Namensraum selbst wird durch einen eindeutigen URI bezeichnet. Bei XSLT heißt diese »http://www.w3.org/1999/XSL/Transform«. Diese Adressen ähneln einem URL im Internet und sind meist auch vorhanden – das ist aber keine Voraussetzung für den Einsatz. Im Dokument selbst wird dieser Namensraum durch einen Alias ersetzt, der frei wählbar ist. In XSLT ist der Alias »xsl:« üblich.

Einfache Vorlagen Zwischen den Wurzelementen wird nun das Regelwerk aufgebaut. Eine zentrale Rolle spielt das Element `<xsl:template>`. Templates (dt. Vorlagen) bilden die Stufen der eigentlichen Transformation. Dabei gibt es zwei Arten von Templates. Zum einen können sie durch eine XPath-Anweisung in ihrer Zuständigkeit programmiert werden. Die folgende Regel zeigt, wie jedes Element `<name>` zu einer Ausgabe im Ausgabedatenstrom führt:

```
<xsl:template match="name">
  <h1>NAME</h1>
</xsl:template>
```

Eine andere Methode ist der Aufruf benannter Vorlagen, dazu später mehr. Der Inhalt des Elements findet hier freilich noch keine Berücksichtigung. Text kann, wie gezeigt, direkt ausgegeben werden. Beachten Sie dabei, dass es sich auch hier um wohlgeformtes XML handeln muss; HTML muss also gegebenenfalls den Regeln von XHTML 1.0 entsprechend modifiziert werden.

Benannte Vorlagen Wenn Sie eine Vorlage mit `<xsl:template select="regelname">` benennen, können Sie diese folgendermaßen aufrufen:

```
<xsl:call-template name="regelname"/>
```

Text Soll explizit Text ausgegeben werden, der mit dem Editor nicht darstellbar ist, muss das `<xsl:text>`-Element eingesetzt werden. Das ist eigentlich – nach der Spezifikation – immer notwendig. Die direkte Angabe von Text oder Tags ist eine Vereinfachung. Beachten Sie, dass im Gegensatz zu VB.NET die Angabe hexadezimaler Codes in XSLT mit dem Präfix »0x« erklärt wird.

```
<xsl:template match="name">
  Hier folgt ein Zeilenumbruch:<xsl:text>0x0A</xsl:text>
</xsl:template>
```

Wo Text ist, sind Kommentare nicht weit. Diese entsprechen, XML-konform, denen aus HTML bekannten und werden nicht in den Ausgabedatenstrom übernommen:

Kommentare

```
<!-- Ein Kommentar in XSLT sieht aus wie in HTML -->
```

Vorlagen werden meist verschachtelt angewendet. Das folgende Beispiel zeigt das Grundgerüst einer HTML-Seite, wie sie mit XSLT erzeugt wird:

**Vorlagen
praktisch
verarbeiten**

```
<xsl:template match="/">
  <html>
    <body>
      <xsl:apply-templates />
    </body>
  </html>
</xsl:template>
```

Zuerst erkennt der XSLT-Prozessor hier, dass die Vorlage das Wurzelement der XML-Quelle verarbeitet. Dann wird das Grundgerüst der HTML-Seite erstellt. Innerhalb des `<body>`-Tags wird versucht, alle übrigen Elemente durch Aufruf der passenden Vorlagen zu verarbeiten. Das `<xsl:apply-template>` keine Parameter hat, ist ein spezieller Fall. Er setzt voraus, dass alle Elemente irgendwo auf eine passende Vorlage stoßen, wobei der Prozessor den besten Treffer auswählt und diesen – und nur diesen – ausführt.

Allerdings besitzt der Prozessor eine Fallback-Funktion. Wenn kein Template zutrifft, wird der Inhalt des aktuellen Tags genommen und als gültiger Ausgabewert betrachtet. Voraussetzung ist aber, dass wenigstens an einer Stelle `<xsl:apply-template>` steht, um die Ausgabe auszulösen.

Sollen Inhalte von Tags gezielt ausgegeben werden, was sicher der häufigste Weg ist, findet `<xsl:value-of>` Verwendung. Das Attribut `select` wählt den Inhalt des durch eine XPath-Anweisung ermittelten Knotens und die gesamte Anweisung gibt diesen als Zeichenkette aus:

```
<xsl:template match="B">
  <xsl:value-of select="."/>
</xsl:template>
```

Attribute verwenden

Beim Einsatz innerhalb einer Vorlage bezieht sich der Pfad, den `select` akzeptiert, auf den übergebenen Knoten, ist also relativ. Sie können aber absolute Angaben verwenden. Der allein stehende Punkt reflektiert in XPath den aktuellen Knoten, im Beispiel also den Inhalt des Tags `<B>`. Auf eben diesem Wege werden auch Attribute gelesen. Das folgende Beispiel sucht nach Elementen vom Typ `<a>` und gibt den Inhalt des Attributes `href` aus:

```
<xsl:template match="a">
  <xsl:value-of select="@href"/>
</xsl:template>
```

Der direkte Zugriff mit einer absoluten XPath-Anweisung wäre `a/@href`. Sie können auf den Parameter eines Attributes auch direkt zugreifen. Ein `<a href>`-Tag wird folgendermaßen in `<img src>` transformiert:

```
<xsl:template match="a">
  
</xsl:template>
```

Die Schreibweise mit den geschweiften Klammern ist immer dann angebracht, wenn der Einsatz eines Tags aufgrund der Syntax nicht möglich ist. Andererseits ist es mit `<xsl:element>` und `<xsl:attribute>` möglich, beliebige Tags indirekt zu erzeugen.

Mit XSLT programmieren

Bei XSLT spricht man von einer funktionalen Programmiersprache. Zum Programmieren gehören jedoch nicht nur Regeln, wie sie in XSLT durch die Vorlagen gebildet werden, sondern auch Programmanweisungen.

Verzweigung

Zuerst eine einfache Verzweigung mit `<xsl:if>`:

```
<xsl:if test="@directory='hasfiles'">
  Dieses Verzeichnis enthält Dateien
</xsl:if>
```

Mehrfachverzweigung

Der Test kann verschiedene Operatoren und Funktionen verwenden, um Knoten nach allerhand Kriterien zu untersuchen. Eine *Else*-Anweisung gibt es übrigens nicht, hierfür ist die Mehrfachverzweigung `<xsl:choose>` gedacht.

```
<xsl:choose>
  <xsl:when test="attribute='archive'">Archiv</xsl:when>
```

```

<xsl:when test="attribute='compressed'">Compressed</xsl:when>
<xsl:when test="attribute='hidden'">Hidden</xsl:when>
<xsl:otherwise>
  Unknown
</xsl:otherwise>
</xsl:choose>

```

Wollen Sie Listen von bestimmten Tags an einer Stelle ausgeben, ist `<xsl:for-each>` sehr praktisch. Schleifen im Sinne imperativer Programmierung gibt es jedoch nicht, weil veränderliche Zustände nicht erlaubt sind:

Schleifen

```

<xsl:for-each select="name">
  <a href="{.}"><xsl:value-of select="."/></a><br/>
</xsl:for-each>

```

Die `<xsl:for-each>`-Anweisung gibt, wie `<xsl:template>` auch, jeweils einen aktuellen Knoten für jedes Element aus, das gefunden wurde. Deshalb funktioniert auch hier der verkürzte Zugriff auf den Inhalt mit dem Punkt-Alias.

Von Interesse ist oft auch eine Sortiermöglichkeit. Sie können dazu innerhalb einer Schleife mit `<xsl:sort>` eine Anweisung platzieren, die das zuverlässig erledigt:

Sortieren

```

<xsl:for-each select="name">
  <xsl:sort select="." order="descending"/>
  <a href="{.}"><xsl:value-of select="."/></a><br/>
</xsl:for-each>

```

Das Element `<xsl:sort>` ist übrigens auch in `<xsl:apply-templates>` anwendbar. Es versteht freilich einige Attribute mehr, mit denen die Steuerung der Sortierung erfolgen kann.

Was Sie in XSLT nicht finden, sind die Anweisungen »for« und »while«. Beide benötigen Variablen, deren Zustand sich ändert. Das ist in funktionalen Sprachen nicht möglich, weswegen die Anweisungen nicht sinnvoll sind. Komplexere Schleifen werden durch Rekursion (Selbstaufzuruf einer Vorlage) und Parameter ermöglicht. Oft ist dies kompakter als vergleichsweise imperative Programme.

Variablen sind dennoch verwendbar, sie verhalten sich aber ähnlich den Konstanten in anderen Sprachen. Sie können Werte, Knoten oder Knotenbäume speichern:

Variablen

```

<xsl:variable name="fieldata" select="attribute"/>

```

Variablen können natürlich auch komplexere Inhalte aufnehmen:

```
<xsl:variable name="alldata">
  <xsl:if test="position()=last()">
    TRUE
  </xsl:if>
  <xsl:if test="position()=1">
    FALSE
  </xsl:if>
</xsl:variable>
```

Sie sehen im letzten Beispiel auch die Verwendung von XPath-Funktionen. Variablen gelten nur innerhalb der Vorlage oder der Schleife, in der sie definiert wurden. Dieses Verhalten ist nicht modifizierbar, das heißt, es gibt keine Modifikatoren wie »public« oder »private«.

Ähnlich wie Variablen werden auch Parameter verwendet. Damit können Sie einer Vorlage verschiedene Werte übergeben, damit diese sich je nach Art des Aufrufes unterschiedlich verhält. Der Aufruf sieht folgendermaßen aus (am Beispiel einer benannten Vorlage):

```
<xsl:call-template name="show.files">
  <xsl:with-param name="handler">no</xsl:with-param>
</xsl:call-template>
```

Innerhalb der Vorlage werden die übergebenen Parameter dann so verwendet:

```
<xsl:template name="show.files">
  <xsl:param name="handler" select=""/>
  ...
```

Das `select`-Attribut in `<xsl:param>` bestimmt einen Standardwert, wenn der Parameter nicht übergeben wurde.

XSLT im Alltag

Nummern	XSLT verfügt über einige komplexere Anweisungen, die für größere Projekte von Bedeutung sind. Dazu gehört <code><xsl:number></code> zum Erzeugen fortlaufender Nummern oder Buchstabenfolgen.
Knoten kopieren	Oft ist auch der Umgang mit ganzen Knoten notwendig, statt des Textinhalts des Knotens. Dann findet <code><xsl:copy-of></code> Verwendung. Sollen Knoten und Attribute kopiert werden, können mehrere Anweisungen mit <code><xsl:copy></code> zusammengefasst werden.

Das folgende Beispiel kopiert ein XML-Dokument vollständig in ein anderes:

```
<xsl:template match="*">
  <xsl:copy>
    <xsl:copy-of select="@*">
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Da sich hier die Frage stellt, warum ein Dokument ausgerechnet mit XSLT unverändert kopiert werden sollte, ist ein Blick auf `<xsl:output>` angebracht. Mit dieser Anweisung, die immer am Anfang des Dokumentes steht, kann die Kodierung des Ausgabestromes gesteuert werden. Wenn Ihr XML-Dokument UTF-8 kodiert ist, können Sie es mit der Kopiervorlage des letzten Beispiels leicht in ein anderes Format bringen, nebenbei auch ins HTML 4.0-Format:

**Ausgabe-
steuerung**

```
<xsl:output encoding="ISO-8859-1" method="html"/>
```

Für größere Projekte können Sie XSLT-Dateien mit Hilfe der Anweisungen `<xsl:output>` und `<xsl:include>` importieren. Mit Hilfe von `<xsl:include>` kann ein Dokument so eingefügt werden, als wäre der Inhalt an dieser Stelle geschrieben worden. Der Import hat eine geringere Priorität; stehen Regeln miteinander im Konflikt, unterliegen die importierten.

XSLT-Funktionen

Da Transformationen oft in Abhängigkeit von konkreten Daten ablaufen, stellt XSLT einige elementare Funktionen zur Verfügung, die vor allem in `select`- und `test`-Attributen Anwendung finden.

5.9.2 XSLT-Klassen in .NET

Am Anfang wurde bereits gezeigt, wie der Internet Explorer die Transformation von XML mit XSLT ausführt. Wenn Sie sich nicht auf den Browser verlassen können oder wollen, muss die Umwandlung auf dem Server stattfinden. Es ist nahe liegend, dies mit .NET zu erledigen. Natürlich ist auch der passende Namensraum verfügbar: `System.Xml.Xsl`. Der in der darin enthaltenen Klasse `XslTransform` verfügbare Prozessor realisiert den Standard XSLT 1.0 vollständig.

Transformationen ausführen

Der kürzeste Weg, die Transformation direkt auszuführen, sieht folgendermaßen aus:

```
Dim xTrans As XslTransform = new XslTransform ()
xTrans.Load("stylesheet.xslt")
xTrans.Transform("quelle.xml", "ziel.xml")
```

Da nicht immer dateiorientiert gearbeitet wird, kennt die Methode Load verschiedene Überladungen, die neben Dateinamen auch TextWriter oder TextReader, Stream-Objekte und URL-Namen akzeptiert.

XSLT praktisch anwenden

Für den praktischen Einsatz von XSLT in .NET ist eine kleine Einschränkung zu beachten. Es ist verhältnismäßig schwer, on-the-fly erzeugte XML-Daten sofort zu transformieren. Das Senden einer XSLT-Datei an den Browser, um die Transformation dort vornehmen zu lassen, ist deutlich einfacher. Liegen die XML-Daten dagegen fertig vor, ist die Transformation auf dem Server sehr einfach. Betrachten Sie folgende XML-Datei, die die Basis eines Nachrichtendienstes darstellt:

```
<?xml version="1.0" encoding="utf-8"?>
<news xmlns="http://tempuri.org/news.xsd">
  <newscontent date="2002-03-26T00:00:00.0000000+01:00">
    <title>SharpTemple veröffentlicht</title>
    <text>Heute wurde die erste öffentliche Version der
      Entwicklungsumgebung SharpTemple vorgestellt.
    </text>
  </newscontent>
  <newscontent date="2002-06-20T00:00:00.0000000+02:00">
    <title>ASP-Kongress</title>
    <text>Burghausen beginnt, erstmals auch mit einer
      eigenen Veranstaltung zu ASP.NET
    </text>
  </newscontent>
  <newscontent date="2001-07-01T00:00:00.0000000+02:00">
    <title>QualityHosting</title>
    <text>preiswert hosten auf den W2000-Systemen
      von Qualityhosting.de
    </text>
  </newscontent>
</news>
```

Listing 5.9: Täglich frische Nachrichten aus einer XML-Datei (data/news.xml)

Diese Daten sollen nun an verschiedenen Stellen der Website platziert werden. Je nach den entsprechenden Erfordernissen muss die Aufbereitung verschiedenartig erfolgen – ideal für den Einsatz von XSLT. Das folgende Programm wählt die XSLT-Datei aufgrund des Namens der aufrufenden *aspx*-Datei aus. Damit der Einbau problemlos erfolgen kann, wird der Nachrichtenblock Teil eines Benutzer-Steuerelements. Zuerst eine Demonstration der Verwendung:

```
<%@ Register TagPrefix="uc" TagName="news"
    Src="XsltNewsControl.ascx"%>
<html>
  <head>
    <title>XSLT NewsSite</title>
  </head>
  <body>
    <h1>Hier kommen die News</h1>
    <uc:news runat="server" ID="News1" layout="mainpage"/>
  </body>
</html>
```

Listing 5.10: So wird das Benutzer-Steuerelement verwendet (XmlNewsXslt.aspx)

Das Benutzer-Steuerelement soll nun die XML-Datei lesen, die aufrufende Datei erkennen (hier ist dies *XmlNewsXslt.aspx*) und danach die passende XSLT-Datei ermitteln (*mainpage.xslt* – gesteuert durch das Attribut *layout*) sowie die transformierten Daten ausgeben. Vorteilhaft an diesem Szenario ist die externe Konfigurationsmöglichkeit. So reicht es aus, die XSLT-Daten zu ändern, um die Darstellung anzupassen. Ebenso steuert die XML-Datei den Inhalt. Änderungen am Programm – das in der Praxis weitaus komplexer ausfallen kann – sind nicht mehr notwendig.

Das Benutzer-Steuerelement selbst besteht lediglich aus einem Label-Element und dem Aufruf der Code-Datei:

Das Benutzer-Steuerelement

```
<%@ Control Language="vb" AutoEventWireup="false"
    Codebehind="XsltNewsControl.ascx.vb"
    Inherits="dasbuch.XsltNewsControl"%>
<asp:Label Runat="server" ID="NewsControl"/>
```

In dieser Code-Datei wird nun die eigentliche Arbeit erledigt, weitgehend von außen steuerbar.

```
Public MustInherit Class XsltNewsControl
    Inherits System.Web.UI.UserControl
    Private _layout As String = String.Empty
```

```

Public NewsControl As Label

Public Property Layout() As String
    Get
        Return _layout
    End Get
    Set(ByVal Value As String)
        _layout = value
    End Set
End Property

Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles MyBase.Load
    Dim xTrans As XsltTransform = New XsltTransform()
    Dim xDoc As XmlDocument = New XmlDocument()
    Dim xWrite As MemoryStream = New MemoryStream()
    xTrans.Load(Server.MapPath(String.Format(
        ("data/{0}.xslt", 
        Layout)))
    xDoc.Load(Server.MapPath("data/news.xml"))
    xTrans.Transform(xDoc, Nothing, xWrite)
    xWrite.Seek(0, SeekOrigin.Begin)
    NewsControl.Text = New StreamReader(xWrite).ReadToEnd()
End Sub

End Class

```

Listing 5.11: XML lesen, transformieren und ausgeben (XsltNewsControl.ascx.vb)

Das Benutzer-Steuerelement kennt ein Attribut `layout`. Dies wird über die Eigenschaft `Layout` ausgewertet. Beachten Sie hier, dass keinerlei Fehlerüberwachung stattfindet. Existiert die geforderte XSLT-Datei nicht, wird eine Ausnahme vom Typ `FileNotFoundException` geworfen. In der Praxis sollten Sie derartige Fehler natürlich gezielt abfangen, beispielsweise durch Nutzung der Methode `FileExists`.

Die eigentliche Arbeit wird in der `Page_Load`-Methode beim Aufbau der Seite erledigt. Zuerst wird ein `XsltTransform`-Objekt erzeugt. Als Datenquelle dient ein `XmlDocument`-Objekt. Sie können als Datenquelle entweder die Datei direkt angeben oder eine navigierbare Darstellung des Dokuments angeben. Letzteres ist hier angebracht, weil die Ausgabe nicht als Datei erfolgen soll, sondern als Zeichenkette an das `Label`-Steuerelement.

Um die Daten auf diesem Wege umzuleiten, eignen sich generell die Stream-Klassen. Eine spezialisierte ist `MemoryStream`, mit der ein Byte-Stream in den Speicher geschrieben wird.

```
Dim xWrite As MemoryStream = New MemoryStream()
```

Dann wird die XSLT-Datei, basierend auf der Auswertung der Eigenschaft `Layout`, geladen. Der korrekte physische Pfad wird mit `Server.MapPath` ermittelt:

```
xTrans.Load (Server.MapPath (
    (String.Format("data/{0}.xslt", Layout))
))
```

Im nächsten Schritt wird das XML-Dokument geladen:

```
xDoc.Load (Server.MapPath ("data/news.xml"))
```

Jetzt kann die Transformation bereits stattfinden. Der mittlere Parameter bleibt `Nothing`. An dieser Stelle lassen sich Parameter an die XSLT-Datei übergeben. Dazu finden Sie Informationen im nächsten Abschnitt. Als Ziel der Transformation wird der `MemoryStream` angegeben; die Daten landen also im Speicher.

```
xTrans.Transform (xDoc, Nothing, xWrite)
```

Die nächsten beiden Zeilen holen die Daten aus dem Speicher und laden sie als Zeichenkette in das `Label`-Steuerelement. Da der Datenzeiger nach dem Schreibvorgang am Ende des Datenstromes steht, muss er zuerst auf den Anfang – Position 0 – gesetzt werden:

```
xWrite.Seek (0, SeekOrigin.Begin)
```

Nun wird ein `StreamReader` verwendet, um die Daten komplett (`ReadToEnd`) auszulesen:

```
NewsControl.Text = New StreamReader(xWrite).ReadToEnd()
```

Damit ist der Vorgang bereits abgeschlossen, die Nachrichten erscheinen nun in der gewünschten Form im Browser. Voraussetzung ist natürlich eine passende XSLT-Datei:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>
</xsl:stylesheet>
```

```

</xsl:template>
<xsl:template match="newscontent">
  <hr noshade="noshade"/>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="title">
  <h3><xsl:value-of select="."/></h3>
</xsl:template>
<xsl:template match="text">
  <div style="background-color:silver">
    <xsl:value-of select="."/>
  </div>
</xsl:template>
</xsl:stylesheet>

```

*Listing 5.12: XSLT-Datei zur Darstellung der Nachrichten aus news.xml
(mainpage.xslt im Ordner data)*

Ausgabe- formatierung

Auf eine Erläuterung soll hier verzichtet werden, da der Code keinerlei Besonderheit enthält. Jedes Element wird entsprechend dem Bedarf formatiert; die Ausgabe erfolgt als HTML. Der XSLT-Prozessor wird die Ausgabe übrigens bei der Angabe von `method="html"` konform zu HTML 4.0 formatieren. Aus der XHTML-Darstellung `<hr noshade="noshade"/>` wird deshalb in der Ausgabe `<hr noshade>`. Möchten Sie XHTML 1.0 erzeugen, schreiben Sie in der XSLT-Datei folgende Formatierung:

```
<xsl:output method="xml" omit-xml-declaration="yes"/>
```

Der zweite Parameter unterdrückt die XML-Deklaration im Ausgabedatenstrom. Das ist dann notwendig, wenn – wie im Beispiel – lediglich ein Teil der HTML-Seite auf diesem Wege erzeugt wird.

Die Modifikation der Ausgabebedingungen über XML und XSLT ist einfach und meist ausreichend. Wenn jedoch innerhalb eines Programms Situationen entstehen, die eine Änderung erforderlich machen, ist dieser Weg nicht flexibel genug. Natürlich könnten Sie für jeden Fall eine eigene XSLT-Datei vorsehen. Ändert sich an der Arbeitsweise jedoch nur wenig, müssten Sie mehrere Dateien parallel pflegen. Besser wäre es, wenn man an eine XSLT-Datei Parameter übergeben könnte, die dann zu einer Modifikation des Verhaltens führen. Der nächste Abschnitt beschreibt dieses Verfahren.

Hier kommen die News

SharpTemple veröffentlicht

Heute wurde die erste öffentliche Version der Entwicklungsumgebung SharpTemple vorgestellt.

ASP-Kongress

Burghausen beginnt, erstmals auch mit einer eigenen Veranstaltung zu ASP.NET

QualityHosting

preiswert hosten auf den W2000-Systemen von Qualityhosting.de

Abbildung 5.34: Ausgabe der News mit Hilfe von XSLT

XSLT mit Parametern verwenden

Am Anfang des Kapitels wurde gezeigt, wie eine Verzeichnis- und Dateiliste in XML erstellt werden kann. Es wäre hilfreich, wenn zusätzliche Informationen zu den Dateien ermittelt werden könnten. Damit die Datenmenge dabei nicht unnötig aufgebläht wird, sollen nur für eine spezielle Datei ausgewählte Informationen gelesen werden.

Dabei werden die bisher bekannten Techniken intensiv genutzt. Der XML-Datenstrom wird dynamisch aufgebaut und im Speicher mit XSLT transformiert. Die ausgewählte Datei wird mit XSLT-Parametern an das Stylesheet übergeben und dort erfolgt die Modifikation der Ausgabe.

Die Klasse *XmlWriterFileList* enthält die nötigen Methoden. Die zugehörige *aspx*-Datei ist bis auf den Aufruf der Klasse leer, das nötige HTML wird in der später noch diskutierten XSLT-Datei definiert.

```
Public Class XmlWriterFileList
    Inherits System.Web.UI.Page
    Private xDoc As XmlDocument = New XmlDocument()
    Private xTrans As XslTransform = New XslTransform()
    Private xE As XmlElement, xEl As XmlElement
    Private xA As XmlAttribute
    Private xArgs As XsltArgumentList = New XsltArgumentList()
```

```

Private Sub ReadDir(ByVal current As String, 
    ByVal currentElement As XmlElement)
    Dim di As DirectoryInfo = New DirectoryInfo(current)
    Try
        Dim sdi As DirectoryInfo
        For Each sdi In di.GetDirectories()
            xE = xDoc.CreateElement("directory")
            If di.GetFiles().Length > 0 Then
                xA = xDoc.CreateAttribute("hasfile")
                xA.InnerText = "true"
                xE.Attributes.Append(xA)
            End If
            xA = xDoc.CreateAttribute("name")
            xA.InnerText = di.Name
            xE.Attributes.Append(xA)
            currentElement.AppendChild(xE)
            currentElement = xE
            ReadDir(sdi.FullName, xE)
        Next
        If di.Name.Substring(0, 
            System.Math.Min(4, di.Name.Length)) <> "_vti" Then
            Dim fi As FileInfo
            For Each fi In di.GetFiles()
                xE = xDoc.CreateElement("file")
                xE1 = xDoc.CreateElement("name")
                xE1.InnerText = fi.Name
                xE.AppendChild(xE1)
                xE1 = xDoc.CreateElement("size")
                xE1.InnerText = fi.Length.ToString()
                xE.AppendChild(xE1)
                xE1 = xDoc.CreateElement("attributes")

                Dim fa As FileAttributes = fi.Attributes
                xA = xDoc.CreateAttribute("archive")
                If fa = FileAttributes.Archive Then
                    xA.InnerText = "true"
                Else
                    xA.InnerText = "false"
                End If
                xE1.Attributes.Append(xA)
                xA = xDoc.CreateAttribute("compressed")
                If fa = FileAttributes.Compressed Then
                    xA.InnerText = "true"
                Else
                    xA.InnerText = "false"
                End If
                xE1.Attributes.Append(xA)
                xA = xDoc.CreateAttribute("hidden")
                If fa = FileAttributes.Hidden Then

```



```

        xA.InnerText = "true"
    Else
        xA.InnerText = "false"
    End If
    xEl.Attributes.Append(xA)
    xA = xDoc.CreateAttribute("system")
    If fa = FileAttributes.System Then
        xA.InnerText = "true"
    Else
        xA.InnerText = "false"
    End If
    xEl.Attributes.Append(xA)
    xA = xDoc.CreateAttribute("readonly")
    If fa = FileAttributes.ReadOnly Then
        xA.InnerText = "true"
    Else
        xA.InnerText = "false"
    End If
    xEl.Attributes.Append(xA)
    xE.AppendChild(xEl)
    currentElement.AppendChild(xE)
Next
End If
Catch
    xE = xDoc.CreateElement("error")
    xE.InnerText = "Verzeichnis nicht gefunden"
    currentElement.AppendChild(xE)
End Try
End Sub

Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles MyBase.Load
    Dim path As String, file As String = "no"
    If Request.QueryString("path") Is Nothing Then
        path = "."
    Else
        path = Request.QueryString("path")
    End If
    If Request.QueryString("file") Is Nothing Then
        file = "."
    Else
        file = Request.QueryString("file")
    End If
    xTrans.Load(Server.MapPath("data/getfiles2.xslt"))
    Dim xDec As XmlDeclaration 
        = xDoc.CreateXmlDeclaration("1.0", "UTF-8", "yes")
    xDoc.AppendChild(xDec)
    xE = xDoc.CreateElement("dirlist")

```

```

        xA = xDoc.CreateAttribute("path")
        xA.InnerText = Server.MapPath(path)
        xE.Attributes.Append(xA)
        xDoc.AppendChild(xE)
        ReadDir(Server.MapPath(path), xE)
        xArgs.AddParam("showfile", "", file)
        xArgs.AddParam("aspx", "", Request.Path)
        xTrans.Transform(xDoc, xArgs, Response.Output)
    End Sub

End Class

```

Listing 5.13: Dynamisches Erzeugen und Transformieren von XML mit Parametern (XmlWriterFileList.aspx.vb)

Wie es funktioniert

Es kommt hier auf einige Details an, die ausführlicher vorgestellt werden sollen. Als Basis dient die Klasse `XmlDocument`, die aufgrund der Parameter für die `Transform`-Methode gewählt wurde. Es kam in dem Beispiel darauf an, die Ausgabe an einen universellen `TextWriter` zu senden. Eine Klasse, die diese Klasse erbt, ist `Response.Output`, der Ausgabepuffer für den Webserver. Damit können die Daten ohne Umwege zum Browser gelangen. Die Methoden zum Erzeugen der Elemente unterscheiden sich natürlich vom letzten Beispiel.

Zuerst ist die Dokumentendeklaration zu erstellen. Die drei Parameter verlangen die Angabe der Version, bei XML immer »1.0«, der Kodierung (Standard in .NET ist »UTF-8«) und des Parameters für das XML-Attribut `standalone`:

```

Dim xDec As XmlDeclaration = 
    xDoc.CreateXmlDeclaration("1.0", "UTF-8", "yes")

```

Mit `AppendChild` wird dann ein neues Element jeweils an das gerade aktuelle angehängt. Bei geschicktem Aufbau entsteht so der XML-Baum. Der Ablauf gleicht sich im Programm immer wieder. Erst wird ein neues Element erzeugt (`xE` ist vom Typ `XmlElement`):

```
xE = xDoc.CreateElement("dirlist")
```

Dann können – optional – ein oder mehrere Attribute angehängt werden. Zuerst werden diese erzeugt:

```
xA = xDoc.CreateAttribute("path")
```

Ihnen wird dann ein Wert zugewiesen:

```
xA.InnerText = Server.MapPath(path)
```

Nun wird das Attribut angehängt:

```
xE.Attributes.Append (xA)
```

Das fertige Element kann nun selbst dem Dokument hinzugefügt werden. Enthaltener Text könnte jetzt noch mit der Eigenschaft `InnerText` eingefügt werden:

```
xDoc.AppendChild (xE)
```

Innerhalb der Methode `ReadDir`, die das ausgewählte Verzeichnis mitsamt aller Unterverzeichnisse und Dateien liest, wird dieses Verfahren immer wieder verwendet.

Parameter werden der Methode `Transform` in Form eines Objekts vom Typ `XsltArgumentList` übergeben. Dies ist letztlich eine Aufzählung, sodass beliebig viele Parameter gleichzeitig übertragen werden können. Ob diese in XSLT tatsächlich ausgewertet werden, wird nicht kontrolliert. Das Objekt wird folgendermaßen erzeugt:

```
Private xArgs As XsltArgumentList = New XsltArgumentList()
```

Jeder Parameter wird dann einzeln hinzugefügt, die drei Parameter bestimmen den Parameternamen, den Namensraumpräfix und den Inhalt des Parameters:

```
xArgs.AddParam ("showfile", "", file)
```

Der Namensraumpräfix wird im Beispiel nicht verwendet; in diesem Fall wird eine leere Zeichenkette übergeben. Die Übergabe der Parameter erfolgt dann zusammen mit dem Start der Transformation. Der erste Parameter bestimmt die Datenquelle, also das XML-Dokument, der zweite die Parameter und der dritte die Ausgabeschnittstelle:

```
xTrans.Transform (xDoc, xArgs, Response.Output)
```

In XSLT werden die Parameter über die Anweisung `<xsl:param name="parameter">` empfangen. Nachfolgend finden Sie das vollständige Listing, dass durch das Vorhandensein des Parameters `showfile` gesteuert wird.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" ⚡
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="showfile"/>
  <xsl:param name="aspx"/>
  <xsl:template match="dirlist">
```

**Parameter
verwenden**

```

<html>
  <head>
    <title>Verzeichnisliste</title>
    <style>
      ul b { background-color: #eeeeee; }
      li { font-size: 10pt; }
    </style>
  </head>
  <body>
    <basefont face="Verdana"/>
    <h1><xsl:value-of select="./@path"/></h1>

    <xsl:apply-templates />
  </body>
</html>
</xsl:template>
<xsl:template match="error">
  <div style="color:red; font-weight:bold">
    <xsl:value-of select="."/>
  </div>
</xsl:template>
<xsl:template match="directory">
  <ul><b><xsl:value-of select="./@name"/></b><br/>
    <xsl:apply-templates/>
  </ul>
</xsl:template>
<xsl:template match="file">
  <li>
    <xsl:element name="a">
      <xsl:attribute name="href">
        <xsl:value-of select="$aspx"/>?file=
        <xsl:value-of select="name"/>
      </xsl:attribute>
      <xsl:value-of select="name"/>
    </xsl:element>
    <xsl:if test="$showfile=name">
      <span style="border:1px blue solid">
        <xsl:apply-templates
          select="size|attributes" />
      </span>
    </xsl:if>
  </li>
</xsl:template>
<xsl:template match="attributes">
  Attribute:
  <xsl:choose>
    <xsl:when test="@archive='true'">A</xsl:when>
    <xsl:when test="@compressed='true'">C</xsl:when>
  </xsl:choose>

```

```
<xsl:when test="@hidden='true'">H</xsl:when>
<xsl:when test="@system='true'">S</xsl:when>
<xsl:when test="@readonly='true'">R</xsl:when>
</xsl:choose>
</xsl:template>
<xsl:template match="size">
  Größe: <xsl:value-of select="."/> Byte
</xsl:template>

</xsl:stylesheet>
```

Listing 5.14: Parametrisiertes XSLT-Programm (data/getfiles2.xslt)

Die Auswertung des Parameters findet mit Hilfe einer `<xsl:if>`-Anweisung statt. Für den Fall, dass der Name einer Datei übergeben wurde, wird beim Auftreten dieser Datei in der gesamten XML-Datei die Ausgabe der zusätzlichen Informationen (Attribute und Größe) gesteuert:

```
<xsl:if test="$showfile=name">
```

Auswertung des Parameters

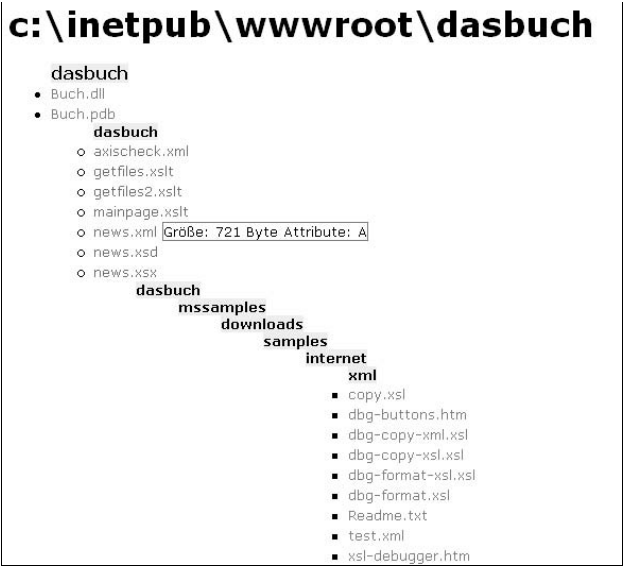


Abbildung 5.35: Ausgabe eines Verzeichnisses mit Anzeige von Zusatzdaten

Das ist übrigens nicht ganz perfekt, weil theoretisch Dateinamen mehrfach auftreten können. Das Programm wurde aber ohnehin soweit vereinfacht, dass die blanke Basisfunktion übrig bleibt. Es ist eine gute Übung, sowohl diese kleine Nachlässigkeit als auch notwendige Fehlerroutrinen einzubauen.

Diskussion und Übung

Zusammenfassung

Dieser Abschnitt zeigte, wie Sie XML direkt erzeugen, transformieren, mit Parametern die Transformation steuern und so den notwendigen »Arbeitscode« auf ein Minimum reduzieren. Denn die eigentliche Arbeit wird von der Transformation und dem XML-Parser erledigt. Das Programm ist deshalb auch im Verhältnis zum erreichten Effekt sehr klein. Dies kann bei der Entwicklung größerer Applikationen vorteilhaft sein, weil die Fehlerquote sinkt. Darüberhinaus ist ein großer Teil der Funktionen auch dann noch änderbar, wenn der Programmcode selbst nur als Assembly vorliegt. Sie können so robuste und dennoch anpassbare Programme schreiben, ohne den Quellcode herausgeben zu müssen. Sie können außerdem gegenüber Kunden darauf verweisen, dass lediglich Kenntnisse in XML und XSLT notwendig sind, um die Änderungen selbst vorzunehmen. Beide Formate sind bekannt, normiert und gut dokumentiert. Zu beiden gibt es viel Fachliteratur und viele Seminare. Und beide sind letztlich einfacher zu beherrschen als ein vollständig in VB.NET geschriebenes Programm. Wenn Sie Bedarf an In-House-Schulungen zu diesen Themen haben, stehen Ihnen die Autoren gern zur Verfügung.

B Praktische Programmierung in ASP.NET

6 Programmierung von Web Forms

Die Schnittstelle zum Anwender wird durch das gebildet, was der Browser anzeigt. Dies sind – im weitesten Sinne – so genannte »Web Forms«. Dies ist ein Oberbegriff für eine gestaltungsorientierte Softwareentwicklung, der an Win Forms aus der Windows-Programmierung angelehnt ist. Letztlich dreht sich alles um die Gestaltung der Benutzerschnittstelle und die Hinterlegung der benötigten Funktionalität.

Dabei geht es um weit mehr, als klassische HTML-Formulare erfassen. Dieses Kapitel führt umfassend und tiefgehend in die Welt der »Web Forms« ein.

6.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

6.1.1 Über dieses Kapitel

In ASP.NET stehen viele Klassen zur Verfügung, die speziell auf die Erzeugung und Verarbeitung von HTML-Elementen ausgelegt sind. Dabei wird weit mehr realisiert, als andere Programmiersysteme bislang boten. Denn neben der Bedienung der Browser-schnittstelle wird auch der Programmierer zufrieden gestellt. Alle Elemente einer HTML-Seite werden auch als Teil einer Objektstruktur der Seite abgebildet. Damit wird erstmals ein aus der Clientprogrammierung bekanntes Prinzip – das Document Object

Model (DOM) – in ähnlicher Art und Weise auf die Serverprogrammierung übertragen. In der Praxis heißt das, dass sich die Web- und die Windowsprogrammierung im Stil annähern und dabei komplexe Applikationen vergleichbar effizient entwickelt werden können.

Dieses Kapitel führt zuerst in die Welt der HTML Server-Steuerelemente ein, die eine direkte Repräsentation als HTML-Tag haben. Abschnitt 6.2, »HTML Server-Steuerelemente (HTML Server Controls)« ab Seite 504 behandelt dies detailliert.

Gezeigt werden auch die Web Server-Steuerelemente, die neben einfachen HTML-Tags auch komplexere und dennoch HTML-kompatible Tag-Sammlungen erzeugen. Lesen Sie ab Abschnitt 6.4, »Web Server-Steuerelemente (Web Server Controls)« ab Seite 535, wie diese Steuerelemente eingesetzt werden.

Zu den wichtigsten Steuerelementen gehören die Daten-Steuerelemente und solche zur Erstellung von Listen. Informationen dazu bieten die Abschnitte 6.5, »Listen-Steuerelemente« ab Seite 556 und 6.6, »Vorlagengebundene Daten-Steuerelemente« ab Seite 566.

Für die Gestaltung von Formularen mit hervorragender Benutzerführung eignen sich die Kontroll-Steuerelemente, die im Abschnitt 6.7, »Kontroll-Steuerelemente (Validation Controls)« ab Seite 616 beschrieben werden. Sie können damit Prüfbedingungen für Eingabefelder definieren – Server- und Clientseitig.

Eine Erweiterung des Konzepts der Web Server-Steuerelemente stellen die besonders umfangreichen komplexen Steuerelemente dar. Dort finden Sie solche zum Darstellen von Kalendern, rotierenden Bannern und zur Ausgabe von XML. Einen kompakten Überblick gibt Abschnitt 6.8, »Komplexe Steuerelemente (Rich Controls)« ab Seite 635.

Eine Fortsetzung des Themas finden Sie im folgenden Kapitel 7, »Erweiterte Web Form-Programmierung« ab Seite 645. Außerdem ist ein Blick in Kapitel 8, »Protokollnahe und ablauforientierte Programmierung« ab Seite 745 angebracht, wo die Basisobjekte der ASP.NET-Welt vorgestellt werden, die die Steuerelemente sinnvoll ergänzen.

6.1.2 Wegweiser in die Referenz

Die folgende Abbildung zeigt die Einordnung des Namensraumes `HtmlControls` in die Hierarchie des Frameworks.

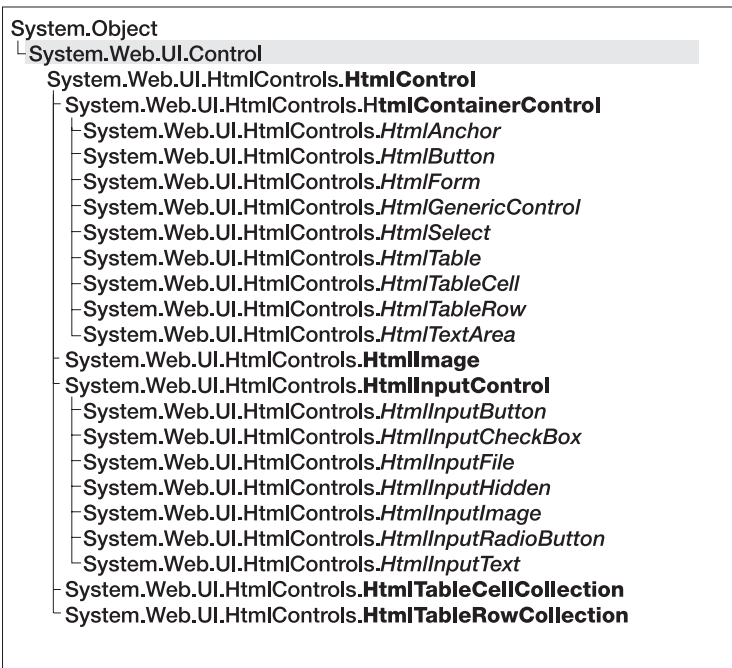


Abbildung 6.1: `HtmlControl` im Framework und untergeordnete Klassen

Für die Web Server-Steuerelemente finden Sie den allgemeinen Überblick der wichtigsten Abhängigkeiten in der folgenden Abbildung 6.2.

Abstrakte Basisklassen, auf die Sie im Allgemeinen nicht direkt zugreifen, sind fett gekennzeichnet; alle übrigen, die bei der Programmierung eine Rolle spielen, sind kursiv. Die in der Abbildung gezeigten Klassen repräsentieren alle eigenständige Steuerelemente. Viele weitere Klassen werden zur Ableitung von Attributen verwendet. Diese stammen aber nicht aus dem Namensraum `System.Web.UI.Control`.

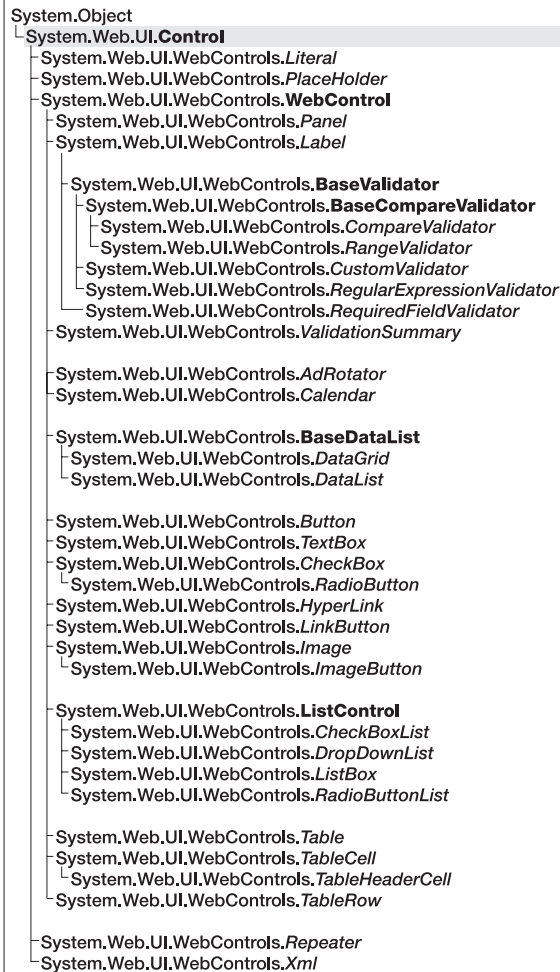


Abbildung 6.2: Hierarchie der Klassen im Namensraum System.Web.UI.Control

6.2 HTML Server-Steuerelemente (HTML Server Controls)

In den vorangegangenen Kapiteln wurden Server-Steuerelemente bereits gelegentlich verwendet. Sie erlauben den einfachen programmtechnischen Zugriff auf HTML-Tags. In diesem Abschnitt werden einleitend die einfachen HTML Server-Steuerelemente gezeigt.

6.2.1 Einführung

Im Gegensatz zu den später besprochenen Web Server-Steuerelementen werden als HTML Server-Steuerelemente solche bezeichnet, die eine direkte Entsprechung in HTML-Tags haben. Sie entstehen, indem im entsprechenden HTML-Tag das Attribut `runat="server"` ergänzt wird. Damit der Zugriff vom Programm aus erfolgen kann, ist noch das Attribut `id` erforderlich. Alle so gekennzeichneten Tags werden Teil einer Auflistung innerhalb des Objekts `Page` bzw. untergeordneter Kollektionen anderer Steuerelemente. Je nach Typ und Inhalt werden verschiedene Eigenschaften und Methoden zur Verfügung gestellt. Damit lassen sich die Attribute und bei Containerelementen auch der Inhalt vom Code aus verändern.

HTML programmieren

In der Klassenhierarchie des .NET-Frameworks werden die HTML Server-Steuerelemente im Namensraum `System.Web.UI.HtmlControls` definiert. UI steht für User Interface, es handelt sich also um eine Klasse, die der Gestaltung der Benutzerschnittstelle dient. Auch dieser Namensraum wird in ASP.NET-Seiten automatisch bereitgestellt. Lediglich beim Einsatz der Code Behind-Technik müssen Sie den Zugriff explizit deklarieren. Dann schreiben Sie an den Anfang der Datei folgende Zeile:

Namensraum

```
Imports System.Web.UI.HtmlControls
```

Wenn Sie mit Visual Studio .NET arbeiten, wird der benötigte Namensraum immer im Code eingebunden. HTML-Elemente, die Sie aus der Toolbox im Designer hinzufügen, sind immer HTML bzw. Web Server-Steuerelemente.



Ereignisse

Wie viele andere Objekte, kennen auch HTML Server-Steuerelemente Ereignisse, die auf Benutzeraktionen reagieren. Mit entsprechend benannten Ereignisbehandlungsmethoden können Sie diese im Code abfangen. Der Name ist frei wählbar, solange er den Bedingungen der Namensvergabe für Methodennamen in der gewählten Programmiersprache entspricht. Er wird im HTML-Element selbst meist als Attribut `onClick` festgelegt (es gibt noch einige andere solche Attribute für Ereignisse, aber die Reaktion auf einen Mausklick ist die häufigste). Beachten Sie, dass dieses Verarbeitungsprinzip nichts am üblichen Ablauf zwischen Browser

Ereignisbehandlung

und Webserver ändert. Wenn Sie auf ein Klick-Ereignis reagieren möchten, wird der Browser das Formular an den Server absenden.

Wenn der Benutzer einen Link anklickt, wird er den HTTP-Befehl GET auslösen, wenn der Link außerhalb eines Formulars liegt. Innerhalb wird dagegen JavaScript verwendet, um die im Link verknüpften Informationen per HTTP-POST zu übertragen. Die funktioniert so , weil ASP.NET die fertige Seite um die entsprechenden Funktionen ergänzt und sich dazu einer mitgelieferten JavaScript-Bibliothek bedient.

In jedem Fall handelt sich also um eine Anforderung (Request) des Clients. Damit der Server nun zuordnen kann, welches Formular zurückgesendet wird, wird ein verstecktes Feld mit Statusinformationen erzeugt, dem so genannten Anzeigestatus (ViewState). Der Anzeigestatus kann für jedes Element und die gesamte Seite ein- und ausgeschaltet werden.

Kompatibilität zu HTML

Auch für Netscape

Der vollkommen auf HTML und HTTP aufbauende Verarbeitungszyklus der HTML Server-Steuerelemente führt auch zu einer sehr wichtigen Aussage: Alle Steuerelemente sind vollständig HTML-kompatibel und lassen sich mit allen Browsern darstellen. Es handelt sich nicht um Nachfahren der ActiveX-Controls oder ähnlicher proprietärer Erweiterungen.

6.2.2 Prinzipieller Umgang mit HTML Server-Steuerelementen

Die Gegenüberstellung

Da jedes HTML Server-Steuerelement eine Entsprechung in HTML besitzt, ist eine direkte Gegenüberstellung der Namen mit den Tags sinnvoll. Die folgende Tabelle zeigt die explizit vorhandenen Klassen und die passenden HTML-Tags:

HTML-Tag	Klassenname
<a>	HtmlAnchor
<button>	HtmlButton

Tabelle 6.1: HTML-Tags und Zuordnung der Klassen des Namensraumes System.Web.UI.HtmlControls

HTML-Tag	Klassenname
Alle sonstigen Tags, die allein stehend auftreten, beispielsweise <code>
</code>	<i>HtmlControl</i>
<code><form></code>	<i>HtmlForm</i>
Dient der Generierung neuer Elemente	<i>HtmlGenericControl</i>
<code></code>	<i>HtmlImage</i>
<code><input type="button"></code> <code><input type="reset"></code> <code><input type="submit"></code>	<i>HtmlInputButton</i>
<code><input type="checkbox"></code>	<i>HtmlInputCheckBox</i>
<code><input type="text"></code> <code><input type="file"></code> <code><input type="submit"></code>	<i>HtmlInputControl</i>
<code><input type="file"></code>	<i>HtmlInputFile</i>
<code><input type="hidden"></code>	<i>HtmlInputHidden</i>
<code><input type="image"></code>	<i>HtmlInputImage</i>
<code><input type="radio"></code>	<i>HtmlInputRadioButton</i>
<code><input type="text"></code> <code><input type="password"></code>	<i>HtmlInputText</i>
<code><select></code>	<i>HtmlSelect</i>
<code><textarea></code>	<i>HtmlTextArea</i>
<code><input type="checkbox"></code>	<i>HtmlCheckBox</i>
<code><table></code>	<i>HtmlTable</i>
<code><td></code>	<i>HtmlTableCell</i>
<code><tr></code>	<i>HtmlTableRow</i>
Mehrere Zellen einer Reihe	<i>HtmlTableCellCollection</i>
Mehrere Reihen einer Tabelle	<i>HtmlTableRowCollection</i>
<code><tr></code>	<i>HtmlTableRow</i>
Alle sonstigen Tags, die als Container auftreten, beispielsweise <code></code> , <code><i></code> usw.	<i>HtmlContainerControl</i>

Tabelle 6.1: HTML-Tags und Zuordnung der Klassen des Namensraumes System.Web.UI.HtmlControls (Forts.)

Der Schwerpunkt liegt eindeutig bei Formular-Elementen, die aber erfahrungsgemäß auch den meisten Aufwand verursachen und die von der Unterstützung in ASP.NET am ehesten profitieren.

Eine Sonderstellung nimmt `HtmlGenericControl` ein, eine Klasse, die der Erzeugung allgemeiner neuer Objekte oder dem Zugriff auf allgemeine Tags dient, die dann der Elemente-Kollektion der Seite hinzugefügt werden können. Angesprochen wird das korrespondierende Objekt über den aus dem Attribut `id` abgeleiteten Namen.

In hinterlegtem Code ist zusätzlich die Deklaration notwendig, die die Sicherheitsebene `Protected` oder `Public` haben muss (wobei in VB.NET die Ebene `Public` standardmäßig verwendet wird; der Visual Studio .NET-Designer dagegen setzt `Protected` ein):

```
Protected MeinIdName As HtmlGenericControl
```

Das Prinzip der Steuerelemente-Hierarchie der Seite

Besteht die gesamte Seite nach der Analyse aus HTML Server-Steuerelementen, ist die Manipulation leicht möglich. Betrachtet man die Elemente als Hierarchie, ist auch die Anordnung neuer Elemente an einer bestimmten Stelle möglich. So kann HTML sehr gut programmtechnisch manipuliert werden. Lesen Sie in Abschnitt »Umgang mit der Objekthierarchie« ab Seite 510 mehr zum Umgang mit der Objekthierarchie.

Die folgende Abbildung zeigt das Prinzip des Zusammenhanges zwischen den Steuerelementen (Links), die als Ausgangspunkt dienen. Bei der Abarbeitung des Programms erfolgt die Verarbeitung im Code (Mitte), woraus dann am Ende des Verarbeitungszyklus der HTML-Code für den Browser entsteht (Rechts). Wenn der Benutzer nun mit der Seite interagiert, beispielsweise durch Anlicken einer Schaltfläche, wird das Formular zurück zu Server gesendet, sie Seite erneut verarbeitet (Mitte) und die Ereignisse werden verarbeiten (*Btn1_Click* im Bild), die mit der Benutzeraktion korrespondieren. Dies führt möglicherweise zu einem anderen Seitenaufbau.

Die Benennung der Steuerelemente nach Typ und ID bestimmt die Erscheinungsform im Code. Der Inhalt dagegen kann im Code entnommen oder manipuliert werden.

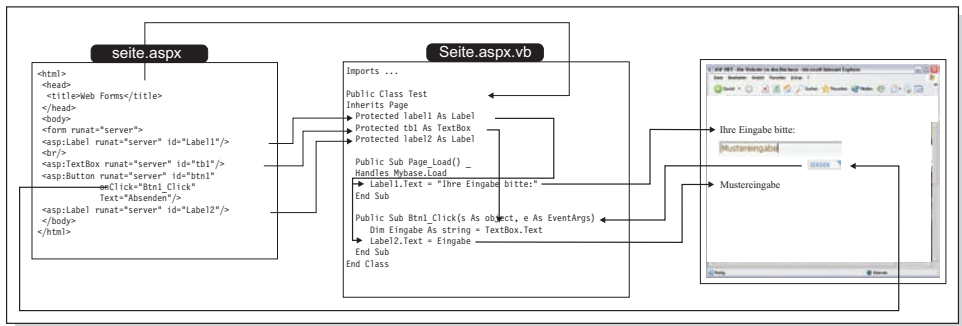


Abbildung 6.3: Zusammenhang zwischen Steuerelement, Code und Ausgabe (vereinfacht)

Universeller Zugriff auf HTML-Tags

Wie bereits in der Tabelle angedeutet, gibt es nur für die wichtigsten HTML-Tags eine direkte Entsprechung in Form einer eigenen Klasse. Der Zugriff ist aber generell auf alle Tags möglich, wenn diese mit `runat="server"` zum Bestandteil der Hierarchie gemacht werden. Diese Elemente werden automatisch von der abstrakten Klasse `HtmlControl` bzw. `HtmlContainerControl` abgeleitet. Der Unterschied besteht lediglich darin, wie die Elemente auftreten dürfen. So enthalten einige keinen Inhalt, wie beispielsweise `<hr/>` oder `<br/>`. Containerelemente treten dagegen paarweise auf, dazu gehören `<b></b>` usw.

HtmlControl
HtmlContainerControl

Eine wichtige Eigenschaft ist `Attributes`. Dies ist eine Kollektion, da Elemente mehr als ein Attribut enthalten können. Das folgende Beispiel zeigt die Anwendung:

Attribute-Kollektion

```
<script language="vb" runat="server">
Private Sub Page_load()
    head1.Attributes("style") = "color:" &
Request.QueryString("header")
End Sub
</script>
<html>
<head>
<title>HTMLControl</title>
</head>
<body>
<h1 style="" id="head1" runat="Server">
    Willkommen auf unserer Website</h1>
    Viel Spaß mit unseren Angeboten:
```

```

<ul>
  <li><a href="htmlcontrol.aspx?header=blue">
    Blaue Überschrift sehen</a></li>
  <li><a href="htmlcontrol.aspx?header=red">
    Rote Überschrift sehen</a></li>
  <li><a href="htmlcontrol.aspx?header=green">
    Grüne Überschrift sehen</a></li>
</ul>
</body>
</html>

```

Listing 6.1: Anwendung von HTML Server-Steuerelementen (HtmlControl.aspx)

Wie es funktioniert

Zum HTML Server-Steuerelement wird hier das Tag `<h1>` bestimmt. Manipuliert wird das Attribut `style`, mit dem sich die Gestaltung über Cascading Style Sheets (CSS) bestimmen lässt. Die Definition erfordert neben dem Attribut `runat="server"` auch die Vergabe eines Namens über `id`:

```
<h1 style="" id="head1" runat="Server">
```

Der Name des Objekts ist nun *head1*. Der Zugriff auf ein bestimmtes Attribut erfolgt über die bereits verwendete Schreibweise für Kollektionen:

```
head1.Attributes("style") =
```

Die meisten Attribute sind sowohl schreib- als auch lesbar.

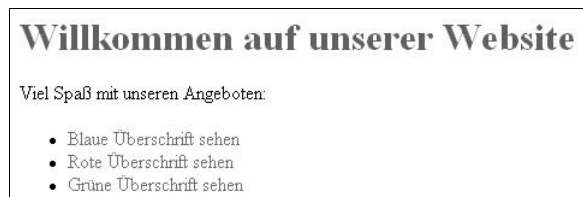


Abbildung 6.4: Nach dem Klick auf einen Link wechselt die Farbe der Überschrift.

Umgang mit der Objekthierarchie

HTML-Tags können innerhalb der Seite nicht nur sequenziell, sondern auch hierarchisch auftreten:

```
<b><i></i></b>
```

In dieser Folge ist `<i>` dem Tag `<b>` untergeordnet. Da alle Objekte innerhalb der Seite instanziiert werden, spielt das nicht zwingend

eine Rolle bei der Programmierung. Wenn Sie aber auf die Hierarchie Rücksicht nehmen müssen, gibt es einen Weg, die Anordnung zu ermitteln. `HtmlControl` kennt dazu die Eigenschaft `ControlCollection`. Dies ist eine Kollektion der Objekte, die dem betreffenden Objekt untergeordnet sind. Dies ist nur dann der Fall, wenn es sich um Containerelemente handelt. Einfache Tags können in der Kollektion zwar selbst enthalten sein, haben aber keine Kindelemente.

Neben dem Zugriff auf Tags kann auch jedes Attribut erreicht werden. Da häufig CSS verwendet wird und das `style`-Attribut in HTML wiederum viele Anweisungen enthalten kann, besteht zusätzlich eine Möglichkeit, auf jeden einzelnen Stylewert zuzugreifen. Beide, Attribute und Styles, bilden Kollektionen, wenn mehrere Werte vorhanden sind.

Attribute

Damit bei der Abfrage das Programm nicht hängen bleibt, wenn ein Element ohne die erwarteten Kindelemente angetroffen wird, ist noch die Methode `HasControls` von Bedeutung. Sie gibt `True` zurück, wenn weitere Kindelemente existieren. Typisch für den Umgang mit Hierarchien sind rekursive Methoden. Diese rufen sich selbst auf und können beliebig tiefe Verschachtelungen analysieren. Das folgende Beispiel zeigt eine solche Anwendung:

```
<body>
  <h1>Willkommen auf unserer Website</h1>
  Viel Spaß mit unseren Angeboten:
  <ul runat="server" type="disk">
    <li runat="server" style="color:red">
      Blaue Überschrift sehen</li>
    <li runat="server" type="square"
      style="color:blue; font-weight:bold">
      Rote <b runat="server">Überschrift</b> sehen</li>
    <li runat="server">Grüne Überschrift sehen</li>
  </ul>
  Servercontrols:<br/>
  <pre><asp:label id="ausgabe" runat="server"/></pre>
</body>
```

Listing 6.2: Auslesen aller HTML Server-Steuerelemente einer Seite mit Attributen (HtmlControlContainer.aspx)

Die eigentliche Arbeit wird in der Code-Datei und dort in der Methode `getControl` erledigt:

```
Public Class HtmlControlContainer
  Inherits System.Web.UI.Page
```

```

Protected ausgebe As Label

Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles MyBase.Load
    getControls(Page)
End Sub

Private Sub getControls(ByVal c As Control)
    If c.HasControls() Then
        If TypeOf c Is HtmlGenericControl _
            Or TypeOf c Is HtmlControl Then
            Dim hc As HtmlControl
            hc = CType(c, HtmlControl)
            ausgebe.Text += ("<" + hc.TagName)
            Dim hk As ICollection = hc.Attributes.Keys
            Dim ss As String
            For Each ss In hk
                ausgebe.Text += (" " + ss + "=\"" + _
                    + hc.Attributes(ss) + " \"")
            Next
            ausgebe.Text += ("> (ID = \"" + hc.ClientID _
                + " \"")<br />")
        End If
        If c.HasControls() Then
            Dim cs As Control
            For Each cs In c.Controls
                getControls(cs)
            Next
        End If
    End If
End Sub
End Class

```

Listing 6.3: Auswertung generischer HTML-Serversteuerelemente
(*HtmlControlContainer.aspx.vb*)

Wie es funktioniert

Die Tags, die untersucht werden können, sind im HTML-Teil des Beispiels mit `runat="Server"` gekennzeichnet. Die Ausgabe der Analyse erfolgt mit einem Web Server-Steuerelement:

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

Damit darauf zugegriffen werden kann, muss bei hinterlegtem Code eine entsprechende Deklaration erfolgen:

```
Protected ausgebe As Label
```

Der Modifizierer `Protected` gibt die Sichtbarkeit innerhalb des Moduls an. Der in VB.NET standardmäßig verwendete Wert `Public`

wäre ebenso möglich. Dann folgt der Typ des Steuerelements, hier `Label`. Als Letztes ist der Namen anzugeben, der exakt der Schreibweise im Attribut `id` entsprechen muss.

Web Server-Steuerelemente werden im nächsten Abschnitt 6.4, »Web Server-Steuerelemente (Web Server Controls)« ab Seite 535 beschrieben. Der Einsatz erfolgt hier, damit das Ausgabe-Tag nicht selbst zum Bestandteil der Hierarchie wird und die Behandlung unnötig erschwert.



Das eigentliche Programm startet in `Page_Load`. Hier wird die Methode `getControls` aufgerufen, als Argument wird die Seite selbst übergeben. Der Zugriff auf `Page` wäre zwar auch direkt möglich, die Methode soll sich aber rekursiv aufrufen und dabei wechselt der Parameter später.

Rekursive Aufrufe werden oft verwendet, wenn eine beliebig verschachtelte Struktur durchlaufen werden soll. Dabei ruft sich eine Methode selbst auf, bis das Ende der Struktur erreicht ist.



Die Definition erwartet, dass ein Wert vom Typ `Control` übergeben wird:

```
Private Sub getControls(ByVal c As Control)
```

Damit wird beim ersten Aufruf eine implizite Typumwandlung erforderlich, die .NET selbstständig erledigt. In der Variablen `c` stehen jetzt nur noch alle Objekte vom Typ `Control`. Nun wird überprüft, ob das Argument weitere Steuerelemente enthält:

```
If c.HasControls() Then
```

Ist das der Fall, wird untersucht, ob es sich um HTML Server-Steuerelemente vom Typ `HtmlControl` bzw. `HtmlGenericControl` handelt. Nicht alle Kindelemente des Objekts `Page` müssen dies sein:

```
If TypeOf c Is HtmlGenericControl Or TypeOf c Is HtmlControl Then
```

Wenn das so ist, wird eine explizite Typumwandlung durchgeführt; das Objekt wird mehrfach benötigt. Beachten Sie, dass dies unbedingt erforderlich ist, da die Tatsache, dass es sich um ein HTML Server-Steuerelement handelt, noch nicht zur Umwandlung des Typs `Control` führt. Sie können nicht erwarten, dass der Compiler dies erkennt, da es aus Sicht der Programmlogik nicht zwingend eindeutig ist:

```
Dim hc As System.Web.UI.HtmlControls.HtmlControl
```

`hc` ist nun tatsächlich ein Objekt vom Typ `HtmlControl`, das auch sicher ein HTML Server-Steuerelement ist. Dieser Typ ist mit `HtmlGenericControl` vollkommen kompatibel. Diese Objekte besitzen eine Reihe von Eigenschaften; die wichtigsten werden am Ende des Abschnitts zusammengefasst. Zuerst wird der Name des Tags ausgelesen und dem Anzeigeelement zugewiesen:

```
ausgabe.Text += ("<" + hc.TagName)
```

Zum Auslesen gibt es viele Verfahren. Hier wird die Kollektion über die Schlüssel erreicht. Die Variable `hk` bildet eine einfache Kollektion vom Typ `ICollection`. Die Eigenschaft `Keys` gibt die Schlüssel aus:

```
Dim hk As ICollection = hc.Attributes.Keys
```

Nun werden alle Attribute anhand der Schlüssel mit einer *For Each*-Schleife durchlaufen. Hier wird direkt auf die Attribute zugegriffen. Dies ist möglich, da die Kollektion intern die Schnittstelle `IEnumerator` implementiert. Die Betrachtung der vielfältigen Möglichkeiten von Kollektionen finden Sie in Abschnitt 4.3, »Aufzählungen und Kollektionen« ab Seite 222.

```
ausgabe.Text += (" " + ss + "=" + hc.Attributes(ss) + " \ " + "
```

Eine weitere Eigenschaft ermittelt die interne ID des Elements. Die Vergabe einer ID ist nicht zwingend notwendig, außer wenn Sie im Code darauf zugreifen wollen. ASP.NET vergibt selbstständig eindeutige ID-Nummern, wenn dies für die interne Verarbeitung notwendig ist. In diesem Beispiel sollen diese angezeigt werden. Dazu wird die Eigenschaft `ClientID` abgefragt. Wird keine ID angegeben, sind ID und `ClientID` identisch:

```
ausgabe.Text += ("> (ID = " + hc.ClientID + " "<br />")
```

HasControls

Der letzte Teil der Methode `getControls` dient dem rekursiven Aufruf. Dazu wird wieder überprüft, ob das Objekt weitere untergeordnete Steuerelemente enthält:

```
If c.HasControls() Then
```

Ist das der Fall, werden diese mit *For Each* durchlaufen und die Methode für jedes Element aufgerufen. Durch die Rekursion wird dieser Prozess beliebig tief fortgesetzt. Irgendwann ist `HasControls` nicht mehr `True`. Dann läuft die Methode zum Ende durch und löst die rekursiven Aufrufe wieder auf.

Wenn Sie derartige Zugriffe auf die HTML Server-Steuerelemente entwerfen, sollten Sie die wichtigsten Methoden und Eigenschaften kennen. Die Tabellen am Ende des Abschnitts fassen diese zusammen.

Methoden und Eigenschaften

Willkommen auf unserer Website

Viel Spaß mit unseren Angeboten:

- Blaue Überschrift sehen
- Rote Überschrift sehen
- Grüne Überschrift sehen

Servercontrols:

```
<ul type="disk"> (ID = "_ctl0")
<li style="color:red"> (ID = "_ctl1")
<li type="square" style="color:blue; font-weight:bold"> (ID = "_ctl2")
<b> (ID = "_ctl3")
</li> (ID = "_ctl4")
```

Abbildung 6.5: Ausgabe serverseitig verwalteter Elemente mit Attributen

Container verhalten sich im Prinzip ebenso, zusätzlich kann aber der Inhalt kontrolliert werden. Dazu dienen die Eigenschaften `InnerHTML`, `InnerText` und bei einigen Elementen `Text`.

Besonderheiten der Container-Steuerelemente

Container-Steuerelemente können Text enthalten. Wenn Sie beispielsweise eine Fehlermeldung nur bei Bedarf anzeigen lassen möchten, können Sie den Text vom Programm aus leicht beeinflussen. Das folgende Beispiel zeigt die Anwendung:

Container enthalten Text oder andere Steuerelemente

```
<body>
  <h1>Willkommen auf unserer Website</h1>
  <div style="background-color:#eeeeee; font-family:Arial"
    runat="Server" id="errorMessage"/>
  Wählen Sie den den richtigen Link:<br/>
  <ul>
    <li><a href="HtmlControlContainerText.aspx?test=2">
      Versuch 1</a></li>
    <li><a href="HtmlControlContainerText.aspx?test=1">
      Versuch 2</a></li>
    <li><a href="HtmlControlContainerText.aspx?test=2">
      Versuch 3</a></li>
  </ul>
</body>
```

Listing 6.4: Anwendung von Container Controls (*HtmlControlContainerText.aspx*)

Die Auswertung der GET-Parameter erfolgt in der `Page_Load`-Methode:

```
Public Class HtmlControlContainerText
    Inherits System.Web.UI.Page
    Protected ErrorMessage As HtmlContainerControl

    Private Sub Page_Load(ByVal sender As Object, ByVal e As
EventArgs) _
        Handles MyBase.Load
        Select Case Convert.ToInt32(Request.QueryString("test"))
            Case 0
                ErrorMessage.InnerText = ""
                Exit Sub
            Case 1
                ErrorMessage.InnerText = "Super, das war richtig"
                Exit Sub
            Case 2
                ErrorMessage.InnerText = "Schade, falscher Link"
                Exit Sub
        End Select
    End Sub
End Class
```

Listing 6.5: Auswertung der Parameter (HtmlControlContainerText.aspx.vb)

Wie es funktioniert

Die Seite enthält einen serverseitigen Container¹:

```
<div runat="Server" id="errorMessage"/>
```

Auf das Objekt mit dem Namen `errorMessage` kann nun außer mit den bereits bei `HtmlControl` beschriebenen Eigenschaften auch mit `InnerText` und `InnerHtml` zugegriffen werden. Damit ist die Manipulation des eingeschlossenen Textes möglich.

**InnerText
InnerHtml**

Die Eigenschaft `InnerText` schreibt den Text nicht nur in das Tag, sondern maskiert auch die für HTML wichtigen Zeichen, »<«, »>« und »&«; durch die Entitäten «<«, »>« und »&« dargestellt. Wenn HTML ausgegeben werden soll, ist deshalb `InnerHtml` zu verwenden.

¹ Die Tatsache, dass dieses Element geschlossen ist, sollte nicht darüber hinweg täuschen, dass `<div>` per Definition Text oder andere Elemente enthalten darf und sich in der gezeigten Schreibweise wie ein leerer Container verhält, der freilich jederzeit gefüllt werden kann – genau das wird im Beispiel auch gemacht.



Abbildung 6.6: Erzeugen von Text zur Laufzeit (Listing 6.4 und 6.5)

6.3 HTML Server-Steuerelemente im Detail

Nach der kompakten Einführung sollen die HTML Server-Steuerelemente nun detaillierter betrachtet werden, denn für die Gestaltung von Websites spielen sie immer eine zentrale Rolle.

6.3.1 Gemeinsame Eigenschaften und Methoden

Die meisten HTML Server-Steuerelemente besitzen einen Pool von gemeinsamen Eigenschaften und Methoden, abgeleitet aus der Klasse `Control`.

Wichtige Eigenschaften

Dieser Abschnitt zeigt die wichtigsten Eigenschaften und Methoden der HTML Server-Steuerelemente in tabellarischer Form.

Eigenschaft	Bedeutung
Attributes	Kollektion der Attribute
Style	Kollektion der Style-Anweisungen des Attributes <code>style</code>
Controls	Kollektion der untergeordneten <code>Control</code> -Objekte
TagName	Name des Tags
Parent	Zugriff auf das übergeordnete <code>Control</code> -Objekt
ID	ID wie im Attribut <code>id</code> angegeben
ClientID	Automatisch vergebene, interne ID

Tabelle 6.2: Wichtige Eigenschaften von `HtmlControl`

Eigenschaft	Bedeutung
Visible	Steuert die Sichtbarkeit, true = Sichtbar, false = Unsichtbar
Disabled	Steuert, ob das Element aktiv ist. Dies ist nicht bei allen Elementen anwendbar.

Tabelle 6.2: Wichtige Eigenschaften von `HtmlControl` (Forts.)

Wichtigen Methoden

Die wichtigsten Methoden der Steuerelemente zeigt die folgende Tabelle:

Methode	Bedeutung
<code>ToString</code>	Umwandlung in eine Zeichenkette
<code>RenderControl</code>	Schreibt das veränderte Element in ein Objekt vom Typ <code>HtmlTextWriter</code> zur späteren Ausgabe
<code>DataBind</code>	Bindet Datenbankinformationen an das Element (nur für Listenelemente und Tabellen interessant)

Tabelle 6.3: Wichtige Methoden aus `HtmlControl`

Der Umgang mit Datenbankinformationen wird in Abschnitt 9.5, »Die Datenbindung an Steuerelemente« ab Seite 926 behandelt.

6.3.2 Basisoperationen mit Steuerelementen

Wenn die gesamte Seite eine Hierarchie von Server-Steuerelementen darstellt, und Sie diese manipulieren können, ist auch das Entfernen und Erzeugen möglich.

Steuerelemente
entfernen
`LiteralControl`

Das Entfernen ist mit den Mitteln einer Kollektion möglich. Alle Steuerelemente einer Ebene sind in solchen Kollektionen vom Typ `ControlCollection` enthalten. Zum Entfernen wird die Methode `Remove` verwendet, die ein Steuerelement als Argument erwartet, bzw. `RemoveAt`, wo ein numerischer Index erforderlich ist. Wenn alle Elemente entfernt werden sollen, kann `Clear` eingesetzt werden. Der Index ist leicht zu ermitteln. In der Objekthierarchie tauchen nur die Steuerelemente explizit auf, die Sie mit `runat="server"` gekennzeichnet haben. Alles übrige wird als Objekt vom Typ `LiteralControl` verfügbar gemacht. Wie das bei einem typischen HTML-Quelltext aussieht, ist der folgenden Abbildung zu entnehmen.

```

<html>
<head>
  <title>HTMLControl</title>
</head>
<body>
  <h1>Willkommen auf unserer Website</h1>
  W&auml;hlen Sie den den richtigen Link:<br/>
  <ul>
    <li><a runat="Server" href="htmlcontainer.aspx?test=2">Ver<sup>1</sup>ch 1</a></li>
    <li><a runat="Server" href="htmlcontainer.aspx?test=1">Ver<sup>3</sup>ch 2</a></li>
    <li><a runat="Server" href="htmlcontainer.aspx?test=2">Ver<sup>5</sup>ch 3</a></li>
  </ul>
</body>
</html>

```

Abbildung 6.7: Indizierung der Steuerelemente einer HTML-Seite

Als `HtmlControl`- oder `HtmlContainerControl`-Steuerelement werden nur diejenigen erfasst, die entsprechend gekennzeichnet sind. Alles Übrige wird zusammengefasst und steht als `LiteralControl`-Objekt in Form von Text zur Verfügung. Die interne Indizierung beginnt mit 0. In Abbildung 6.7 sind dies die Elemente 0, 2, 4 und 6. Um beispielsweise aus dem gezeigten Code den zweiten Link zu entfernen, eignet sich folgende Vorgehensweise:

Indizierung der Elemente

```

Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    If Page.HasControls() Then
        Page.Controls.RemoveAt(3)
    End If
End Sub

```

Listing 6.6: Entfernen eines HTML Server-Steuerelements (Code in `HtmlControlCollectionRemove.aspx.vb`, ausführbar über `HtmlControlCollectionRemove.aspx`)

Das Hinzufügen von Elementen ist ebenso möglich, setzen Sie dazu die Methode `Add` ein:

Hinzufügen

```

Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    If Page.HasControls() Then
        Dim LiElement As HtmlGenericControl
        = New HtmlGenericControl("li")
        LiElement.InnerText = "Neues Element"
        Liste.Controls.Add(LiElement)
    End If
End Sub

```

Listing 6.7: Hinzufügen von HTML-Code (Code in `HtmlControlCollectionAdd.aspx.vb`, ausführbar über `HtmlControlCollectionAdd.aspx`)

Wie es funktioniert

Hier wird zuerst ein neues Steuerelement erzeugt:

```
HtmlGenericControl LiElement= new HtmlGenericControl("li")
```

Dem Konstruktor wird dabei der Name des Elements als Zeichenkette übergeben, in diesem Fall entsteht also ein `<li>`-Tag. Dieses Tag ist ein Container und kann deshalb mit Text gefüllt werden:

```
LiElement.InnerText = "Neues Element"
```

Dann wird das Element der Kollektion der Steuerelemente des Tags `<ul>` hinzugefügt. Die Liste wird über den Namen adressiert:

```
Liste.Controls.Add(LiElement)
```

Nach Ausführung der Seite werden vier Listenelemente angezeigt:



Abbildung 6.8: Seite mit programmtechnisch erzeugtem HTML
(*HtmlControlCollectionAdd.aspx*)

6.3.3 Ereignisse verarbeiten

Serverseitig Ereignisse verarbeiten

Alle Ereignisse – insbesondere Reaktionen auf die Maus –, die clientseitig verarbeitet werden können, haben in ASP.NET ein vergleichbares serverseitiges Verarbeitungsmodell. Das bedeutet, dass Sie an ein Ereignis wie `OnClick`, also die Reaktion auf einen Mausklick, eine Ereignisbehandlungsmethode im Code binden können. Das führt zwangsläufig immer zum Senden des Formulars, wenn eine sofortige Reaktion erwünscht ist. Ohne sofortiges Senden wird die Behandlung des Ereignisses bis zum Übertragen des Formulars verzögert. Die gesamte Technik entspricht zwar weitgehend der in der Windows-Programmierung, ändert aber nichts am Prinzip des Request-Response-Spiels zwischen Client und Server und ist damit zu allen Clientsystemen kompatibel.

Dennoch ist die Erleichterung der Programmierung enorm. Dies gilt auch bei der Behandlung von Klicks auf Hyperlinks, denn intern bedient sich ASP.NET hier einem kleinen Stück JavaScript, ohne dass Sie sich als Programmierer darum kümmern müssen.

Umgang mit Hyperlinks in Formularen

Normalerweise führt ein Klick auf einen Link zu einer HTTP-GET-Anforderung. Damit ist aber die Übertragung aller anderen Werte, vor allem des in einem versteckten Feld gespeicherten Anzeigestatus, praktisch unmöglich. Wie nun löst ASP.NET dieses Problem?

JavaScript

Die Lösung ist ebenso trickreich wie banal – es wird hier intern JavaScript verwendet. In der HTML-Programmierung ist dies ein bekanntes und verbreitetes Verfahren. Im Link wird dazu als Ziel eine JavaScript-Funktion angegeben:

Das konventionelle Verfahren

```
<a href="javascript:mycall()">Klick mich!</a>
```

Die Funktion `mycall()` wird nun folgendermaßen definiert:

```
mycall()
{
    document.form.action = "formularziel.aspx";
    document.form.submit();
}
```

Der Klick auf den Link führt so zum Senden des Formulars. Damit arbeitet auch ASP.NET. Der einzige Unterschied: Sie müssen JavaScript nicht kennen und brauchen auch keine Funktionen definieren. Um Hyperlinks in Formularen behandeln zu können, müssen Sie diese nur innerhalb der `<form>`-Tags platzieren und zum Server-Steuerelement erklären. Das folgende Beispiel zeigt eine alternative Form der Datenverarbeitung. Zuerst der HTML-Teil:

Wie ASP.NET JavaScript verwendet

```
<body ms_positioning="gridlayout">
    <h1>
        shop
    </h1>
    unsere produkte:
    <form runat="server" id="myform">
        <ul>
            <li>
                <a runat="server" id="a0001"
                    onserverclick="check_click"></a>
```

```

        </li>
        <a runat="server" id="a0002"
            onserverclick="check_click"></a>
    </li>
        <a runat="server" id="a0003"
            onserverclick="check_click"></a>
    </li>
</ul>
</form>
<div id="bestaetigung" runat="server"></div>
</body>

```

Listing 6.8: Reaktion auf Klickereignisse (AnchorState.aspx)

Der Code ist wieder in eine Code-Datei ausgelagert worden:

```

Public Class AnchorState
    Inherits System.Web.UI.Page
    Protected myform As HtmlForm
    Protected WithEvents A0001 As HtmlAnchor
    Protected WithEvents A0002 As HtmlAnchor
    Protected WithEvents A0003 As HtmlAnchor
    Protected Bestaetigung As HtmlContainerControl

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As EventArgs) _
        Handles MyBase.Load
        Const ProdName As Integer = 0
        Const ProdPreis As Integer = 1
        Const ProdID As Integer = 2
        Dim ProduktArr(,) As String = New String(3, 3) {}
        ' Produktdefinition als Array
        ProduktArr(0, ProdID) = "A0001"
        ProduktArr(1, ProdID) = "A0002"
        ProduktArr(2, ProdID) = "A0003"
        ProduktArr(0, ProdName) = "Codeschreiber 1.0"
        ProduktArr(1, ProdName) = "Datenbanker 1.1"
        ProduktArr(2, ProdName) = "Projektor 2.0"
        ProduktArr(0, ProdPreis) = "6299,90"
        ProduktArr(1, ProdPreis) = "99,90"
        ProduktArr(2, ProdPreis) = "39,00"
        Dim c As ControlCollection = myform.Controls
        Dim k As Integer = 0
        Dim i As Integer
        For i = 0 To c.Count - 1
            Dim cc As Control = c(i)
            If TypeOf cc Is HtmlAnchor Then
                Dim anchor As HtmlAnchor = CType(cc, HtmlAnchor)
                If anchor.ID = ProduktArr(k, ProdID) Then
                    anchor.InnerText = ProduktArr(k, ProdName)
                End If
            End If
            k = k + 1
        Next i
    End Sub
End Class

```

```

        + " (" + ProduktArr(k, ProdPreis) + ")"
    End If
    k = k + 1
End If
Next
End Sub

Public Sub check_Click(ByVal sender As Object,
    ByVal e As EventArgs) _
Handles A0001.ServerClick,
    A0002.ServerClick,
    A0003.ServerClick
    Dim Link As HtmlAnchor = CType(sender, HtmlAnchor)
    Bestaetigung.InnerHtml = "Artikel " & Link.InnerText
    & " mit ID " & Link.ID
    & " wurde
gew&auml;hlt."
End Sub
End Class

```

Listing 6.9: Dynamisches Füllen von HTML-Elementen mit Daten und Definition der Klick-Methode check_Click (AnchorState.aspx.vb)

Das Programm basiert auf den mit `runat="server"` zu HTML Server-Steuerelementen konvertierten `<a>`-Tags. Diese enthalten jeweils ein spezielles Attribut, `onServerClick`. Der Parameter bezeichnet eine Methode, mit der Sie auf das Klickereignis reagieren möchten:

Wie es funktioniert

```
<a runat="server" id="A0003" onServerClick="check_Click"></a>
```

Der Link ist vorerst leer, er wird im Beispiel erst zur Laufzeit mit Daten gefüllt. Dazu ist ein Zugriff auf die Kollektion der HTML Server-Steuerelemente erforderlich. Da es sich um ein Formular handelt, interessieren hier nur die Steuerelemente, die im Formular sind:

```
Dim c As ControlCollection = myform.Controls
```

Alle Steuerelemente werden mit einer Schleife durchlaufen:

```
For i = 0 To c.Count - 1
```

Jedes einzelne `Control`-Objekt wird dann entnommen:

```
Dim cc As Control = c(i)
```

Nun ist festzustellen, ob es sich um ein `<a>`-Tag handelt, Typ `HtmlAnchor`:

```
If TypeOf cc Is HtmlAnchor Then
```

Zum Zugriff muss eine Typumwandlung vorgenommen werden:

```
Dim anchor As HtmlAnchor = CType(cc, HtmlAnchor)
```

Da dem passenden Tag die Bezeichnung zugeordnet werden soll, wird nun die ID des Tags mit der ID des Array verglichen:

```
If anchor.ID = ProduktArr(k, ProdID) Then
```

Stimmen beide überein, wird der Text zugewiesen:

```
anchor.InnerText = ProduktArr(k, ProdName) & " (" & ProduktArr(k, ProdPreis) + ")"
```

Den Index auf das Array bestimmt die Hilfsvariable *k*, die mit jeder erfolgreichen Zuweisung um eins erhöht wird:

```
k = k + 1
```

Ereignisbehandlung

Nun ist die Seite vollständig und wird an den Browser gesendet. Jeder Klick auf einen Link führt zum Aufruf der Methode `check_Click`. Diese müssen Sie selbst definieren. Sie hat, wie alle anderen Ereignisbehandlungsmethoden auch, die Parameter `Sender` und `EventArgs`. Außerdem wird mit `Handles` definiert, auf welche Ereignisse reagiert werden soll:

```
Public Sub check_Click(ByVal sender As Object,   
                        ByVal e As EventArgs)   
    Handles A0001.ServerClick,   
            A0002.ServerClick,   
            A0003.ServerClick
```

Die Variable `sender` enthält in diesem Fall das Objekt des Steuerelements, welches das Ereignis ausgelöst hat. Es wird nun explizit in ein `HtmlAnchor` konvertiert:

```
Dim link As HtmlAnchor = CType(sender, HtmlAnchor)
```

Nun kann auf die Eigenschaften und Methoden zugegriffen werden, die dieses Objekt enthält. Die folgende Zuweisung erzeugt eine Bestätigung des Links im HTML Server-Steuerelement `Bestaetigung`, das am Ende als `<div>`-Tag definiert wurde:

Bestaetigung.InnerHtml="Artikel " & Link.InnerText & " mit ID " & Link.ID & " wurde gewählt."

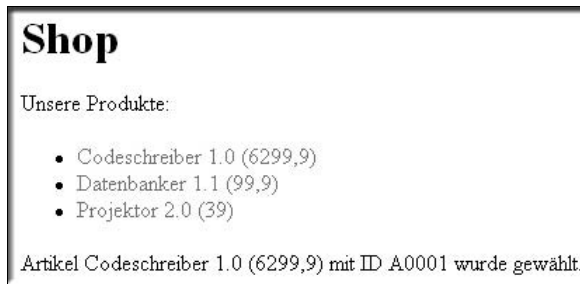


Abbildung 6.9: Nutzung des Anzeigestatus mit Links (AnchorState.aspx)

Zum Verständnis ist ein Blick in den Quelltext der HTML-Seite sinnvoll. Dort finden Sie das am Anfang erwähnte Verfahren mit JavaScript in der Anwendung:

```
<body ms_positioning="gridlayout">
  <h1>
    Shop
  </h1>
  Unsere Produkte:
  <form name="myform" method="post" action="AnchorState.aspx"
id="myform">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTYxODcyNzYxODt0PDtsPGk8MT47aTwzPjs+02w8dDw7
bDxpPDE+02k8Mz47aTw1Pjs+02w8dDxwPGw8aW5uZXJodG1s0z47bD
xDb2R1c2NocmVpYmVyIDEuMCAoNjI5OSw5MCK7Pj47Oz47dDxwPGw8
aW5uZXJodG1s0z47bDxEYXR1bmJhbmt1ciAxLjEgKDk5LDkwKTs+Pjs7Pj
t0PHA8bDxpbm5lcmh0bWw7PjtsPFByb2pla3RvciAyLjAgKDM5LDAwKTs
+Pjs7Pjs+Pjt0PHA8bDxpbm5lcmh0bWw7PjtsPEFydGlrZWwgQ29kZXNja
HJlaWJlciAxLjAgKDYyOTksOTApIG1pdCBJRChMDAwMSB3dXJkZSBn
ZXcmYXVtbFw7aGxOLjs+Pjs7Pjs+Pjs+DMGUCj6qjHZLNGuwz6QYGMtxkvQ=" />

<script language="javascript">
<!--
  function __doPostBack(eventTarget, eventArgument) {
    var theform = document.myform;
    theform.__EVENTTARGET.value = eventTarget;
    theform.__EVENTARGUMENT.value = eventArgument;
    theform.submit();
  }
}
```

```
// -->
</script>

<ul>
  <li>
    <a id="a0001" href="javascript:__doPostBack('a0001','')"> ⚡
      Codeschreiber 1.0 (6299,90)</a>
  <li>
    <a id="a0002" href="javascript:__doPostBack('a0002','')"> ⚡
      Datenbanker 1.1 (99,90)</a>
  <li>
    <a id="a0003" href="javascript:__doPostBack('a0003','')"> ⚡
      Projektor 2.0 (39,00)</a>
  </li>
</ul>
</form>
<div id="bestaetigung">Artikel Codeschreiber 1.0 (6299,90) ⚡
  mit ID a0001 wurde gew&auml;hlt.</div>
</body>
```

Automatisches Senden: Die AutoPostBack-Funktion

Wenn ein Ereignis sofort bedient werden soll, muss das gesamte Formular gesendet werden. Dazu ist es clientseitig notwendig, den entsprechenden Auslöser, beispielsweise einen Mausklick, mit einer JavaScript-Funktion zu verbinden, die das Senden zuverlässig veranlasst. Bei Steuerelementen können Sie dies aktivieren, indem das Attribut `AutoPostBack` auf `True` gesetzt wird.

Serverseitig Serverseitig wird der Vorgang »Formular wurde gesendet« allgemein erkannt, also unabhängig von einem konkreten Ereignis, dessen Ereignisbehandlungsmethode erst später ausgeführt wird. Innerhalb der Seite steht Ihnen die Eigenschaft `IsPostBack`, definiert in der Klasse `Page`, zur Verfügung. Sie können damit überprüfen, ob der Aufruf durch ein Formular oder durch direkten Aufruf der Seite erfolgte.

Clientseitig Aber wie funktioniert das clientseitig?

Zum einen finden Sie den Aufruf der JavaScript-Funktion, die immer den Namen `__doPostBack` trägt:

```
<a id="A0003" href="javascript:__doPostBack('A0003','')">
```

Die Funktionsdefinition selbst enthält keine Besonderheiten. Einige Steuerelemente können zusätzlich Kommandos versenden, um sich selbst besser zu identifizieren. Das trifft vor allem für Schalt-

flächen zu. Die beiden Parameter der Funktion werden als `eventTarget` und `eventArgument` bezeichnet. Die Übertragung erfolgt, indem die Werte an versteckte Felder übergeben werden:

```
theform.__EVENTTARGET.value = eventTarget;  
theform.__EVENTARGUMENT.value = eventArgument;
```

Das Attribut `action` wurde bereits im `<form>`-Tag definiert, verbleibt also nur die Aufgabe, das Formular zu versenden:

```
theform.submit();
```

Das Objekt vom Typ `HtmlAnchor` kann ein Ereignis `onServerClick` verarbeiten. Dies können Sie auch für `<button>` und `<input type="button">` einsetzen. Ein zweites Ereignis ist, `onServerChange`, das auftritt, wenn der Inhalt eines Objekts geändert wurde. Anwendbar ist dies bei folgenden HTML Server-Steuerelementen:

**onServerChange
onServerClick**

- ▶ `<input type="text">`
- ▶ `<input type="password">`
- ▶ `<input type="checkbox">`
- ▶ `<input type="radio">`
- ▶ `<input type="image">`
- ▶ `<input type="hidden">`
- ▶ `<textarea>`
- ▶ `<select>`

Das Ereignis wird ausgelöst, wenn der Inhalt eines Feldes des vorhergehenden Formulars mit dem aktuellen Formular nicht übereinstimmt. Auch die Methode, die `onServerChange` bedient, ist vom Typ `void` und kann zwei Parameter vom Typ `object` und `EventArgs` verarbeiten.

6.3.4 Gestalterische Elemente

Auf die Möglichkeit, Attribute von HTML Server-Steuerelementen programmtechnisch zu verändern, wurde bereits hingewiesen. Das ist im Detail nicht immer einfach, wenn es sich um Attribute handelt, die selbst wieder in viele kleinere Einheiten aufgelöst werden können. Dies trifft für Stylesheets zu. Das folgende Steuerelement enthält mehrere Style-Attribute (mit `runat="server"` `id="btn"` wird es zum Server-Steuerelement):

```
<input type="button"
      style="font-weight:bold; font-size:24pt; color:green"/>
```

Attribut style Wenn Sie nun auf das Attribut `style` zugreifen, erhalten Sie eine Zeichenkette, die mühevoll zerlegt werden muss. Das erledigt ASP.NET für Sie jedoch automatisch. Das folgende Beispiel zeigt, wie eine Schaltfläche einfach gestaltet werden kann:

```
<body>
  <h1>Schaltfläche gestalten:</h1>
  <form runat="server" >
    <input type="button" id="btn"
          runat="server" onClick="Btn_Click"/>
  </form>
  <div id="Bestaetigung" runat="server"/>
</body>
```

Listing 6.10: Gestaltung einer Schaltfläche mit CSS (ButtonStyle.aspx)

```
Public Class ButtonStyle
  Inherits System.Web.UI.Page
  Protected WithEvents btn As HtmlInputButton
  Protected Bestaetigung As HtmlContainerControl

  Private Sub Page_Load(ByVal sender As Object,
                        ByVal e As EventArgs) _
    Handles MyBase.Load
    btn.Value = "Ich bin ein schicker Button"
    btn.Style("font-family") = "Arial"
    btn.Style("font-weight") = "bold"
    btn.Style("color") = "green"
    btn.Style("font-size") = "24pt"
  End Sub

  Public Sub Btn_Click(ByVal sender As Object,
                      ByVal e As EventArgs) _
    Handles btn.ServerClick
    Dim btn As HtmlInputButton = CType(sender, HtmlInputButton)
    Bestaetigung.InnerHtml =
      "Meine Gestaltung basiert auf:<br />"
    Dim keys As IEnumerable = btn.Style.Keys.GetEnumerator()
    Dim key As String
    While keys.MoveNext()
      key = CType(keys.Current, String)
      Bestaetigung.InnerHtml += key + " = '" + btn.Style(key)
      + "'<br />"
    End While
  End Sub
End Class
```

```

    End While
  End Sub
End Class

```

Listing 6.11: Definition der Schaltfläche und Klickbehandlungsmethode (ButtonStyle.aspx.vb)

Hier wird die Zuweisung der einzelnen CSS-Attribute innerhalb von `Page_Load` erledigt. Die entsprechende Eigenschaft heißt `Style`.

Schaltfläche gestalten:

Ich bin ein schicker Button

```

Meine Gestaltung basiert auf
font-family='Arial'
font-weight='bold'
color='green'
font-size='24pt'

```

Abbildung 6.10: Gestaltung von HTML Server-Steuerelementen (button_style.aspx)

Das Klickereignis übergibt die Schaltfläche als Objekt. Damit es weiterverarbeitet werden kann, ist eine Typkonvertierung erforderlich:

```
Dim btn As HtmlInputButton = CType(sender, HtmlInputButton)
```

Verwechseln Sie dies nicht mit `HtmlButton`; diese Klasse bedient `<button>`-Tags, die im Sprachumfang von HTML 4 definiert sind und nur selten zum Einsatz kommen, weil zur praktischen Nutzung zwingend JavaScript erforderlich ist.



Wie es funktioniert

Die `Style`-Eigenschaft ist eine Aufzählung. Innerhalb der Methode `clickBtn`, die ein Anklicken dieser Schaltfläche verarbeitet, werden zur Demonstration alle Elemente ausgelesen. Das geht am einfachsten, indem zuvor die Schlüssel der Kollektion zum Typ `IEnumerator` konvertiert werden:

Style-Attribute verarbeiten

```
Dim keys As IEnumerator = btn.Style.Keys.GetEnumerator()
```

Die Aufzählung wird mit einer `While`-Schleife durchlaufen, wobei bei jedem Durchlauf das nächste Element mit `MoveNext` ausgewählt wird:

Aufzählung durchlaufen

```
While keys.MoveNext()
```

Dann wird der aktuelle Inhalt des Elements mit `Current` ermittelt:

```
key = CType(keys.Current, String)
```

Nun wird dem Ausgabe-Tag der Name und der Wert des `Style-Attributes` zugewiesen:

```
Bestaetigung.InnerHtml += key + " = '" + btn.Style(key) + "'<br />"
```

Das Ergebnis war bereits in der Abbildung 6.10 zu sehen. Ausführliche Informationen zum Umgang mit Aufzählungen und Kollektionen finden Sie in Abschnitt 4.3, »Aufzählungen und Kollektionen« ab Seite 222.

6.3.5 Dateien per HTTP hochladen (Upload)

Das Hochladen von Dateien ist ein oft genutzter Bestandteil von interaktiven Webseiten. Selbstverständlich bietet ASP.NET hier eine optimale Unterstützung. Zentraler Punkt ist das HTML Server-Steuerelement `HtmlInputFile` und die Klasse `HttpPostedFile`.

Vorbereitung des Formulars

enctype-Attribut

Bevor Dateien hochgeladen werden können, müssen Sie das `<form>`-Tag um ein weiteres Attribut ergänzen:

```
<form enctype="multipart/form-data" runat="server">
```

Damit wird dem Browser mitgeteilt, dass die sich die Art der Übertragung der Daten im Körper der HTTP-Nachricht von der eines normalen Formulars unterscheidet. Das ist notwendig, weil die übertragenen Dateien binäre Daten enthalten können und die normale Kodierung, wie sie bei Formulardaten verwendet wird, nur unzureichenden Schutz bietet. Eine genaue Definition des Vorgangs finden Sie in der RFC 1867, »Formularbasiertes Hochladen von Dateien«.

Der zweite Punkt besteht darin, ein speziellen Eingabefeld zu erzeugen. Dies erfolgt entweder programmtechnisch über die Klasse `HtmlInputFile` oder über ein entsprechend gekennzeichnetes Steuerelement:

```
<input type="file" id="upload" runat="server"/>
```

Dieses Feld unterscheidet sich etwas von anderen Eingabefeldern. Zum einen haben Sie auf die Gestaltung nur wenig Einfluss. Neben dem Eingabebereich erscheint immer eine Schaltfläche mit der Beschriftung bzw. bei englischem Betriebssystem . Sie können diese Beschriftung nicht ändern. Ebenso

kann der Inhalt nicht mit `value` vorausgefüllt oder per JavaScript beeinflusst werden. Dies ist keine fehlenden Funktionen, sondern Sicherheitsmaßnahmen. Eine Manipulation des Feldes könnte nämlich über den waren Einsatz hinwegtäuschen und eine verdeckte Vorauswahl könnte verwendet werden, um ohne weiteres Zutun gezielt Dateien vom Computer des Benutzers zu beschaffen. Da sich die Schaltfläche nicht ändern lässt und es keinen Weg gibt, daran vorbei das Feld zu füllen – abgesehen von der direkten Eingabe – ist das Verfahren relativ sicher.

Das bedeutet natürlich auch, dass der Erhalt des Inhalts mit dem Anzeigestatus nicht funktionieren kann.

Dateien hochladen und auswerten

Der Vorgang des Hochladens besteht aus zwei Schritten:

1. Ausführung des Ladevorgangs
2. Kopieren der Datei an ihren endgültigen Bestimmungsort

Der Kopiervorgang ist notwendig, weil ASP.NET die empfangenen Daten erstmal in einem temporären Verzeichnis ablegt. Der Zugriff auf das Dateisystem ist also immer notwendig. Sie müssen dazu folgenden Namensraum einbinden: `System.IO`.

**System.IO
erforderlich**

Das folgende kleine Programm zeigt das Prinzip. Es empfängt Dateien bis maximal 1 MByte Größe und legt Sie in einem Unterverzeichnis `data\upload` ab. Anschließend werden verschiedene Informationen über die Datei gesammelt und angezeigt sowie eine Liste aller bereits hochgeladenen Dateien ausgegeben.

```
<h1>Dateien hochladen</h1>
<form id="HttpUploadFile" method="post" runat="server"
    enctype="multipart/form-data">
    <input type="file" id="HttpUpload" runat="server"/>
    <input type="submit" value="Jetzt übertragen"/>
</form>
<asp:Label Runat="server" ID="FileResult"/>
<asp:DataGrid Runat="server" ID="FileList"/>
```

Listing 6.12: Formular zum Hochladen von Dateien (HttpUploadFile.aspx)

Außer dem zuvor beschriebenen Formulartyp ist hier nichts Besonderes zu finden. Die Dateiliste wird über ein nicht weiter gestaltetes `DataGrid`-Steuerelement ausgegeben. Dies dient hier nur der Demonstration und hat keinen Einfluss auf die Funktion. Eine

nähere Betrachtung dazu folgt in Abschnitt 6.6, »Vorlagengebundene Daten-Steuerelemente« ab Seite 566.

Der spannende Teil steckt in der Code-Datei:

```
Public Class HttpUploadFile
    Inherits System.Web.UI.Page
    Protected FileResult As Label
    Protected FileList As DataGrid
    Protected HttpUpload As HtmlInputFile

    Private sb As StringBuilder = New StringBuilder()

    Private Sub Page_Load(ByVal sender As Object, 
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Page.IsPostBack Then
            Dim target As String = Server.MapPath("data/upload") + "\"
            If Not HttpUpload.PostedFile Is Nothing Then
                Dim pf As HttpPostedFile = HttpUpload.PostedFile
                If pf.ContentLength > 0 
                    And CType(pf.ContentLength, Double) < Math.Pow(2, 20)
                Then
                    Dim source As String = Path.GetFileName(pf.FileName)
                    pf.SaveAs(target + source)
                    sb.Append("Datei erfolgreich geladen:<br />")
                    sb.AppendFormat("Dateiname: {0}<br />", pf.FileName)
                    sb.AppendFormat("Größe: {0}<br />", pf.ContentLength)
                    sb.AppendFormat("Typ: {0}<br />", pf.ContentType)
                    FileResult.Text = sb.ToString()
                Else
                    FileResult.Text = "Datei zu groß oder zu klein: " 
                        + pf.ContentLength.ToString()
                End If
            End If
            FileList.DataSource = Directory.GetFiles(target)
            FileList.DataBind()
        Else
            FileResult.Text = "Noch keine Datei übertragen"
        End If
    End Sub
End Class
```

Listing 6.13: Verarbeitung von hochgeladenen Dateien (HttpUploadFile.aspx.vb)

Wie es funktioniert

Dieses Programm hat mehrere Aufgaben. Zum einen soll geprüft werden, ob überhaupt Daten übertragen wurden. Wenn das der Fall ist, sollen alle Dateien größer als 1 MByte abgelehnt werden. Leider können Sie dies erst nach der Übertragung prüfen, die bei

ISDN einige Zeit in Anspruch nimmt. Lassen Sie sich hier nicht von der Geschwindigkeit auf einem Entwicklungssystem täuschen.

Der Ablauf beginnt mit der Feststellung, ob es sich um einen *PostBack*-Vorgang handelt:

```
If Page.IsPostBack Then
```

Ist das der Fall, wird zuerst der Zielpfad berechnet:

```
Dim target As String = Server.MapPath("data/upload") + "\"
```

Dann wird geprüft, ob die Datei übertragen wurde. Wenn die Übertragung erfolgreich war, existiert ein entsprechendes Objekt vom Typ `HttpPostedFile`.

```
If Not HttpUpload.PostedFile Is Nothing Then
```

Nun wird daraus ein reguläres `HttpPostedFile`-Objekt erstellt. Das ist nicht zwingend notwendig, erleichtert aber den Zugriff auf die Dateieigenschaften etwas:

```
Dim pf As HttpPostedFile = HttpUpload.PostedFile
```

Jetzt folgt die Prüfung der Dateigröße; 2^{20} entspricht exakt 1 MByte:

```
If pf.ContentLength > 0
```

```
And pf.ContentLength < Math.Pow(2, 20) Then
```

Stimmen alle Bedingungen, kann die Datei an ihren Bestimmungsort transportiert werden. Dafür ist aus dem temporären Dateinamen der blanke Name zu bestimmen:

```
Dim source As String = Path.GetFileName(pf.FileName)
```

Dann wird die Datei an diesen Ort kopiert:

```
pf.SaveAs(target + source)
```

Die temporären Daten selbst werden im Rahmen der Garbage Collection entfernt. Sie müssen nicht befürchten, dass dadurch Speicherplatz verloren geht.



Im Beispiel werden nun noch einige Informationen über die Datei ermittelt. Die Zusammenfassung erfolgt in einem `StringBuilder`-Objekt (hierfür ist der Namensraum `System.Text` erforderlich).

Informationen dazu finden Sie im Abschnitt »Die Klassen `StringWriter` und `StringBuilder` im Detail« ab Seite 262. Beachtenswert ist auch die Anzeige des Dateityps. Dies erfolgt im MIME-Format. Hintergrundinformationen dazu sind im Exkurs »Was ist eigentlich MIME?« auf Seite 764 zu finden.

Dateien hochladen

Datei erfolgreich geladen:
 Dateiname: C:\docprocess\asp.net\pics\cdr\ccvarianten.psd
 Größe: 185160
 Typ: application/octet-stream

Item
c:\inetpub\wwwroot\dasbuch\csharpwebform\data\upload\4-1_Bild1.tif
c:\inetpub\wwwroot\dasbuch\csharpwebform\data\upload\AcceptLaguageFallback.jpg
c:\inetpub\wwwroot\dasbuch\csharpwebform\data\upload\ccvarianten.psd
c:\inetpub\wwwroot\dasbuch\csharpwebform\data\upload\Skizze1.jpg

Abbildung 6.11: Verhalten des Hochlade-Programms bei mehrfacher Nutzung

Eine etwas erweiterte Anwendung für das Hochladen von Dateien finden Sie in Abschnitt 7.5.6, »Weitere Funktionen der Applikation »Server Explorer«« ab Seite 703, dort ist die Anzahl der Dateien variabel, indem dynamisch weitere Eingabefelder vom Typ `HttpInputFile` erzeugt werden.

Direkter Zugriff auf binäre Dateidaten

Eine weitere Eigenschaft ist für die Praxis interessant: `InputStream`. Damit erhalten Sie direkten Zugriff auf den Datenstrom. `InputStream` ist vom Typ `System.IO.Stream`. Viele andere Klassen verfügen über Methoden und Eigenschaften, die Daten aus einem Stream annehmen. So können Sie empfangene Bilddaten gleich im entsprechenden Format aufnehmen und weiterverarbeiten – ohne den Umweg über das Dateisystem (*HttpUpload* ist das `HtmlInputFile`-Objekt):

```
Dim st As System.IO.Stream = HttpUpload.PostedFile.InputStream
Dim img As System.Drawing.Image ←
    = System.Drawing.Image.FromStream(st)
Dim width As Integer = img.Width
Dim height As Integer = img.Height
```

Dieses Beispiel ermittelt Breite und Höhe eines Bildes. Mit der gesamten GDI+-Bibliothek kann nun die Verarbeitung der Daten erfolgen.

6.4 Web Server-Steuerelemente (Web Server Controls)

Web Server-Steuerelemente sind ebenso wie die HTML Server-Steuerelemente für das Erzeugen von HTML-Code zuständig. Es gibt jedoch wesentliche Unterschiede, die den Einsatz der einen oder anderen Version sinnvoll erscheinen lassen.

Web Server-Steuerelemente sind teilweise komplexer als HTML Server-Steuerelemente und erlauben eine echte Abstraktion von der HTML-Ebene. Das bedeutet, dass die Definition generell mit eigenen Tags und unabhängig vom Objektmodell von HTML erfolgt. Dadurch kann ein Web Server-Steuerelement mehr als ein HTML-Tag generieren bzw. verwalten. Sie sind insgesamt reichhaltiger und strenger typisiert.

Hohe Abstraktion

HTML kennt per se keine Datentypen. Das führt dazu, dass Attribute bei HTML Server-Steuerelementen immer als Zeichenkette betrachtet werden. Tatsächlich sind aber bestimmte Attribute durchaus an bestimmte Datentypen bindbar, wie beispielsweise die Breite einer Tabellenzelle (`width="43"`, Typ `Unit`). Hier sind entweder ganze Zahlen oder Prozentwerte erlaubt, aber mit Sicherheit nie eine Zeichenkette. Web Server-Steuerelemente besitzen für alle in HTML zulässigen Attribute einzelne Eigenschaften. Insofern ist die Programmierung damit strenger am Klassen- und Typkonzept des NET-Frameworks ausgerichtet. Von Vorteil ist auch die Verfügbarkeit komplexer Objekte, wie ganzer Kalender, die intern aus Tabellen und Links bestehen.

Datentypen

Bei den HTML Server-Steuerelementen wurde gezeigt, dass Standard-HTML-Tags erzeugt werden. Das funktioniert bei den HTML 3.2 entstammenden Tags mit jedem Browser. Komplexere Objekte sind dagegen nicht so leicht allein mit HTML 3.2 zu erstellen. Manches Tag ist nicht in jedem Browser gleichartig implementiert. Außerdem kommt manchmal JavaScript zum Einsatz, was auch nicht problemlos in allen Browsern funktioniert. Web Server-Steuerelemente enthalten deshalb eine automatische Browser-Erkennung. Je nach Browsertyp wird dann automatisch

Browser-Erkennung

unterschiedlicher Code erzeugt. Dies ist sehr vorteilhaft, denn Sie müssen sich keine Gedanken darüber machen, dass durch den Einsatz von ASP.NET Benutzer bestimmter Browser eventuell ausgeschlossen wären.

6.4.1 Übersicht über die Web Server-Steuerelemente

Dieser Abschnitt zeigt eine Übersicht der verfügbaren Steuerelemente und deren Einordnung im .NET-Framework. Dies erleichtert das Auffinden der Methoden und Eigenschaften in der Referenz. Allerdings ist dies nicht ganz so einfach wie bei den HTML Server-Steuerelementen. Zwischen den Klassen des Namensraumes `System.Web.UI.WebControls` und dem Rest des Frameworks bestehen vielfältige Abhängigkeiten und Beziehungen. Die Anzahl der verfügbaren Elemente ist auch deutlich höher. Es ist also etwas Systematik erforderlich, um der schierenden Menge an Möglichkeiten Herr zu werden. Einteilen kann man die Web Server-Steuerelemente nach ihrer Komplexität und der Aufgabe, die sie erfüllen:

► **Web Form-Steuerelemente** (Web Form Controls)

Diese Gruppe enthält alle Steuerelemente, die in Formularen zum Einsatz kommen. Sie ist den bereits im letzten Abschnitt vorgestellten HTML Server-Steuerelementen ähnlich. Oft werden dieselben Tags erzeugt. Die Gruppe lässt sich feiner unterteilen:

► **Web Server-Steuerelemente** (Web Controls)

Dies sind alle normalen Eingabe-Tags, Label und Tabellen-Tags.

► **Text-Steuerelemente** (Text Controls)

Hiermit erfolgt die Ausgabe von Text.

► **Listen-Steuerelemente** (List Controls)

Listen-Tags sind deshalb von besonderer Bedeutung, weil sie wiederholende Teile enthalten; ideal zum Anzeigen der Ergebnisse von Datenbankabfragen.

► **Daten-Steuerelemente** (Data Controls)

Direkt zur Ausgabe von Datenmengen aus Datenbanken sind diese Steuerelemente geeignet.

- ▶ **Kontroll-Steuerelemente** (Validation Controls)
Daten aus Formularen müssen vielfältig geprüft werden. Mit den Steuerelementen dieser Gruppe ist das besonders einfach zu erledigen.
- ▶ **Komplexe Steuerelemente** (Rich Controls)
Diese Gruppe umfasst komplexere Steuerelemente, die beispielsweise Banner verwalten oder Kalender darstellen.

Die folgenden Tabellen zeigen die Mitglieder der einzelnen Gruppen.

Name des Steuerelements	Erzeugtes HTML-Tag
<asp:hyperlink>	<a>
<asp:linkbutton>	<a>
<asp:image>	
<asp:button>	<input type="button">
<asp:textbox>	In Abhängigkeit von Attributen: <input type="text">, <input type="password">, oder <textarea>
<asp:checkbox>	<input type="checkbox">
<asp:radiobutton>	<input type="radio">
<asp:imagebutton>	<input type="image">
<asp:table>	<table>
<asp:tablerow>	<tr>
<asp:tablecell>	<td>

Tabelle 6.4: Web Server-Steuerelemente

Informationen zum dynamischen Erstellen von Tabellen finden Sie in vielen Listings im Buch. Die Anwendung ist nicht allzu schwierig, sodass Ihnen eine langweilige Ausbreitung aller Eigenschaften und Methoden erspart bleiben soll. Umfangreichere Anwendungen finden Sie in Abschnitt 7.5.5, »Toolbar« ab Seite 699 und 9.3.5, »Aktualisieren einer Datenbank mit CommandBuilder« ab Seite 904.

Tabellen

Text-Steuerelement	Erzeugtes HTML-Tag
<asp:panel>	<div>
<asp:label>	
<asp:literal>	Einfacher Text, der verändert werden kann
<asp:placeholder>	An dieser Stelle kann neuer Text erzeugt werden

Tabelle 6.5: Übersicht Text-Steuerelemente

Folgende Listen-Steuerelemente kennt ASP.NET 1.0 (mit der Version 1.1 des Frameworks kamen noch keine neuen Elemente hinzu):

Listen-Steuerelement	Erzeugtes HTML-Tag
<asp:dropdownlist>	<select size="1">
<asp:listbox>	<select>
<asp:checkboxlist>	Mehrere Elemente vom Typ <input type="checkbox">
<asp:radiobuttonlist>	Mehrere Elemente vom Typ <input type="radio">
<asp:listitem>	Expliziter Zugriff auf <option>
<asp:repeater>	Wiederholt das enthaltene Element

Tabelle 6.6: Übersicht Listen-Steuerelemente

Die Daten-Steuerelemente dienen der Ausgabe von Datensätzen, wie sie Datenbanken entnommen werden können. Dieser Teil wird zusammen mit dem Datenbankzugriff in den Abschnitten 6.6, »Vorlagengebundene Daten-Steuerelemente« ab Seite 566 und 9.5, »Die Datenbindung an Steuerelemente« ab Seite 926 beschrieben.

Es folgt eine Übersicht über die Kontroll-Steuerelemente. Diese Elemente dienen der direkten Überprüfung von Benutzereingaben, sind also nur im Zusammenhang mit Eingabefeldern sinnvoll anwendbar.

Name des Steuerelements	Bedeutung
<asp:RequiredFieldValidator>	Prüft, ob ein Feld ausgefüllt worden ist
<asp:RangeValidator>	Prüft, ob der erfasste Werte innerhalb eines Wertebereiches ist
<asp:CompareValidator>	Vergleicht zwei Felder

Tabelle 6.7: Übersicht Kontroll-Steuerelemente

Name des Steuerelements	Bedeutung
<asp:RegularExpressionValidator>	Prüft den Inhalt mit Hilfe eines regulären Ausdrucks
<asp:CustomValidator>	Prüft den Inhalt mit einer selbstdefinierten Methode
<asp:ValidationSummary>	Zeigt alle Fehlermeldungen auf Grund der Feldprüfungen an

Tabelle 6.7: Übersicht Kontroll-Steuerelemente (Forts.)

Die Übersicht über komplexe Steuerelemente zeigt, welches Potenzial in Steuerelementen steckt. Vorgestellt werden diese Elemente nur überblicksartig am Ende des Kapitels.

Komplexes Steuerelement	Beschreibung
<asp:adrotator>	Dient der Verwaltung von Bannern
<asp:calender>	Ein sehr umfangreiches Control zur Anzeige von Kalendern
<asp:xml>	Ausgabe von XML und optional Transformation mit XSLT

Tabelle 6.8: Übersicht Komplexe Steuerelemente

Alle Steuerelemente stammen aus dem Namensraum `System.Web.UI.Control`. Da die vielen Klassen unzählige Eigenschaften und Methoden enthalten, ist ein Blick auf die Hierarchie sehr hilfreich, denn die Definitionen in höheren Klassen wiederholen sich in allen tiefer liegenden. Wenn Sie sich eine Basisklasse anschauen, kennen Sie damit schon eine Großteil der Leistungen der konkreten Klassen.

Eine Übersicht finden Sie in der Einleitung zu diesem Kapitel in Abbildung 6.2.

6.4.2 Einsatzprinzipien und Basiseigenschaften

Die Vielfalt der Elemente systematisch darzustellen, ist ein mühsames und wenig hilfreiches Unterfangen. Ähnlich wie bei den HTML Server-Steuerelementen ist es entscheidend, das Prinzip zu verstehen. Mit Hilfe der Online-Referenz können Sie sich dann jedes gerade benötigte Element leicht erschließen. In der Übersicht der Klassenhierarchie haben Sie bereits die Namen der Klassen

Aufbau der Tags

kennen gelernt. Die Elemente, die im HTML-Teil der Seite eingesetzt werden können, entsprechen diesen Namen. So implementieren Sie ein Objekt der Klasse `Label` folgendermaßen:

```
<asp:label id="TextAusgabe" runat="server" />
```

Diese Syntax stammt aus der XML-Welt. Der konkrete Name setzt sich aus dem XML-Namensraum »asp« und dem Namen der Klasse »label« zusammen. Wie schon bei den HTML Server-Steuerelementen wird das Objekt dann im Programmteil bereitgestellt, wenn es mit dem Attribut `runat="server"` versehen wurde.



Beachten Sie, dass die Web Server-Steuerelemente von ASP.NET nur dann erkannt werden, wenn sie das Attribut `runat="server"` tragen. Ohne diese Angabe werden die Tags völlig ignoriert und landen unverändert in der HTML-Ausgabe. Da der Browser damit nichts anfangen kann, blendet er sie in der Darstellung aus. Eine Fehlermeldung erscheint weder auf der einen noch auf der anderen Seite.

Das Attribut `id` bestimmt, unter welchem Namen das Objekt entsteht. Der Zugriff ist, wie nachfolgend gezeigt, möglich:

```
TextAusgabe.Text = "Dieser Text wird angezeigt"
```

Wenn sie alle Eigenschaften eingestellt haben und der Seiten-erstellungsprozess abgeschlossen ist, wird das fertige HTML an den Browser gesendet. Im Beispiel würde dies folgendermaßen aussehen:

```
<span id="TextAusgabe">Dieser Text wird angezeigt</span>
```

Web Server-Steuerelemente haben noch einige Basiseigenschaften, die die bei HTML Server-Steuerelementen beschriebenen Möglichkeiten erweitern.

Zugriff auf die Feldwerte

Request.Form

Die Nutzung der `Form`-Kollektion (`Request.Form`) ist zwar weiter möglich, aber nicht unbedingt empfehlenswert. Je nach Typ des Web Server-Steuerelementes werden verschiedene Eigenschaften für den Zugriff auf die Feldwerte angeboten. Damit ist ein direkter Zugriff auf die Inhalte möglich. ASP.NET »füllt« die Eigenschaften automatisch mit den per HTTP-POST empfangenen und aus dem Anzeigestatus berechneten Daten.

Der Zugriff auf die Feldwerte erfolgt bei Textfeldern über die Eigenschaft `Text`. Listen, also `DropDownList` oder `ListBox`, werden über die Listenmitglieder angesprochen. Der erste in der Liste ausgewählte Index wird in der Eigenschaft `SelectedIndex` übergeben, das Listenobjekt selbst in `SelectedItem`. Bei Kontrollkästchen (`CheckBox`) und Optionsfeldern (`RadioButton`) ist die Eigenschaft `Checked` gleich `True`, wenn das Element ausgewählt wurde.

Textfelder

Automatisches Senden von Formularen

Im weitesten Sinne geht es bei fast allen Web Server-Steuerelementen um Formulare. Bislang war es immer notwendig, nach dem Ausfüllen eines Formulars dieses mit einer Sendeschaltfläche abzusenden. Manchmal ist das lästig und unnötig, beispielsweise wenn jede Änderung an einer Option sofort zu einem Ereignis führen soll. Die Funktion, die das erledigt, heißt *AutoPostBack*. Dabei kommt JavaScript zum Einsatz, was Sie aber nicht selbst implementieren müssen. Dies erledigt die ASP.NET-Komponente für Sie. Standardmäßig steht dieses Verhalten bei folgenden Objekten zur Verfügung: `CheckBox` (Kontrollkästchen), `RadioButton` (Optionsfeld) und `TextBox` (Eingabefelder). Die Funktion basiert auf den in HTML definierten `onClick`- bzw. `onChange`-Ereignissen.

AutoPostBack

*Denken Sie bei der Implementierung der *AutoPostBack*-Funktion daran, dass die Ereignisverarbeitung auf dem Server erfolgt und dass jeder Klick des Benutzers zum Absenden des Formulars führt. Unter Umständen, beispielsweise bei langsamen Verbindungen oder großen Seiten, ist dies lästig und erschwert die Bedienung mehr, als dass es nutzt.*



Ob ein Element dieses Verhalten aufweisen soll, ist einfach über das Attribut `autopostback` festzulegen, bzw. programmtechnisch über die Eigenschaft `AutoPostBack`. Das folgende Beispiel zeigt das Prinzip:

```
<html lang="de">
  <head>
    <title>RadioButton</title>
  </head>
  <body>
    <h1>Was ist die beste Programmierumgebung ☹
      f&uuml;r Webserver?</h1>
    <form runat="server" >
      <asp:radiobutton autopostback="true" ☹
        OnCheckedChanged="Btn_Click" ☹
```

```

        id="Wahl1" text="ASP.NET mit VB.NET" ␣
        groupname="Wahl" runat="server"/>
    <br/>
    <asp:radiobutton autopostback="true" ␣
        OnCheckedChanged="Btn_Click" ␣
        id="Wahl2" ␣
        text="Klassik ASP mit VBScript" ␣
        groupname="Wahl" runat="server"/>
    <br/>
    <asp:radiobutton autopostback="true" ␣
        OnCheckedChanged="Btn_Click" ␣
        id="Wahl3" text="PHP" ␣
        groupname="Wahl" runat="server"/>
    <br/>
    <asp:radiobutton autopostback="true" ␣
        OnCheckedChanged="Btn_Click" ␣
        id="Wahl4" text="Perl/CGI" ␣
        groupname="Wahl" runat="server"/>
    <br/>
    <asp:radiobutton autopostback="true" ␣
        OnCheckedChanged="Btn_Click" ␣
        id="Wahl5" text="Java" ␣
        groupname="Wahl" runat="server"/>
</form>
<asp:label id="Bestaetigung" runat="server"/>
</body>
</html>

```

Listing 6.14: Optionsfelder mit AutoPostBack abfragen (AutoPostBack.aspx)

```

Public Class AutoPostBack
    Inherits System.Web.UI.Page
    Protected WithEvents Wahl1 As RadioButton
    Protected WithEvents Wahl2 As RadioButton
    Protected WithEvents Wahl3 As RadioButton
    Protected WithEvents Wahl4 As RadioButton
    Protected WithEvents Wahl5 As RadioButton
    Protected Bestaetigung As Label

    Private Sub Page_Load(ByVal sender As Object, ␣
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Bestaetigung.Text = "Keine Auswahl getroffen"
    End Sub

    Public Sub Btn_Click(ByVal sender As Object, ␣
        ByVal e As EventArgs) _
        Handles Wahl1.CheckedChanged, ␣
        Wahl2.CheckedChanged, ␣
        Wahl3.CheckedChanged, ␣

```

```

        Wahl4.CheckedChanged, _
        Wahl5.CheckedChanged
    Dim currentRadio As RadioButton = CType(sender,
RadioButton)
    Bestaetigung.Text = currentRadio.Text
End Sub
End Class

```

Listing 6.15: Code-Datei zur Behandlung des Autopostback-Ereignisses (AutoPostBack.aspx.vb)

Im HTML-Teil wird eine Gruppe von Optionsfeldern definiert. Die Gruppierung findet über das Attribut `groupname` statt. Die Methode `Btn_Click`, die auf ein Klickereignis reagieren soll, wird mit `onCheckedChanged` festgelegt. Außerdem wird natürlich mit `autopostback="true"` sichergestellt, dass eine Änderung der Option zum Senden des Formulars führt. Viel Code ist hier ohnehin nicht notwendig. Die Ereignisbehandlungsmethode empfängt das Element als Objekt über den Parameter `sender`. Da bekannt ist, dass es sich um Optionsfelder handelt, wird eine explizite Typkonvertierung ausgeführt:

```
Dim currentRadio As RadioButton = CType(sender, RadioButton)
```

Zur Bestätigung des Vorgangs soll auf die Eigenschaft `Text` zugegriffen werden:

```
Bestaetigung.Text = currentRadio.Text
```

Hier kommt ein Steuerelement vom Typ `Label` zum Einsatz, dass als HTML zu `<span>` mutiert. Ein Blick auf den HTML-Code ist an dieser Stelle ohnehin spannender. Zuerst aber das Bild der fertigen Seite:

Was ist die beste Programmierungsumgebung für Webserver?

```

☒ ASP.NET mit C#
☐ Klassik ASP mit VBScript
☐ PHP
☐ Perl/CGI
☐ Java
ASP.NET mit C#

```

Abbildung 6.12: Auswahl ohne Sendeschaltfläche mit AutoPostBack

Im HTML-Code finden Sie die schon bekannte Vorgehensweise in ASP.NET. Dabei ist zu beachten, dass eine Erstauswahl nicht sinnvoll ist, weil eine Auswahl eines bereits selektierten Optionsfeldes nicht zum Auslösen des Ereignisses führt. Dies kann den Benutzer

Wie es funktioniert

Umsetzung in HTML

möglicherweise irritieren, da das Programm ihm keine explizite Rückkopplung der Aktion liefert (auch wenn es intern wie erwartet funktionieren mag). Das Formular sendet sich an dieselbe Seite zurück:

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

Der Auslöser für das Absenden des Formulars ist der Aufruf einer JavaScript-Funktion beim Auftreten des Mausereignisses `onClick`. Dies führt zu folgendem Eingabefeld:

```
<input id="Wahl1" type="radio" name="Wahl"
      value="Wahl1" checked="checked"
      onClick="__doPostBack('Wahl1','')"
      language="javascript" />
```

Interessant ist noch die Darstellung des Textes des Optionsfeldes, der an das Objekt gebunden ist und nicht – wie bei HTML sonst üblich – losgelöst auf der Seite steht:

```
<label for="Wahl1">ASP.NET mit VB.NET</label>
```

Soweit zum Prinzip. Nachfolgend werden exemplarisch einige Klassen näher vorgestellt.

6.4.3 Text auf der Seite steuern

Die Ausgabe von Textelementen, die vorerst keine Interaktion zulassen, mag kein sinnvoller Einsatz für Web Server-Steuerelemente sein. Denken Sie aber an eine gute Benutzerführung, sollten Webseiten auf Aktionen jederzeit reagieren. Es kann also notwendig sein, Textelemente auszutauschen, gestalterisch zu verändern oder überhaupt erst später erscheinen zu lassen, um den Benutzer über bestimmte Vorgänge oder Ereignisse zu informieren.

Diese Möglichkeiten werden von folgenden Klassen geboten:

► Panel

Dies ist ein Container, der eine definierte Fläche belegt und mit statischem Inhalt oder weiteren Elementen gefüllt werden kann. Das korrespondierende HTML-Element ist `<div>`. Panel-Steuerelemente erzeugen immer Absätze.

► Label

Dies ist ein Container, der mehrere Zeichen umfasst und entweder allein im Fließtext oder als Kind eines `Panel`-Steuerelements auftritt.

► Placeholder

Diese Klasse wird direkt von `Control` abgeleitet, nicht von `WebControl`, wie die anderen. Die Objekte sind etwas primitiver und haben weniger Methoden und Eigenschaften. Das `Placeholder`-Steuerelement dient der Einbettung anderer `Controls` oder der direkten Ausgabe von Text. Es erzeugt kein eigenes HTML-Tag. Natürlich fehlen mangels umschließendem Tag sämtliche Formatierungsmöglichkeiten. Enthaltene Steuerelemente können wiederum alles enthalten, was mit Web Server-Steuerelementen möglich ist.

► Literal

Auch dieses Steuerelement erzeugt kein HTML selbst, sondern dient der Ausgabe von einfachem Text. Auch hier sind keine Formatierungsmöglichkeiten gegeben.

Das folgende Beispiel zeigt die Anwendung der `Panel`- und `Label`-Steuerelemente. Zu `Label` finden Sie außerdem unzählige Verwendungen in den Beispielen dieses Buches.

```
<html>
<head>
  <title>Panel und Label</title>
  <style>
    #PanelLabel {font-size:12pt; color:red; font-weight:bold}
  </style>
</head>
<body>
  <h3>Panel und Label</h3>
  <form runat=server>
    <asp:panel id="GroupPanel" runat="server"
      BackColor="#eeeeee"
      Height="50px" Width="250px">
      <b>Einige Label:</b><br/>
    </asp:panel>
    <p>
      <asp:checkbox id="CheckVisibility"
        Text="Panel verstecken"
        autopostback="true"
        onCheckedChanged="checkVisibility"
        runat="server"/>
    </p>
```

```

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

```

Listing 6.16: Erzeugen eines Panels und Einbau von Text (Panellabel.aspx)

```

Public Class Panellabel
    Inherits System.Web.UI.Page
    Protected GroupPanel As Panel
    Protected WithEvents CheckMyVisibility As CheckBox

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As EventArgs)
        Dim TextLabel As Label = New Label()
        Dim TextLit As Literal = New Literal()
        TextLit.Text = "<hr />"
        TextLabel.Text = "Dieser Text ist ein Label"
        TextLabel.ID = "Panellabel"
        GroupPanel.Controls.Add(TextLit)
        GroupPanel.Controls.Add(TextLabel)
    End Sub

    Public Sub CheckVisibility(ByVal sender As Object, _
        ByVal e As EventArgs) _
        Handles CheckMyVisibility.CheckedChanged
        If CheckMyVisibility.Checked Then
            GroupPanel.Visible = False
            CheckMyVisibility.Text = "Panel wieder anzeigen"
        Else
            GroupPanel.Visible = True
            CheckMyVisibility.Text = "Panel verstecken"
        End If
    End Sub
End Class

```

**Listing 6.17: Code-Datei zur Behandlung des Ereignisses
(Ausschnitt aus Panellabel.aspx.vb)**

Wie es funktioniert

Die Definition des Panel-Steuerelements erfolgt im HTML-Teil mit `<asp:panel>`. Enthalten ist bereits ein statischer Text. Der Zugriff auf den Inhalt erfolgt nicht mit der Eigenschaft `Text`, sondern – und dies ist der Unterschied zu `Label` – mit einer `Control`-Kollektion. Im Beispiel werden zwei neue Steuerelemente hinzugefügt, einmal ein `Label` mit dem Namen `TextLabel` und ein Stück HTML-Code, der als `Literal` (einfacher Text) ausgegeben wird. Literale verändern den empfangenen Text niemals, sie unterscheiden sich insofern von den Eigenschaften `InnerText` und `InnerHtml`, die bei

den HTML Server-Steuerelementen zur Ausgabe von Text zum Einsatz kamen. Das Hinzufügen zur `Control`-Kollektion erfolgt dagegen wieder ganz konservativ mit `Add`:

```
GroupPanel.Controls.Add(TextLit)
```

Was genau passiert, verrät ein Blick in den HTML-Code, den dieses Programm erzeugt:

```
<div id="GroupPanel" ␣
    style="background-color:#EEEEEE;␣
        height:50px;␣
        width:250px;">
<b>Einige Label:</b><br/>
<span id="PanelLabel">Dieser Text ist ein Label</span>
<hr/>
</div>
```

Erwartungsgemäß ist ein `<div>`-Tag zu finden, das optimal mit CSS gestaltet wurde. Es enthält neben dem statischen Text – der intern als `Literal` gespeichert wird – das Label in Form eines `<span>`-Tags und das zusätzliche HTML-Tag `<hr />`.



Abbildung 6.13: Textfelder als Panel mit gestalteten Elementen

Die Gestaltung des Steuerelements `Label` rührt von einem zusätzlichen Style her, dass als `<style>`-Anweisung direkt im HTML-Teil eingebettet wurde. Alle Elemente, die ein `id`-Attribut haben, lassen sich auch über CSS ansprechen. Das hat zwar nichts mit ASP.NET zu tun, ist aber eine interessante Verquickung beider Technologien, die die Gestaltung von Seiten vereinfachen kann.

Ohne das dies nochmals erläutert werden soll, enthält auch dieses Beispiel eine Anwendung der `AutoPostBack`-Funktion, die in diesem Fall das `Panel`-Objekt ein- und ausschaltet. Dies erfolgt mit der Eigenschaft `Visible`.



Wenn ein Element unsichtbar gemacht wird, erledigt dies das Style-Attribut `display:none`. Nichtsdestotrotz bleibt der HTML-Code im Quelltext der Seite erhalten und ist für neugierige Benutzer sichtbar. Zum Verstecken von Informationen ist dieser Weg nicht geeignet.

6.4.4 Texteingabefelder erzeugen und auswerten

Formulare bestehen fast immer auch aus Texteingabefeldern. ASP.NET verfolgt mit den Web Server-Steuerelementen hier einen interessanten Ansatz, denn die logisch ähnlichen Texteingabefelder `<input type="text">` und `<textarea>` sind unter einer Klasse erreichbar. Dass ein einzeliges Feld so völlig anders benannt und behandelt wird als ein mehrzeiliges, bereitet HTML-Anfängern oft Schwierigkeiten.

Das TextBox-Steuerelement

Texteingabefelder

Damit ist nun Schluss, denn das Web Server-Steuerelement `TextBox` definiert alle Arten von Eingabefeldern. Dank CSS erfolgt dies auch mit einheitlichem Aussehen bei vergleichbaren Attributen. Bei HTML allein ist das nicht der Fall, denn die Angabe `size="10"` im Tag `<input type="text">` führt nicht zu derselben Breite des Feldes wie die Angabe `columns="10"` im Tag `<textarea>`, was man eigentlich erwarten könnte.

Trotzdem wird ASP.NET in jedem Fall das passende HTML-Tag erzeugen und senden und damit die Kompatibilität mit allen Browsern sicherstellen. Ein Beispiel zeigt, wie dieses Steuerelement eingesetzt werden kann:

```
<html>
<head>
  <title>Textbox</title>
</head>
<body>
  <h3>Textbox</h3>
  <form runat=server>
    <asp:radiobutton id="Zeile1"
      groupname="Zeilen" runat="server"
      autopostback="true" onCheckedChanged="setBox"
      text="1 Zeile" value="1" /><br/>
    <asp:radiobutton id="Zeile3"
      groupname="Zeilen" runat="server"
      autopostback="true" onCheckedChanged="setBox" />
```



```

        text="3 Zeilen" value="3"/><br/>
<asp:radiobutton id="Zeile7"
    groupname="Zeilen" runat="server"
    autopostback="true" onCheckedChanged="setBox"
    text="7 Zeilen" value="7" /><br/>
<asp:textbox id="TextEingabe" runat="server"/>
</form>
</body>
</html>

```

Listing 6.18: *Dynamisch verändertes Texteingabefeld (TextBox.aspx)*

```

Public Class Mytextbox
    Inherits System.Web.UI.Page
    Protected WithEvents Zeile1 As RadioButton
    Protected WithEvents Zeile3 As RadioButton
    Protected WithEvents Zeile7 As RadioButton
    Protected TextEingabe As TextBox

    Private Sub Page_Load(ByVal sender As Object,
        ByVal e As EventArgs)
        TextEingabe.Width = Unit.Pixel(300)
        TextEingabe.BorderWidth = Unit.Pixel(1)
        TextEingabe.Text = "Ihre Eingabe"
    End Sub

    Public Sub setBox(ByVal sender As Object,
        ByVal e As EventArgs)
        Dim z As Integer
        z = Convert.ToInt16(Request.Form("Zeilen"))
        If z > 1 Then
            TextEingabe.TextMode = TextBoxMode.MultiLine
        Else
            TextEingabe.TextMode = TextBoxMode.SingleLine
        End If
        TextEingabe.Rows = z
    End Sub
End Class

```

Listing 6.19: *Code zu TextBox.aspx (Ausschnitt aus TextBox.aspx.vb)*

Das Formular enthält drei Optionsfelder, welche die Auswahl zwischen einer, drei oder sieben Zeilen für das Textfeld erlauben. Auch hier wird wieder `AutoPostBack` verwendet. Die Größe wird in der Methode `Page_Load` voreingestellt. Außerdem wird hier ein Text vorbelegt. Wird eine der Optionen angeklickt, sendet der Browser das Formular an den Server. Die Methode `setBox` wird zur Bearbeitung des Ereignisses ausgelöst.

Wie es funktioniert

**Probleme mit
RadioButton**

Dann wird – ganz konventionell – mit der `Form`-Kollektion gearbeitet. Die Technik der Web Server-Steuerelemente beruht jedoch eigentlich darauf, mit den vorhandenen Eigenschaften zu arbeiten. Optionsfelder sollen danach über ihre ID verarbeitet werden. Wenn Sie aber direkt Werte verarbeiten möchten, sollte eigentlich das Attribut `value` benutzt werden. Erstaunlicherweise gibt es in der Klasse `RadioButton` dazu keine passende Eigenschaft. Hier wird zur Abwechslung und zur Lösung des »value«-Problems mal direkt auf die Form-Daten zugegriffen, mit dem »alten« `Request.Form`:

```
Dim z As Integer = Convert.ToInt16(Request.Form("Zeilen"))
```

Optionsfelder bilden Gruppen über gleiche Namen, das Attribut `groupname` taucht in HTML als `name` auf. Das ist der Name des Feldes, das gesendet wird.

**TextBox-
Eigenschaften**

Jetzt wird die `TextBox` entsprechend der Angabe eingerichtet. Dies erfolgt über die Eigenschaft `TextMode`. Zulässig sind drei Eigenschaften, die aus einer Aufzählung vom Typ `TextBoxMode` entnommen werden können:

- ▶ `TextBoxMode.MultiLine`
Mehrzeiliges Feld, erzeugt `<textarea>`.
- ▶ `TextBoxMode.SingleLine`
Einzeiliges Feld, erzeugt immer `<input type="text">`.
- ▶ `TextBoxMode.Password`
Einzeiliges Feld, erzeugt immer `<input type="password">`.

Allein mit der Angabe der Zeilenanzahl ändert sich die Darstellung nicht. Wenn die Zeilenzahl größer als 1 ist, wird der Modus geändert:

```
TextEingabe.TextMode = TextBoxMode.MultiLine
```

Dann wird die Anzahl der Zeilen zugewiesen:

```
TextEingabe.Rows = z
```

Das Ergebnis ist, gemessen am Aufwand, durchaus überzeugend und vor allem sehr einfach erweiterbar und mit vielfältigen Formatierungsoptionen zu ergänzen.

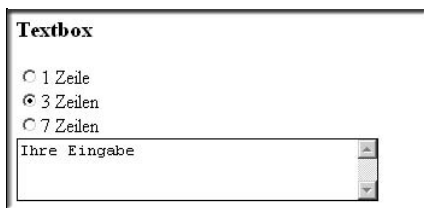


Abbildung 6.14: Steuerung der Zeilenzahl einer TextBox (textbox.aspx)

Was jetzt noch fehlt, ist eine Möglichkeit, das Formular zu senden. Dazu werden Schaltflächen benötigt.

Diskussion der anderen Zugriffsformen auf Werte von Optionsfeldern

Wie bereits angedeutet, gibt es ein paar Probleme beim Zugriff auf die Klasse `RadioButton`, da die Eigenschaft `Value` fehlt. Das Attribut kann aber dennoch verwendet werden, indem es direkt dem Tag `<asp:radiobutton>` hinzugefügt wird. Alle Attribute, die nicht direkt als Eigenschaft zur Verfügung stehen, lassen sich über die Kollektion `Attributes` erreichen. Der entsprechende Code sieht dann folgendermaßen aus:

```
Dim rb As RadioButton = CType(sender, RadioButton)
Dim z As Integer = CInt(rb.Attributes("value"))
```

Einen anderen Weg bieten die HTML Server-Steuerelemente. In diesem Fall wäre die Klasse `HtmlInputRadioButton` einsetzbar, denn dieses Element kennt als stringentere Umsetzung der Optionsfelder die Eigenschaft `Value`. Damit gelingt folgende Konvertierung mit anschließendem Zugriff:

```
Dim rbtn As HtmlInputRadioButton = CType(sender,
HtmlInputRadioButton)
Dim z As Integer = CInt(rbtn.Value)
```

Was letztlich verwendet wird, ist Geschmackssache. Falls mehrere Formulare diese Technik verwenden, sollten Sie aber einheitlich arbeiten, um die Pflege des Codes zu erleichtern.

6.4.5 Schaltflächen erzeugen und verwenden

Der Umgang mit Schaltflächen beschränkt sich in HTML auf Sendeschaltflächen (Submit) und einige Tricks mit JavaScript. Letzte-

res ist unumgänglich, verliert aber Dank ASP.NET seine Tücken und wird zu einem gemeinsamen Steuerelement: `Button`. Eine typische Aufgabe war bislang immer wieder die Reaktion auf mehr als eine Schaltfläche. Dies ist mit so genannten Kommandoschaltflächen möglich. Einheitlich ist auch die Darstellung des Steuerelements:

```
<asp:button id="name" runat="server"/>
```

Auf Ereignisse reagieren

Klickereignisse auswerten

Um gezielt auf das Anklicken einer Schaltfläche reagieren zu können, wird mit dem Ereignis `onClick` eine Ereignisbehandlungsmethode verbunden. Wenn Sie mit einer Ereignisbehandlungsmethode mehrere Schaltflächen bedienen möchten, müssen Sie eine Unterscheidung treffen können. Das ist sicher möglich, indem aus dem Parameter *sender* das `Button`-Objekt entnommen und dessen ID ausgewertet wird. Eleganter ist die Nutzung von Kommandos. Dazu wird das Ereignis `onCommand` definiert, das auch auf einen Mausklick reagiert. Zusätzlich stehen aber zwei weitere Attribute zur Verfügung, die dem Ereignis Parameter mitgeben:

► `CommandName`

Hiermit teilen Sie dem Kommando einen willkürlichen Namen zu. Innerhalb der Ereignisbehandlungsmethode steht dieser Name als Eigenschaft `CommandName` des Ereignisarguments vom Typ `CommandEventArgs` zur Verfügung.

► `CommandArgument`

Hiermit teilen Sie dem Kommando ein zusätzliches Argument zu. Innerhalb der Ereignisbehandlungsmethode steht dieser Name als Eigenschaft `CommandArgument` des Ereignisarguments vom Typ `CommandEventArgs` zur Verfügung. Es steht Ihnen beim Entwurf der Ereignisstruktur frei, dies zu verwenden und damit die Kommandos weiter zu strukturieren.

Das folgende Beispiel zeigt die Nutzung derartiger Kommandos:

```
<html lang="de">
  <head>
    <title>Buttons</title>
  </head>
  <body>
    <h3>Ihre Bewertung bitte:</h3>
    <form runat="server">
```

```

        <asp:button id="b1" runat="server"
            commandname="vote"
            commandargument="1"
            oncommand="checkVote" />
        <asp:button id="b2" runat="server"
            commandname="vote"
            commandargument="2"
            oncommand="checkVote" />
        <asp:button id="b3" runat="server"
            commandname="vote"
            commandargument="3"
            oncommand="checkVote" />
        <asp:button id="b4" runat="server"
            commandname="vote"
            commandargument="4"
            oncommand="checkVote" />
        <asp:button id="b5" runat="server"
            commandname="vote"
            commandargument="5"
            oncommand="checkVote" />
        <asp:button id="bCancel" runat="server"
            commandname="cancel"
            oncommand="checkVote" />

    </form>
    <asp:label id="Result" runat="server" />
</body>
</html>

```

Listing 6.20: Schaltflächen mit Kommandos identifizieren (ButtonCommand.aspx)

```

Public Class ButtonCommand
    Inherits System.Web.UI.Page
    Protected b1 As Button
    Protected b2 As Button
    Protected b3 As Button
    Protected b4 As Button
    Protected b5 As Button
    Protected bCancel As Button
    Protected Result As Label

    Private Sub Page_Load(ByVal sender As Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        b1.Text = "Wähle 1"
        b2.Text = "Wähle 2"
        b3.Text = "Wähle 3"
        b4.Text = "Wähle 4"
        b5.Text = "Wähle 5"
        bCancel.Text = "Abbrechen"
    End Sub

```

```

Public Sub checkVote(ByVal sender As Object,
                    ByVal e As CommandEventArgs)
    If e.CommandName = "vote" Then
        Result.Text = "Ihre Wahl war: "
        + CType(e.CommandArgument, String)
    Else
        Result.Text = "Schade..."
    End If
End Sub
End Class

```

Listing 6.21: Code zu ButtonCommand.aspx (Ausschnitt aus ButtonCommand.aspx.vb)

Wie es funktioniert

Die Definition der Steuerelemente unterscheidet sich nur wenig von allen anderen. Neu sind lediglich die Attribute `commandname` und `commandargument`. Zur Definition der Ereignisbehandlung wird diesmal `oncommand` eingesetzt. Interessant ist hier, wie die Kommandos im Programm verarbeitet werden. Die Ereignisbehandlungsmethode `checkVote` übernimmt im Beispiel diese Aufgabe. Dabei werden die Argumente leicht modifiziert. Bislang wurde das zweite Argument, Typ `EventArgs`, kaum verwendet. Diesmal kommt es zum Einsatz, und zwar in einer überladenen Form, `CommandEventArgs`. Nur so haben Sie Zugriff auf die Ereignisse der Kommandoschaltflächen:

```

Public Sub checkVote(ByVal sender As Object,
                    ByVal e As CommandEventArgs)

```

Die Variable `e` enthält nun die Kommandos, die mit den Eigenschaften `CommandName` und `CommandArgument` abgefragt werden können:

```

If e.CommandName = "vote" Then

```

Das ist eigentlich alles, was es speziell zu dieser Art Schaltfläche zu sagen gibt. Die übrigen Eigenschaften betreffen vor allem die Gestaltung.

Ihre Bewertung bitte:

Wähle 1 Wähle 2 Wähle 3 Wähle 4 Wähle 5 Abbrechen

Ihre Wahl war: 4

Abbildung 6.15: Mehrere Schaltflächen in einem Formular

Beschriftung der Schaltfläche

Die Kommandos müssen natürlich nicht verwendet werden, wenn es nur eine einfache Sendeschaltfläche gibt. Das Formular gelangt dann auf dem normalen Weg zum Server, wo mit `IsPostBack` festgestellt werden kann, ob die Seite das erste Mal oder nachfolgend aufgerufen wurde. Wollen Sie die Sendeschaltfläche im Programm gestalten, ist es dennoch notwendig, sie als Steuerelement zu deklarieren. Das folgende Beispiel zeigt, wie die Beschriftung in Abhängigkeit von der Situation geändert werden kann.

```
<html lang="de">
  <head>
    <title>Sendeschaltfl&auml;che</title>
  </head>
  <body>
    <h1>Sendeschaltfl&auml;che</h1>
    <form runat="server">
      <asp:button id="senden" runat="server"/>
    </form>
  </body>
</html>
```

Listing 6.22: Beschriftung der Sendeschaltfläche dynamisch ändern (*ButtonSubmit.aspx*)

```
Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles MyBase.Load
    If Not IsPostBack Then
        senden.Text = "Bitte senden Sie das Formular"
    Else
        senden.Text = "Formular erneut senden"
    End If
End Sub
```

Listing 6.23: Code zu *ButtonSubmit.aspx* (Ausschnitt aus *ButtonSubmit.aspx.vb*)

Als einzige Besonderheit ist das Fehlen besonderer Techniken anzumerken. Die Eigenschaft `Text` beeinflusst die Beschriftung der Schaltfläche.

Nach dem das Senden der Formulare mit den gezeigten Methoden möglich ist, sollte auch die Qualität des Inhalts geprüft werden. Eine spezielle Gruppe von Web Server-Steuerelementen ist dafür geeignet – die Kontroll-Steuerelemente (Validation Controls). Beschrieben werden sie in Abschnitt 6.7, »Kontroll-Steuerelemente (Validation Controls)« ab Seite 616.



Abbildung 6.16: Zustand der Schaltfläche nach dem ersten Senden

6.5 Listen-Steuerelemente

Zur Anzeige von Auswahlmöglichkeiten sind Listen besonders geeignet. In ASP.NET ist es möglich, nicht nur einfache Drop-Down-Listen zu erzeugen, sondern auch die Standardfelder Optionsfeld und Kontrollkästchen zu Listen zusammenzufassen. Dabei wird auch hier lediglich HTML erzeugt, während aus Sicht des Programmierers nur ein Objekt existiert.

6.5.1 Listen erzeugen und verwalten

Besondere Schwierigkeiten bereiten immer wieder Listen, deren Umfang von dynamischen Daten abhängig ist. ASP.NET kennt dafür mehrere Elemente. Ähnlich wie bei Texteingabefeldern ist die Darstellung logischer und der Windows-Bedienung angepasster. Bei den Textfeldern existierten in HTML zwei Tags, die logisch zu einem zusammengefasst wurden. Für die Listen, die in zwei verschiedenen Darstellungen auftreten, gibt es dagegen nur ein Tag.

Eine weitere Anwendung von Listen finden Sie im Abschnitt 4.3.4, »Zugriff auf Aufzählungen« ab Seite 231. Dort wird die Liste mit den Werten eines Arrays gefüllt.

Einfache Listboxen

Listen Bei den Listen gibt es in HTML nur ein Tag, das von zwei Web Server-Steuerelementen erzeugt wird:

► DropDownList

Dies sind Listen, die immer einzellig erscheinen und beim Anklicken herunter klappen (engl. *drop down*, daher der Name). Sie werden in HTML mit `<select size="1">` erzeugt.

► ListBox

Hier werden alle Elemente der Liste gleichzeitig angezeigt. Es können ein oder auch mehrere Elemente ausgewählt werden. In HTML wird dazu ebenso `<select>` verwendet, mit dem Attribut `size` und einem Parameter größer als 1. Die Mehrfachauswahl kann mit dem Attribut `multiple` gestattet werden.

Die Elemente der Listen lassen sich über eigene Steuerelemente erzeugen und ändern, `ListItems` genannt. Die Darstellung erfolgt als `<option>`-Tag, das in HTML nur innerhalb einer Liste vorkommen darf. ASP.NET bringt bei Listboxen deutliche Fortschritte gegenüber der herkömmlichen Programmierung. So war es nicht einfach, den Zustand der Liste, also den Status der gewählten Elemente, zu erhalten. Mit Web Server-Steuerelementen ist dies kein Problem mehr. Das folgende Beispiel zeigt, wie die Elemente einer Listbox aus einem Array erzeugt werden können:

Listenelemente

```
<html lang="de">
  <head>
    <title>Select</title>
  </head>
  <body>
    <h1>Wählen Sie die Versandart</h1>
    <form runat="server">
      <asp:button id="auswahl" runat="server"
        text="Bitte wählen"/>
      <asp:dropdownlist id="versand" runat="server"/>
    </form>
  </body>
</html>
```

Listing 6.24: Listbox dynamisch füllen (*SelectDropDownList.aspx*)

```
Public Class SelectDropDownList
  Inherits System.Web.UI.Page
  Protected versand As DropDownList
  Protected auswahl As Button

  Private Sub Page_Load(ByVal sender As Object,
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    If Page.IsPostBack Then
      auswahl.Text = "Bestätigen Sie "
      + versand.SelectedItem.Text.ToString() + "
```

```

Else
    Dim versandarten() As String = {"Express-Paket", "↵",
                                     "Over-Night", "↵",
                                     "Standard 48 Stunden"}

    Dim i As Integer = 0
    Dim versandart As String
    For Each versandart In versandarten
        Dim versandoption As ListItem = New ListItem()
        versandoption.Text = versandart
        i = i + 1
        versandoption.Value = i.ToString()
        versand.Items.Add(versandoption)
    Next
End If
End Sub
End Class

```

*Listing 6.25: Code zu SelectDropDownList.aspx
(Ausschnitt aus SelectDropDownList.aspx.vb)*

Wie es funktioniert

Im Formular selbst wird nur das Listen-Steuerelement selbst definiert:

```
<asp:dropdownlist id="versand" runat="server"/>
```

Beim Aufruf der Seite wird ein Array erzeugt, dass die Einträge enthält. Dieses Array wird in einer Schleife durchlaufen:

```
For Each versandart In versandarten
```

Für jedes Element wird nun ein Objekt vom Typ `ListItem` erzeugt:

```
Dim versandoption As ListItem = New ListItem()
```

Dann werden diesem Element zwei Attribute zugewiesen, `Text` für den Teil, der angezeigt wird und `Value` für den zu übertragenden Wert. Der zu übertragende Wert ist eine Zahl, die meist einfacher weiterverarbeitet werden kann. Diese Zahl wird mit jedem Durchlauf um eins erhöht ($i = i + 1$) und muss für die Zuweisung noch in eine Zeichenkette konvertiert werden:

```
versandoption.Value = i.ToString()
```

Zum Schluss wird das `ListItem`-Objekt der Liste hinzugefügt:

```
versand.Items.Add(versandoption)
```

Wählen Sie die Versandart



Abbildung 6.17: Darstellung bei der ersten Auswahl

Wählen Sie die Versandart



Abbildung 6.18: Nach der Wahl einer Option

Die Eigenschaft `Items` spielt beim Umgang mit Listen eine herausragende Rolle. Objekte, die hier enthalten sind, sind vom Typ `ListItemCollection`. Auch diese Klasse ist Bestandteil des Namensraumes `System.Web.UI.WebControls`. Da Elemente der Kollektion nicht allein existieren können, sondern nur als Bestandteil von `Items`, waren sie in der Übersicht am Anfang des Kapitels nicht enthalten. Wichtige Eigenschaften sind:

ListItemCollection

► `Capacity`

Gibt die maximale Anzahl Elemente an, die erlaubt sind.

► `Count`

Die aktuelle Anzahl der Elemente.

► `Item`

Zugriff auf ein Element. Außerdem ist dies die Standardmethode, sodass der Zugriff mit der Schreibweise `MeineListe("ListItem")` erfolgen kann.

Die Methoden entsprechen denen anderer Listen. `Add` fügt ein Element am Ende der Liste hinzu, `Insert` dagegen an einer bestimmten Position. `Clear` löscht alle Elemente, `Remove` dagegen nur ein bestimmtes. Interessant sind zwei spezielle Methoden dieser Listenart: `FindByText` sucht ein Element mit einem bestimmten Inhalt der Eigenschaft `Text`; `FindByValue` entsprechend für `Value`.

Umfangreichere Datenmengen stammen meist aus Datenbanken. Die Zuweisung über einzelne `ListItem`-Objekte wäre umständlich. Weitere Eigenschaften dienen deshalb dem direkten Zugriff auf Datenquellen.

Listen aus Optionsfeldern und Kontrollkästchen

Optionsfelder

Auswahlmöglichkeiten in Form von Listen sind nicht immer optimal für die Benutzerführung. Oft sind Listen von Kontrollkästchen oder Optionsfeldern besser geeignet. Die Erzeugung solcher Gruppen kann mit Objekten vom Typ `CheckBoxList` oder `RadioButtonList` vereinfacht werden. Das Beispiel zeigt, wie dies aussehen kann:

```
<html lang="de">
  <head>
    <title>Select</title>
  </head>
  <body>
    <h1>Wählen Sie die Versandart</h1>
    <form runat="server">
      <asp:button id="auswahl" runat="server"
        text="Bitte wählen"/>
      <asp:radiobuttonlist id="versand" runat="server"/>
    </form>
  </body>
</html>
```

Listing 6.26: Dynamisch erzeugte Gruppe von Optionsfeldern (RadioButtonList.aspx)

Im folgenden Listing wird unter anderem geprüft, ob der zurückgegebene Index der Optionsfelder den Wert -1 enthält. Ist das der Fall, wurde keines der drei Felder selektiert und die Entnahme des Wertes würde misslingen. Solche Prüfungen sind in der Praxis enorm wichtig, um Benutzer nicht mit Laufzeitfehlern zu konfrontieren. Es ist im übrigen auch falsch, zu diesem Zweck *Try-Catch*-Kombinationen zu benutzen, denn dies sind keine Konstruktionen, um im regulären Betrieb typischerweise auftretende Fehler abzufangen, sondern sie sollten tatsächlich »Ausnahmen« vorbehalten bleiben.

```
Public Class RadioButtonListControl
  Inherits System.Web.UI.Page
  Protected versand As RadioButtonList
  Protected auswahl As Button

  Private Sub Page_Load(ByVal sender As Object, ↵
```

```

ByVal e As System.EventArgs) _
Handles MyBase.Load
If IsPostBack Then
    If versand.SelectedIndex = -1 Then
        auswahl.Text = "Keine Auswahl"
    Else
        auswahl.Text = versand.SelectedItem.Text.ToString() &
            + " OK?"
    End If
Else
    Dim versandarten() As String = {"Express-Paket", "Over-Night",
        "Standard 48 Stunden"}

    Dim i As Integer = 0
    Dim versandart As String
    For Each versandart In versandarten
        Dim versandoption As ListItem = New ListItem()
        versandoption.Text = versandart
        i = i + 1
        versandoption.Value = i.ToString()
        versand.Items.Add(versandoption)
    Next
End If
End Sub
End Class

```

Listing 6.27: Code zu RadioButtonList.aspx (Ausschnitt aus RadioButtonList.aspx.vb)

Tatsächlich besteht der einzige Unterschied gegenüber dem letzten Beispiel im Austausch des Tags im HTML-Teil; hier erscheint nun `<asp:radiobuttonlist>`. Interessanter als eine erneute Analyse ist ein Blick in den erzeugten HTML-Code:

HTML

```
<form name="_ctl0" method="post"
    action="radiobutton_ddlist.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTcwNzM1MzA7dDw7bDxpPDI+Oz47bDx0PDtsPGk8MT47a
TwzPjs+02w8dDxwPHA8bDxUZxh00z47bDxJaHJlIFdhaGwg2Fy0js+
Pjs+0zs+03Q8dDw7cDxsPGk8MD47aTwxPjtpPDI+Oz47bDxwPEV4cHJ
lc3MtUGFrZXQ7MD47cDxpDmVyLU5pZ2h00zE+O3A8U3RbhmRhcMqg
NDggU3R1bmRlbjJsyPjs+Pjs+0zs+0z4+0z5UqZgPBlvAhn2FSJdtw1la
q0uSw==" />
<input type="submit" name="auswahl"
    value="Ihre Wahl war:" id="auswahl" />
<table id="versand" border="0">
    <tr>
        <td><input id="versand_0" type="radio" name="versand"
            value="0" checked="checked" />
            <label for="versand_0">Express-Paket</label></td>
```

```
</tr><tr>
  <td><input id="versand_1" type="radio" name="versand"
    value="1" />
    <label for="versand_1">Over-Night</label></td>
</tr><tr>
  <td><input id="versand_2" type="radio" name="versand"
    value="2" />
    <label for="versand_2">Standard 48 Stunden</label></td>
</tr>
</table>
</form>
```

**Formatierung von
Listen**

Interessant ist, dass eine Tabelle zur Formatierung eingesetzt wird. Tatsächlich sind die Formatieroptionen, die sich daraus ableiten lassen, sehr umfangreich. Wenn Sie sehr viele Elemente haben, wie es beispielsweise bei Umfrageseiten der Fall ist, helfen diese Optionen enorm. Hervorzuheben sind folgende Eigenschaften:

- ▶ RepeatColumns
Anzahl der Spalten, in denen die Anzeige gruppiert wird.
- ▶ RepeatDirection
Richtung, in der gruppiert wird. Die beiden Tabellen zeigen die Folge der Zellen.

1	3	5
2	4	6

Abbildung 6.19: RepeatDirection.Vertical

1	2	3
4	5	6

Abbildung 6.20: RepeatDirection.Horizontal

In beiden Fällen ist die Anzahl der Spalten drei.

► RepeatLayout

Diese Eigenschaft bestimmt die Art der Darstellung. Zulässig sind folgende Werte der Aufzählung RepeatLayout:

► RepeatLayout.Table

Die Darstellung erfolgt als Tabelle. Die ist der Standardwert.

► RepeatLayout.Flow

Die Darstellung erfolgt ohne Tabellenstruktur, also fließend.

Wenn Sie das Layout *RepeatLayout.Table* verwenden, wird HTML-Code kompatibel zu HTML 3.2 verwendet. Damit erfolgt die Anzeige in allen Browsern weitgehend ähnlich. Wird dagegen *RepeatLayout.Flow* eingesetzt, nutzt der HTML-Generator die Möglichkeiten von DHTML und HTML 4.0 einschließlich Cascading Style Sheets. Dies funktioniert zuverlässig nur im Internet Explorer.



Bei Kontrollkästchen gilt das bereits Gezeigte fast unverändert. Das folgende Beispiel zeigt eine andere Art der Produktpräsentation für einen kleinen Shop:

Kontrollkästchen

```
<html lang="de">
  <head>
    <title>Shop</title>
  </head>
  <body>
    <h1> Shop </h1>
    Unsere Produkte:
    <form runat="server">
      <asp:checkboxlist id="produkte" runat="server"/>
      <asp:button id="bestellen" runat="server"
        text="Jetzt Bestellen"/>
    </form>
  </body>
</html>
```

Listing 6.28: Mehrfachauswahl mit Kontrollkästchen (*CheckBoxesMultiple.aspx*)

```
Public Class CheckBoxMultiple
  Inherits System.Web.UI.Page
  Protected produkte As CheckBoxList
  Protected bestellen As Button
  Protected ausgabe As Label
```

```

Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles MyBase.Load
    If Not IsPostBack Then
        Dim ProduktArr(,) As String = New String(3, 3) {}
        Const ProdName As Integer = 0
        Const ProdPreis As Integer = 1
        Const ProdID As Integer = 2
        ProduktArr(0, ProdName) = "Codeschreiber 1.0"
        ProduktArr(0, ProdPreis) = "6299,9"
        ProduktArr(0, ProdID) = "A0001"
        ProduktArr(1, ProdName) = "Datenbanker 1.1"
        ProduktArr(1, ProdPreis) = "99,9"
        ProduktArr(1, ProdID) = "A0002"
        ProduktArr(2, ProdName) = "Projektor 2.0"
        ProduktArr(2, ProdPreis) = "39"
        ProduktArr(2, ProdID) = "A0003"
        Dim produkttext As String
        Dim i As Integer
        For i = 0 To ProduktArr.GetLength(0) - 2
            produkttext = ProduktArr(i, ProdName) 
                + " (" + String.Format("{0:N2}", 
                    Convert.ToDouble(ProduktArr(i, ProdPreis))) +
                ")"

            Dim produktoption As ListItem = New ListItem()
            produktoption.Text = produkttext
            produktoption.Value = ProduktArr(i, ProdID)
            produkte.Items.Add(produktoption)
        Next
    Else
        Dim li As ListItem
        ausgabe.Text = "Ihre Auswahl:<p />"
        For Each li In produkte.Items
            If li.Selected = True Then
                ausgabe.Text += li.Text + "<br />"
            End If
        Next
    End If
End Sub
End Class

```

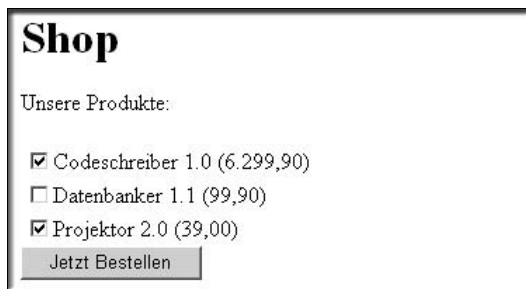
*Listing 6.29: Code zu CheckBoxMultiple.aspx
(Ausschnitt aus CheckBoxesMultiple.aspx.vb)*

Wie es funktioniert

Im Programm hat sich gegenüber den letzten Beispielen nur wenig geändert. Als Tag kommt nun `CheckBoxList` zum Einsatz:

```
<asp:checkboxlist id="produkte" runat="server"/>
```


Unverändert ist das Befüllen der Werte, wo erneut die Eigenschaften `Text` und `Value` der `ListItem`-Objekte zum Einsatz kommen. Das Ergebnis ist durchaus überzeugend:



The screenshot shows a web form titled "Shop". Below the title, it says "Unsere Produkte:". There is a list of three items, each with a checkbox and text: "Codeschreiber 1.0 (6.299,90)", "Datenbanker 1.1 (99,90)", and "Projektor 2.0 (39,00)". The checkboxes for "Codeschreiber 1.0" and "Projektor 2.0" are checked, while the one for "Datenbanker 1.1" is unchecked. At the bottom of the list is a button labeled "Jetzt Bestellen".

Abbildung 6.21: Produktauswahl mit einem einzigen Tag

Auch hier zeigt ein Blick in den HTML-Code, dass ASP.NET sehr konventionell vorgeht und `<input type="checkbox">`-Tags in einer Tabelle platziert. Die `Repeater`-Eigenschaften stehen auch hier zur Verfügung, falls größere Datenmengen anfallen.

Dank `Anzeigestatus` (*ViewState*) bleiben in allen Fällen die Werte erhalten, wenn die Formularseite an sich selbst gesendet wird. Auch die Verwendung von *AutoPostBack* ist möglich, damit allein die Aktivierung oder Deaktivierung einer Option das Formular sendet.

ViewState
AutoPostBack

Verwendung der Datenbindung

In den letzten Abschnitten wurden Listen-Steuerelemente vorgestellt. In allen Fällen wurden dabei die enthaltenen Elemente direkt erzeugt und angefügt. Wenn Sie eine Datenquelle haben, die `IEnumerator` implementiert, können Sie auch eine direkte Bindung vornehmen. Mehr Informationen dazu finden Sie in Abschnitt 9.5, »Die Datenbindung an Steuerelemente« ab Seite 926. Dabei geht es um eine deutlich einfachere Erzeugung von Listenelementen, als dies mit dem hier gezeigten Weg möglich ist. In der Praxis müssen Sie indes beide Wege beherrschen, denn nicht immer kann eines der vorgefertigten Elemente problemlos eingesetzt werden.

6.6 Vorlagengebundene Daten-Steuerelemente

In der Gruppe der Web Server-Steuerelemente gibt es drei Klassen, die komplexere Elemente realisieren. Sie können damit größere Datenmengen komfortabel ausgeben. Ebenso wie bei den bereits vorgestellten Listen-Steuerelementen ist eine Bindung an eine Datenquelle möglich. Zusätzlich kann jedoch die Steuerung in einer für Listen typischen Weise erfolgen. Die verfügbaren Steuerelemente sind:

- ▶ `<asp:repeater>`
- ▶ `<asp:datalist>`
- ▶ `<asp:datagrid>`

Das Repeater-Element ist am einfachsten und erlaubt eine weitgehend freie Gestaltung der Datenliste. Am komplexesten ist DataGrid, wo sogar die Bearbeitung von Datensätzen möglich ist.



In diesem Abschnitt werden die grundlegenden Eigenschaften der Elemente vorgestellt. Die Bindung von Daten des SQL Servers wird vertiefend in Abschnitt 9.5, »Die Datenbindung an Steuerelemente« ab Seite 926 behandelt. Dort finden Sie auch detailliertere Informationen über die Nutzung in verschiedenen Szenarien.

6.6.1 Einführung in die Daten-Steuerelemente

Die folgende Einführung gibt Tipps für die Auswahl der Elemente.

Repeater

Einsatz Das Steuerelement `Repeater` ist das einfachste und universellste Element. Es basiert wie alle Daten-Steuerelemente auf Vorlagen. Innerhalb des Elementes können mehrere Bereiche definiert werden, die sowohl die Gestaltung als auch den Inhalt beeinflussen. Dies hat einen signifikanten Vorteil gegenüber der reinen programmtechnischen Steuerung der Ausgabe. Denn wenn Design und Code parallel entwickelt werden, ist eine solche Trennung optimal.

Eigenschaften Das `Repeater`-Steuerelement verfügt über einige grundlegende Eigenschaften, anhand derer Sie den sinnvollen Einsatz abwägen können:

- ▶ Die Verwendung von Vorlagen ist zwingend; es gibt kein Standard-Layout.
- ▶ Elemente können nicht mit integrierten Methoden bearbeitet werden, es ist eine reine Ausgabeliste
- ▶ Die seitenweise Ausgabe wird nicht unterstützt. Normalerweise erscheinen alle Datensätze in einer einzigen Liste.
- ▶ Es ist möglich, Separatoren zwischen den Elementen zu zeichnen, indem eine Separator-Vorlage definiert wird.
- ▶ Benutzerdefinierte Ereignisse sind möglich, um bestimmte datensatzgebundene Ereignisbehandlungsmethoden aufzurufen.
- ▶ Es ist das einzige Steuerelement, mit dem HTML-Tags über mehrere Vorlagen hinweg aufgeteilt werden können.

Um das Prinzip der Vorlagen besser verstehen zu können, zeigt die folgende Abbildung die Struktur in drei Varianten:

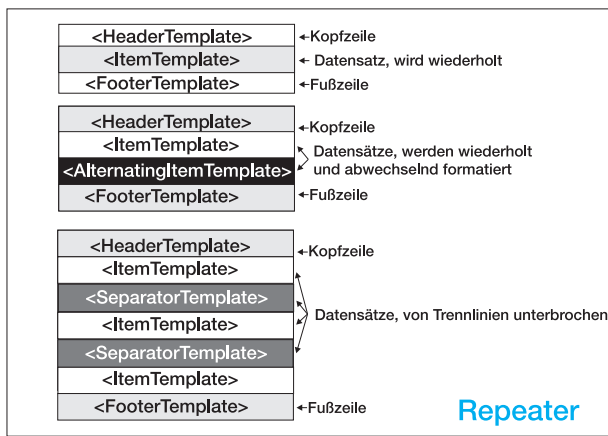


Abbildung 6.22: Drei mögliche Formen des Vorlageneinsatzes beim Steuerelement *Repeater*

Die Vorlage `<ItemTemplate>` ist erforderlich, die anderen sind optional.

DataList

Das `DataList`-Steuerelement basiert wie alle Daten-Steuerelemente auf Vorlagen. Innerhalb des Elementes können mehrere Bereiche definiert werden, die sowohl die Gestaltung als auch den Inhalt beeinflussen. Dies hat einen signifikanten Vorteil gegenüber der

Einsatz

reinen programmtechnischen Steuerung der Ausgabe. Denn wenn Design und Code parallel entwickelt werden, ist eine solche Trennung optimal.

Eigenschaften Das `DataList`-Steuerelement verfügt über einige grundlegende Eigenschaften, anhand derer Sie den sinnvollen Einsatz abwägen können:

- Die Ausgabe erfolgt standardmäßig als Tabelle. Das Layout kann Daten horizontal oder vertikal anordnen. Es ist jedoch auch ein freies Layout ohne Tabellen möglich.
- Es gibt integrierte Bearbeitungsfunktionen.
- Die Listenansicht kann umfassend angepasst werden.
- Benutzerdefinierte Ereignisse können mit Ereignisbehandlungsmethoden gekoppelt werden.

Um das Prinzip der Vorlagen besser verstehen zu können, zeigt die folgende Abbildung die Struktur in drei Varianten:

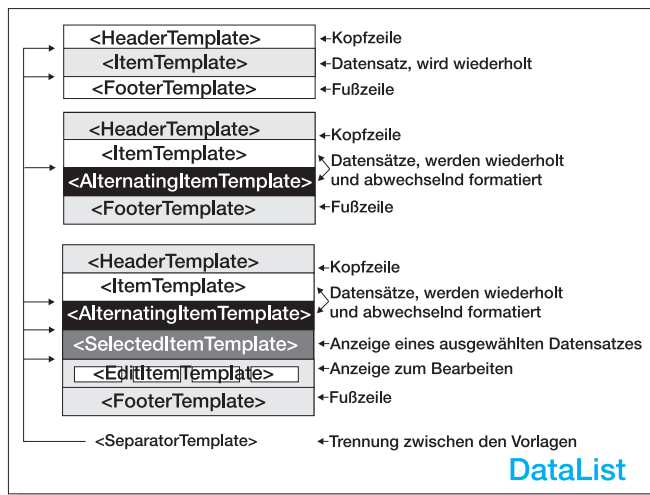


Abbildung 6.23: Drei mögliche Formen des Vorlageneinsatzes beim Steuerelement `DataList`

DataGrid

Einsatz Das `DataGrid`-Steuerelement basiert wie alle Daten-Steuerelement auf Vorlagen. Innerhalb des Elementes können mehrere Bereiche definiert werden, die sowohl die Gestaltung als auch den Inhalt

beeinflussen. Dies hat einen signifikanten Vorteil gegenüber der reinen programmtechnischen Steuerung der Ausgabe. Denn wenn Design und Code parallel entwickelt werden, ist eine solche Trennung optimal.

Das `DataGrid`-Steuerelement verfügt über einige grundlegende Eigenschaften, anhand derer Sie den sinnvollen Einsatz abwägen können:

Eigenschaften

- ▶ Bietet standardmäßig ein Datengitter (Tabellenform) als Formatierung an, ohne das Vorlagen oder Stile notwendig wären. Die Tabelle kann weit reichend formatiert werden.
- ▶ Die Darstellung kann umfassend mit Vorlagen angepasst werden.
- ▶ Es gibt keine gesonderte Trennzeichenvorlage.
- ▶ Es wird die Sortierung von Daten sowie das Bearbeiten und Löschen unterstützt.
- ▶ Benutzerdefinierte Ereignisse können mit Ereignisbehandlungsmethoden gekoppelt werden.

Um das Prinzip der Vorlagen besser verstehen zu können, zeigt die folgende Abbildung die Struktur in drei Varianten:

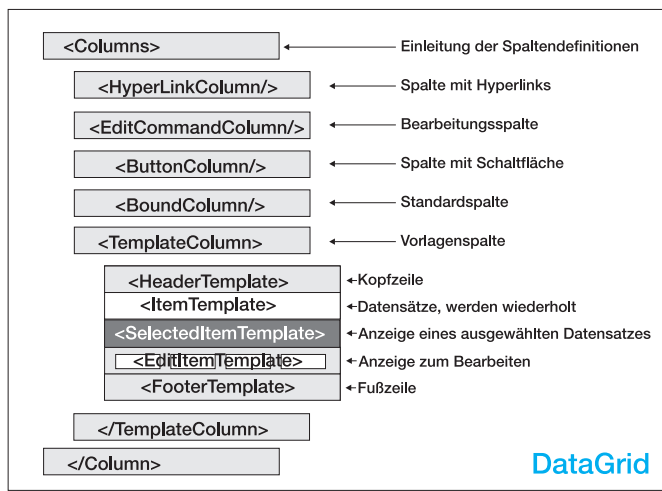


Abbildung 6.24: Teile des Vorlageneinsatzes beim Steuerelement `DataGrid`

6.6.2 Aufbau der Vorlagen in Daten-Steuerelementen

Der Aufbau von vorlagengebundenen Daten-Steuerelementen erfolgt nach folgendem Prinzip:

```
<asp:repeater id="repeater_name" runat="server">
  <HeaderTemplate>
    HTML-Daten für den Kopfbereich
  </HeaderTemplate>
  <ItemTemplate>
    HTML-Daten, die für jedes Element wiederholt werden
  </ItemTemplate>
  <FooterTemplate>
    HTML-Daten für den Fußbereich
  </FooterTemplate>
</asp:repeater>
```

Schreibweise Die Schreibweise der Vorlagen-Tags muss nicht zwingend in Großbuchstaben erfolgen. Ebenso ist keines der Elemente obligatorisch. Wenn die Vorlage `ItemTemplate` entfällt, ist der Sinn des Einsatzes natürlich fraglich. Grundsätzlich verfügen alle drei Daten-Steuerelemente über folgende Vorlagen:

► `<HeaderTemplate>`

Dieses Element wird nur einmal am Beginn der Liste ausgegeben. Hier kann die Gestaltung des Kopfbereiches erfolgen oder ein öffnendes HTML-Tag wie `<ul>`, oder `<table>` stehen. Die Angabe ist nicht zwingend. Eine Datenbindung an dieses Element ist nicht möglich, weil es nicht für die Ausgabe wiederholender Werte gedacht ist.

► `<FooterTemplate>`

Der Fußbereich wird mit dieser Vorlage erzeugt. Die Angabe ist freiwillig. Wenn Sie bereits den Kopfbereich definiert haben, sollten Sie den Fußbereich unbedingt ebenfalls verwenden, weil Sie so sicherstellen können, dass die schließenden HTML-Tags an der richtigen Stelle erscheinen (`</ul>` bzw. `</table>` usw.). Auch dieses Element kann keine Daten binden.

► `<ItemTemplate>`

In dieser Vorlage können Sie Datenbindungsausdrücke verwenden. Das Steuerelement sorgt dann dafür, dass diese wiederholt werden. Hier sollten Sie HTML so platzieren, dass es zwischen die Kopf- und Fußzeilen passt.

► `<AlternatingItemTemplate>`

Wenn dieses Element angegeben wird, erscheint es abwechselnd zu den zuvor definierten `ItemTemplate`-Vorlagen.

► `<SeparatorTemplate>`

Diese Element dient der Definition von Trennzeilen zwischen den Elementen einer Auflistung.

Eine Einführung in die Datenbindung in Vorlagen

Der Abruf der Daten innerhalb der Vorlagen erfolgt mit der Datenbindungssyntax. Dabei erfolgt der Zugriff in Form einer speziellen Syntax:

**Allgemeiner
Umgang mit der
Datenbindung**

```
<%# DatenzugriffsAusdruck %>
```

Für die hier behandelten Steuerelemente wird als Container der Daten immer der Ausdruck `Container.DataItem` verwendet. Zur Auswertung des aktuellen Inhalts des Containers kann zusätzlich die statische Methode `Eval` der Klasse `DataBind` verwendet werden. `Eval` ist dann angebracht, wenn Formatierungen oder Konvertierungen notwendig sind. VB.NET erledigt dies in den meisten Fällen implizit, sodass die relativ langsame Methode nur dann benutzt werden sollte, wenn die Darstellung auf anderem Wege nicht gelingt. Der Ausdruck würde dann folgendermaßen aussehen:

```
<%# DataBinder.Eval (Container.DataItem, "Datenquelle") %>
```

Was als Datenquelle angegeben wird, hängt davon ab, wie die Daten strukturiert sind. Handelt es sich um Daten aus einer SQL-Datenbank, werden meist Spaltennamen verwendet. Werden Objekte verwendet, sollten diese über eine lesbare Eigenschaft verfügen, deren Namen angegeben wird. Das folgende Beispiel demonstriert dies.

Wenn die Daten so gestaltet sind, dass die Auswahl einer Datenquelle nicht erforderlich bzw. nicht möglich ist, nutzen Sie folgende Syntax:

```
<%# Container.DataItem %>
```

Für die Formatierung wird dann die Standardform mit `String.Format` benutzt.

Abgesehen davon sind Sie in der Gestaltung des Ausdrucks frei. Sie können beliebigen Code verwenden, eigene oder eingebaute

**Formatierung der
Ausgabe**

Methoden aufrufen und Formatierungen vornehmen. Da Letzteres besonders häufig ist, kann `DataBinder.Eval` mit einem weiteren Parameter versehen werden, der Formatanweisungen enthält:

```
<%# DataBinder.Eval(Container.DataItem, "Datenquelle", "{0:c}") %>
```

Verkürzte Schreibweise

In VB.NET können Sie eine verkürzte Schreibweise verwenden, wenn keine Formatierung erforderlich ist. Diese entspricht der üblichen Objektschreibweise:

```
<%# Container.DataItem.Datenquelle %>
```

In welcher Sprache die Klasse selbst geschrieben wurde, spielt keine Rolle, entscheidend ist die Sprache der Seite, die in der Deklaration `@Page` eingestellt wird.

Zugriff auf eigene Aufzählungsklassen

Generell können sehr viele verschiedene Datenquellen verwendet werden. Letztlich ist allein das Vorhandensein einer Implementierung von `GetEnumerator` ausreichend. Wenn Sie nun selbst Datenquellen bereitstellen, die nicht den bereits gezeigten »Standardtypen« entsprechen, muss der Abruf in der Vorlage eine korrekte Typumwandlung vornehmen. Das folgende Beispiel zeigt, wie Eigenschaften einer eigenen Klasse verwendet werden können:

```
<%#
    CType(Container.DataItem.Eigenschaft, Namespace.MeineKlasse)
%>
```

Zuerst wird auf den `Container`, dann auf das aktuelle Element `DataItem` und – optional – eine seiner Eigenschaften zugegriffen. Dann wird das Element in den ursprünglichen Datentyp gewandelt. Sie müssen dazu den vollständigen Namensraumverweis anwenden, wenn nicht mit der Direktive `@Imports` gearbeitet wird. Danach stehen die Eigenschaften und Methoden des Objekts zur Verfügung.



Detailliertere Informationen über den Abruf von Datenquellen im Zusammenhang mit Datenbanken finden Sie im Abschnitt 9.5, »Die Datenbindung an Steuerelemente« ab Seite 926.

Die Datenbindung im Code

In der Klasse selbst, die das Steuerelement bedient, greifen Sie folgendermaßen darauf zu:

```
Public Repeater elementname
```


Die Datenquelle muss eine Auflistung sein. Dies erfordert die Ableitung von der Schnittstelle `IEnumerable`. Dazu gehören beispielsweise:

- ▶ `ArrayList`
- ▶ `DataGridView`
- ▶ `Hashtable`
- ▶ **Verschiedene spezialisierte Kollektionen, wie `CaptureCollection`, `MatchCollection`, `StringCollection` usw.**
- ▶ `Queue`
- ▶ `SortedList`

Als erstes Beispiel soll der Inhalt eines `ArrayList`-Objekts ausgegeben werden. Zuerst die Definition der Vorlage:

```
<asp:Repeater Runat="server" ID="SimpleList">
  <HeaderTemplate>
    <ul>
  </HeaderTemplate>
  <ItemTemplate>
    <li>
      <%# String.Format("{0:f}", Container.DataItem) %></li>
  </ItemTemplate>
  <FooterTemplate>
    </ul>
  </FooterTemplate>
</asp:Repeater>
```

Listing 6.30: Definition einer universellen Vorlage mit Repeater (Ausschnitt aus `SimpleRepeater.aspx`)

Offensichtlich soll die Ausgabe mit einer normalen Auflistung erfolgen. Der Zugriff auf den Container muss und kann nicht mit `Eval` ausgewertet werden, weil die Liste keine untergeordneten Daten enthalten kann, die angegeben werden können. In der Klasse, die die Daten erzeugt und die Liste füllt, wird nun folgendermaßen zugegriffen:

```
Public Class SimpleRepeater
  Inherits System.Web.UI.Page
  Public SimpleList As Repeater

  Private Sub Page_Load(ByVal sender As Object, ←
                        ByVal e As System.EventArgs) _
  Handles MyBase.Load
    Dim aList As ArrayList = New ArrayList()
    aList.Add(11.99)
```

```

        aList.Add(9.5)
        aList.Add(24.9)
        SimpleList.DataSource = aList
        SimpleList.DataBind()
    End Sub
End Class

```

Listing 6.31: Füllen der Liste mit Daten (Ausschnitt aus SimpleRepeater.aspx.vb)

Wie es funktioniert

Die Definition der `ArrayList` erfolgt wie üblich. Da `ArrayList` unter anderem die Schnittstelle `IEnumerable` implementiert, kann das daraus abgeleitete Objekt `aList` direkt als Datenquelle verwendet werden. Mit der Bindung durch die Methode `DataBind` erscheinen die Daten auf der Seite.

Daten-Steuerelemente

Zugriff auf ArrayList mit Repeater

- 11,99
- 9,50
- 24,90

Abbildung 6.25: Ausgabe einer Datenliste mit dem Repeater-Steuerelement

Nun können Sie derart einfache Listen sicher leichter erstellen. Denken Sie jedoch daran, dass die Gestaltung des HTML-Teils sehr viel komplexer ausfallen kann. Das ist dann notwendig, wenn nicht nur einfache Zahlen oder Zeichenketten ausgegeben werden müssen, sondern ganze Objekte.

Bindung komplexer Objekte

Angenommen, Sie speichern Daten in einer Struktur, die etwa folgendermaßen definiert ist:

```

Class SimpleRepeater2
    Inherits System.Web.UI.Page
    Private Structure Artikel
        Dim _name As String
        Dim _preis As Double
        Dim _menge As Integer
    Public Sub New(ByVal Name As String, ↵
        ByVal Preis As Double, ↵
        ByVal Menge As Integer)

```

```

        _name = Name
        _preis = Preis
        _menge = Menge
    End Sub
    ReadOnly Property Name() As String
        Get
            Return _name
        End Get
    End Property
    ReadOnly Property Preis() As Double
        Get
            Return _preis
        End Get
    End Property
    ReadOnly Property Menge() As Integer
        Get
            Return _menge
        End Get
    End Property
End Structure
End Class

```

Listing 6.32: Struktur zur Speicherung von Objekten
(Ausschnitt aus SimpleRepeater2.aspx.vb)

Sie können dann daraus abgeleitete Objekte in einer Liste speichern und die einzelnen Eigenschaften der Struktur gezielt ausgeben. Das Speichern in der Liste erfolgt wie bereits zuvor für skalare Werte gezeigt:

Datenbindung bei der Zuweisung

```

Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles
    MyBase.Load
        If Not Page.IsPostBack Then
            Dim aList As ArrayList = New ArrayList()
            aList.Add(New Artikel("Ordner", 29.9, 2))
            aList.Add(New Artikel("Hefter", 9.9, 14))
            aList.Add(New Artikel("Federtasche", 24.9, 5))
            SimpleList.DataSource = aList
            SimpleList.DataBind()
        End If
    End Sub

```

Listing 6.33: Speicherung von Objekten in einer Liste
(Ausschnitt aus SimpleRepeater2.aspx.vb)

Der Container, der innerhalb des Repeater-Steuerelements den Zugriff gestattet, liefert nun Objekte aus – und zwar die von der Struktur abgeleiteten. Die Vorlage nutzt deshalb die erweiterte

Datenbindung in der Vorlage

Form der Datenbindungssyntax, wobei auf die Eigenschaften der Struktur direkt zugegriffen werden kann.

```
<asp:Repeater Runat="server" ID="SimpleList">
  <ItemTemplate>
    <li>
      <%# Container.DataItem.Name %>
      <br />
      Preis:
      <%# String.Format("&euro; {0:f}", Container.DataItem.Preis)
%><br>
      Am Lager:
      <%# Container.DataItem.Menge %> Stück
    </li>
  </ItemTemplate>
</asp:Repeater>
```

Listing 6.34: Abruf der Daten aus den Objekten (Ausschnitt aus SimpleRepeater2.aspx)

Die Auswahl der Datenquelle mit »Name«, »Preis« usw. entspricht der Definition der Eigenschaften in der Struktur. Da der Zugriff nur Lesend erfolgt, reicht der `Get`-Zweig aus. In diesem Fall müssen Sie Eigenschaften natürlich als `ReadOnly` kennzeichnen, sonst verlangt der Compiler den `Set`-Zweig. Sie sind aber ansonsten in der Gestaltung völlig frei, abgesehen davon, dass die Eigenschaften `Public` sein müssen, was bei VB.NET standardmäßig der Fall ist. Das die Struktur selbst als `Private` deklariert wurde, ist völlig korrekt, denn auf die Definition muss von außerhalb nicht zugegriffen werden – nur auf die Instanzen.

6.6.3 Vorbereiten einer individuellen Datenquelle

Wenn Sie eigene Klassen entwickeln, deren Objekte Sammlungen gleichartiger Daten enthalten, können diese ebenfalls an Daten-Steuerelemente gebunden werden. Dazu ist die Implementierung der Schnittstelle `IEnumerable` notwendig. Das Implementieren von Schnittstellen ist ein typischer Vorgang, der im Buch immer wieder exemplarisch gezeigt wird. Auf den ersten Blick mag dies umständlich erscheinen. Wenn Sie eine fertige Klasse finden, besteht auch kein Grund, eine eigene zu entwickeln. Reicht das Angebot aber nicht aus, sollten Sie Schnittstellen verwenden. Denn Ihre neue Klasse passt dann optimal zu anderen Objekten, die darauf ausgelegt sind, eine ganz bestimmte Struktur vorzufinden.

Das folgende Beispiel durchsucht ein Verzeichnis und listet alle enthaltenen Dateien mit Hilfe des `Repeater`-Steuerelementes auf. Gegenüber einem einfachen Array werden verschiedene Dateiinformationen gespeichert. Nach der Größe einer Datei kann das Array sortiert werden. Für die Sortierfunktion wird `IComparer` implementiert, für die Ausgabe der Ergebnisse `IEnumerable`, für die Speicherung der Kollektion `IEnumerator`.

**IComparer
IEnumerable**

Das Beispiel *CollectionImplementation* zeigt die vollständige Implementierung. Alle Teile dieses Programms werden nachfolgend ausführlich vorgestellt. Es besteht aus insgesamt drei Dateien; die ASPX-Datei *CollectionImplementation.aspx*, die die `Repeater`-Steuerelemente enthält. Dann die Code Behind-Datei *CollectionImplementation.aspx.vb*, die Verzeichnisinformationen ausliest und in speziellen Klassen speichert. Die gesamte Definition der Hilfsklassen wurde nach *FileCollectionClass.vb* ausgelagert.

```
<asp:Repeater Runat="server" ID="filelist1">
  <HeaderTemplate>
    <ul>
  </HeaderTemplate>
  <ItemTemplate>
    <li><a href="<%# Container.DataItem.FileName %>">
      <%# Container.DataItem.FileName %></a>
      (<%# Container.DataItem.FileSize %> Bytes)
    </li>
  </ItemTemplate>
  <FooterTemplate>
    </ul>
  </FooterTemplate>
</asp:Repeater>
```

Listing 6.35: Eines der beiden `Repeater`-Steuerelemente (Ausschnitt aus *CollectionImplementation.aspx*)

Die Schreibweise der Datenbindungsaufrufe deutet an, dass das gelieferte Objekt über Eigenschaften wie *FileName* oder *FileSize* verfügen muss. Es soll in diesem Programm außerdem sortierte Listen ausgeben. Die Tatsache, dass `Repeater` verwendet werden, erfordert außerdem die Implementierung entsprechender Aufzählungsmethoden. Es ist sinnvoll, sich dem Problem systematisch zu widmen. Insgesamt ist die Implementierung nicht schwierig und doch enorm leistungsfähig. Am Anfang erschließt sich jedoch die Logik dahinter nicht sofort.

Werfen Sie zuerst einen Blick auf die Klasse, die die Repeater-Steuerelemente mit den entsprechenden Werten füllt:

```
Public Class StartCollection
    Inherits System.Web.UI.Page
    Const path As String = "."
    Public filelist1 As Repeater, filelist2 As Repeater

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim fi As FileInfo
        Dim fc As FileCollection = New FileCollection()
        Dim currentpath As String
        currentpath = Server.MapPath(path)
        Dim filename As String
        For Each filename In Directory.GetFiles(currentpath, _
            "*.aspx")

            fi = New FileInfo(filename)
            Dim f As FileObject = New FileObject(fi.Name)
            f.FilePath = fi.DirectoryName
            f.FileExtension = fi.Extension
            f.FileSize = fi.Length
            fc.AddFile(f)
        Next
        If fc._length > 0 Then
            fc.SortSizeDown()
            filelist1.DataSource = fc
            filelist1.DataBind()
            fc.SortSizeUp()
            filelist2.DataSource = fc
            filelist2.DataBind()
        End If
    End Sub
End Class
```

*Listing 6.36: So werden Dateinamen ausgelesen und dem Repeater übergeben
(CollectionImplementation.aspx.vb)*

Wie es funktioniert

Die Klasse besteht praktisch nur aus der Methode `Page_Load`. Dort sind zwei Variablen von Bedeutung. `fi` vom Typ `FileInfo` enthält die jeweils aktuellen Dateiinformatoren, die dann in einem Objekt der Klasse `fc` – Typ `FileCollection` – gespeichert werden. Die Objekte selbst entstammen einer weiteren Klasse, `FileObject`.

Während `FileInfo` eine Standardklassen aus dem Namensraum `System.IO` ist, werden `FileCollection` und `FileObject` komplett selbst implementiert. Die Aufgabe des Programms besteht darin, ein gegebenes Verzeichnis nach Dateien mit der Endung `«.aspx»`

zu durchsuchen und diese auszugeben, und zwar sortiert nach der Dateigröße. Dies hat weniger einen praktischen als mehr einen demonstrativen Sinn. Das Durchlaufen der Dateinamen sieht folgendermaßen aus:

```
For Each filename In Directory.GetFiles(currentpath, "*.aspx")
```

Für die jeweils ermittelte Datei werden nun die Dateiinformati-
onen beschafft:

```
fi = New FileInfo(filename)
```

Daraus wird ein Dateiojekt erstellt:

```
Dim f As FileObject = New FileObject(fi.Name)
```

Nun folgen die Zuweisungen der Eigenschaften und am Schluss wird das fertige Objekt der Kollektion *fc* hinzugefügt:

```
fc.AddFile(f)
```

Die Ausgabe der Dateisammlung erfolgt nach dem Sortieren:

```
fc.SortSizeDown()
```

Um die Darstellung müssen Sie sich nicht kümmern. Dank der Implementierung der richtigen Schnittstellen reicht die Angabe der Kollektion als Datenquelle:

```
filelist1.DataSource = fc  
filelist1.DataBind()
```

Der spannende Code steckt offensichtlich in der Implementierung der schon erwähnten Klassen *FileCollection* und *FileObject*. Tatsächlich werden noch zwei weitere benötigt, die nachfolgend gezeigt werden. Zuerst ist die Klasse *FileObject* von Interesse. Hierin werden die Dateiinformati-
onen gespeichert. Objekte dieser Klasse lagern später als Elemente in der Kollektion vom Typ *FileCollection*. Die Implementierung sieht folgendermaßen aus:

```
Public Class FileObject  
  
    Private _size As Long  
    Private _name As String  
    Private _ext As String  
    Private _path As String  
  
    Sub New(ByVal name As String)  
        _name = name  
    End Sub
```

```
Property FileExtension() As String
    Get
        Return _ext
    End Get
    Set(ByVal Value As String)
        _ext = Value
    End Set
End Property

Property FilePath() As String
    Get
        Return _path
    End Get
    Set(ByVal Value As String)
        _path = Value
    End Set
End Property

Property FileName() As String
    Get
        Return _name
    End Get
    Set(ByVal Value As String)
        _name = Value
    End Set
End Property

Property FileSize() As Long
    Get
        Return _size
    End Get
    Set(ByVal Value As Long)
        _size = Value
    End Set
End Property
End Class
```

Listing 6.37: Implementierung von FileObject (Ausschnitt aus FileCollectionClass.vb)

Speicherklasse Hierzu gibt es wenig bemerkenswertes; letztlich ist es eine reine Speicherklasse mit einfachen Eigenschaften. Als nächstes kommt die Implementierung der Schnittstelle `IEnumerable` an die Reihe – `FileCollection`. Diese Klasse wird von der Hauptapplikation direkt verwendet.


```

Public Class FileCollection
    Implements IEnumerable

    Public _files As FileObject() = New FileObject() {}
    Public _length As Integer = 0

    Public Function GetEnumerator() As IEnumerator _
        Implements IEnumerable.GetEnumerator
        Return New FileEnumerator(Me)
    End Function

    Public Sub SortSizeUp()
        Array.Sort(_files, New FileSizeComparer(True))
    End Sub

    Public Sub SortSizeDown()
        Array.Sort(_files, New FileSizeComparer(False))
    End Sub

    Public Sub AddFile(ByVal f As FileObject)
        If _length = _files.Length Then
            ' Array erweitern
            Dim _Newfiles() As FileObject
                = New FileObject(_files.Length + 1) {}
            Array.Copy(_files, 0, _Newfiles, 0, _length)
            _files = _Newfiles
        End If
        _files(_length) = f
        _length = _length + 1
    End Sub
End Class

```

Listing 6.38: Die Kollektion mit den speziellen Sortierfunktionen (Ausschnitt aus *FileCollectionClass.vb*)

Die Methode `AddFile` fügt der Kollektion, die intern als Array definiert ist, ein weiteres Element hinzu. Dazu wird die Größe des Arrays jeweils um ein Element erhöht. Das Verfahren dient nur der Demonstration (und erleichtert die Nutzung des Debuggers). In einer »echten« Implementierung ist es sinnvoller, immer mehrere leere Arrayelemente anzufügen und dann nur in größeren Schritten den Umkopiervorgang zu starten, der sich auf folgende Zeile beschränkt:

```
Array.Copy(_files, 0, _Newfiles, 0, _length)
```

Das private Feld `_length` enthält die aktuelle Größe der Liste.

Die Klasse liefert auch die Sortierfunktionen. Diese basieren zum einen auf dem Aufruf der statischen Methode `Sort` der Klasse `Array`.

```
Array.Sort(_files, New FileSizeComparer(True))
```

Da die standardmäßige Sortierung hier nicht verwendbar ist, wird der Sortiermethode eine Instanz derjenigen Klasse übergeben, die letztlich mit den speziellen Dateiobjekten umgehen kann. Logischerweise muss man das selbst implementieren. Die Klasse `FileSizeComparer` wird in Listing 6.39 gezeigt. Zuvor jedoch noch ein Blick auf die Aufzählungsmethode, die das `Repeater`-Steuerelement benötigt:

```
Public Function GetEnumerator() As IEnumerator _
    Implements IEnumerable.GetEnumerator
    Return New FileEnumerator(Me)
End Function
```

Der `Repeater` erwartet, dass eine Methode `GetEnumerator` bereit steht, die ein weiteres Aufzählungsobjekt liefert. Listing 6.40 zeigt dessen vollständige Implementierung. Der Trick besteht darin, dass dieses Objekt Methoden enthalten muss, die eine sequenzielle Abarbeitung der Elemente erlauben.

Sortierklasse Zuvor soll jedoch die Sortierklasse vorgestellt werden:

```
Public Class FileSizeComparer
    Implements IComparer

    Private _up As Integer, _down As Integer
    Private _a As FileObject, _b As FileObject

    Sub New(ByVal direction As Boolean)
        If direction Then
            _up = 1
            _down = -1
        Else
            _up = -1
            _down = 1
        End If
    End Sub

    Function CompareFileSize(ByVal a As Object, 
                                ByVal b As Object) As Integer _
    Implements IComparer.Compare
        _a = CType(a, FileObject)
        _b = CType(b, FileObject)
        If _a.FileSize > _b.FileSize Then
```

```

        Return _down
    End If
    If _a.FileSize < _b.FileSize Then
        Return _up
    End If
    Return 0
End Function

```

End Class

Listing 6.39: Die Implementierung der Schnittstelle IComparer für individuelle Sortierfunktionen (Ausschnitt aus FileCollectionClass.vb)

Die Besonderheit besteht hier in zwei Funktionen. Zum einen soll nach der Dateigröße sortiert werden, was standardmäßig nicht möglich ist. Zum anderen soll die Sortierrichtung – auf- oder absteigend – frei gewählt werden können. Damit dennoch nur eine Kodierung notwendig ist, wird der Sortierklasse im Konstruktor ein Parameter übergeben, der die Sortierung bestimmen wird:

Wie es funktioniert

```
Sub New(ByVal direction As Boolean)
```

Ist dieser Wert `True`, werde die Zahlenwerte `_up` und `_down` so gesetzt, dass die Rückgabewerte aufsteigend bzw. absteigend sortiert werden können, entsprechend dem Parameter des Konstruktors beim Aufruf:

```
Array.Sort(_files, New FileSizeComparer(True))
```

Die Sortierfunktion, die die Methode `Compare` implementiert, muss mit zwei Parametern vom Typ `Object` definiert werden. Da im Beispiel `FileObject`-Objekte übergeben werden, findet eine explizite Konvertierung statt:

```
_a = CType(a, FileObject)
```

Um den Vergleich auszuführen, wird nun auf die Eigenschaft `FileSize` dieser Objekte zugegriffen:

```
If _a.FileSize > _b.FileSize Then
```

Behandelt werden die Zustände Größer, Kleiner und Gleich. Die Rückgabewerte müssen -1, 1 oder 0 sein, wobei die Werte -1 und 1 vertauscht werden, um die Sortierrichtung zu ändern.

Als letzten Schritt für die Umsetzung der gesamten Lösung wird die Aufzählungsklasse geschrieben. Dies sind die Objekte, die die Methode `GetEnumerator` zurückgibt. Damit der `Repeater` funktio-

niert, muss es hier eine Logik geben, nach der eine Kollektion durchlaufen werden kann. Dazu sind drei Methoden erforderlich:

► `MoveNext()`

Diese Methode bewegt einen internen Zeiger auf die Kollektion einen Schritt weiter. Sie liefert `True`, wenn weitere Elemente verfügbar sind. Steht der Zeiger am Ende oder ist die Kollektion leer, soll `False` zurückgegeben werden.

► `Current`

Diese Eigenschaft gibt das aktuelle Element zurück. Um die Kollektion zu durchlaufen und die Werte zu entnehmen, ist wechselseitig `MoveNext` und `Current` aufzurufen.

► `Reset`

Diese Methode setzt den Zeiger auf die Elemente wieder auf das erste Element.

Beachten Sie, dass es sich hier lediglich um eine Hilfsklasse zum Auslesen einer Kollektion handelt, nicht um die Kollektion selbst. Deshalb werden Methoden zum Auffüllen oder Leeren nicht benötigt. Dies wurde bereits in `FileCollection` realisiert. Tatsächlich wird dem Objekt vom Typ `FileEnumerator` über den Konstruktor der Klasse ein Objekt vom Typ `FileCollection` übergeben. Das nachfolgende Listing zeigt die Implementierung:

```
Public Class FileEnumerator
    Implements IEnumerator

    Private _fc As FileCollection
    Private _index As Integer

    Sub New(ByVal fc As FileCollection)
        _fc = fc
        _index = 0
    End Sub

    Default ReadOnly Property Item(ByVal index As Integer) As FileObject
    Get
        If index < _length Then
            Return _files(index)
        End If
    End Get
End Property
```

```

Function MoveNext() As Boolean _
Implements IEnumerator.MoveNext
    If _fc._length = 0 Then
        Return False
    End If
    _index = _index + 1
    If _index = _fc._length Then
        Return False
    End If
    Return True
End Function

ReadOnly Property Current() As Object _
Implements IEnumerator.Current
    Get
        Return _fc._files(_index)
    End Get
End Property

Overridable Overloads Sub Reset() _
Implements IEnumerator.Reset
    _index = 0
End Sub

End Class

```

*Listing 6.40: FileEnumerator, eine Klasse zum Auslesen einer Kollektion
(Ausschnitt aus FileCollectionClass.vb)*

Die Implementierung ist genauso minimalistisch, wie es die Definition der Schnittstelle erfordert. Im Gegensatz zur Verwendung eines Arrays werden hier aber die selbst definierten Objekte verarbeitet, was der gesamten Applikation zwei Effekte verleiht: Zum einen ist der Code streng an den eigenen Erfordernissen ausgerichtet, zum anderen dennoch kompatibel zum Framework, wodurch sich die Datenbindung mit dem `Repeater`-Steuerelement so einfach gestaltet. Mit allen anderen derart datengebundenen Steuerelementen funktioniert das natürlich ebenso. Dies ist ein Grund für den Einsatz von Schnittstellen.

Die Abbildung zeigt, wie sich das Programm verhält, wenn in dem betroffenen Verzeichnis ein paar Dateien liegen.

Der Umgang mit Schnittstellen und Aufzählungen zur Datenbindung gilt gleichermaßen auch für die Steuerelemente `DataGrid` und `DataList`. Diese erlauben vor allem komplexere Gestaltungen durch eine Vielzahl von Attributen und Eigenschaften. Sie werden nachfolgend vorgestellt.

Willkommen bei ASP.NET mit VB.NET

Startseite :: Projekt- und Programmübersicht

Anzeige, sortiert nach Größe der Dateien (oben aufsteigend, unten absteigend).

- [treenodes.xml](#) (243 Bytes)
- [region2treeview.xslt](#) (665 Bytes)
- [regions.xsd](#) (1249 Bytes)
- [universal.xslt](#) (1403 Bytes)
- [regions.xml](#) (5155 Bytes)

- [universal.xslt](#) (1403 Bytes)
- [regions.xsd](#) (1249 Bytes)
- [region2treeview.xslt](#) (665 Bytes)
- [treenodes.xml](#) (243 Bytes)
- [regions.xsx](#) (212 Bytes)

Abbildung 6.26: Sortierte Ausgabe von Dateilisten über ein *Repeater*-Steuerelement

6.6.4 Automatische Listen mit DataList

Das `DataList`- unterscheidet sich vom `Repeater`-Steuerelement durch einige Erweiterungen. Die bereits gemachten Aussagen gelten unverändert. Prinzipiell ist die Darstellung nicht mehr völlig frei wählbar, sondern erlaubt die Wahl zwischen zwei Anzeigeprinzipien.

`DataList` arbeitet zeilenorientiert, die Vorlagen beziehen sich also immer auf eine Reihe in der Auflistung. Ob diese aus einer oder mehreren Spalten besteht, ist nur eine Frage der Gestaltung. Im Gegensatz dazu arbeitet das `DataGrid` spaltenorientiert (siehe Abschnitt 6.6.5, »DataGrid« ab Seite 605).

Hinzu kommen zwei weitere Vorlagen:

► <SelectedItemTemplate>

Diese Vorlage bereitet den Bearbeitungsmodus vor. Mit der Auswahl einer Zeile der Datenliste können beispielsweise Optionen eingeblendet werden, die auf die ausgewählten Daten anwendbar sind.

► <EditItemTemplate>

Wenn Daten bearbeitet werden können, ist die Auswahl dieser Vorlage angebracht. Praktisch verbergen sich dahinter meist `TextBox`-Steuerelemente.

Ein Überblick über die Struktur der `DataList`-Vorlagen zeigt Abbildung 6.23.

Es gibt natürlich auch Ereignisse, die den Auswahl- bzw. Bearbeitungsmodus starten, abbrechen oder gültig beenden.

Ereignisse

Neu sind auch die Stilvorlagen, mit denen den Vorlagen Gestaltungsattribute hinzugefügt werden können. Intern werden diese Attribute in Cascading Style Sheets umgesetzt, sodass jeder moderne Browser das Ergebnis verarbeiten kann.

Stilvorlagen

Die Gestaltung kann auf zwei Ebenen erfolgen. Es stehen Attribute für das gesamte Steuerelement zur Verfügung, die im `<asp:repeater>`-Tag definiert werden.

Gestaltung des Steuerelementes in der Vorlage

In der Vorlage werden verschiedene Attribute eingesetzt, um das gesamte Steuerelement oder einen Teil davon zu gestalten. Dabei stehen sowohl direkte Attribute, wie beispielsweise `BackColor` zur Verfügung, als auch Kollektionen von Stilattributen, die Teil von Cascading Style Sheets (CSS) sind. Die beiden Verfahren führen teilweise zum selben Ergebnis, den intern setzt ASP.NET alles in CSS um. Allerdings interagieren die Angaben nicht. Wenn Sie eine Eigenschaft wie `background-color` im CSS-Attribut setzen, ändert sich die Eigenschaft `BackColor` nicht automatisch, obwohl im Endeffekt dasselbe Stilattribut verwendet wird. Der programmtechnische Zugriff auf die Eigenschaften ist deshalb mit Vorsicht zu verwenden.

Reine globale Stilattribute können Sie der folgenden Tabelle entnehmen:

Stilattribute

Attribut	Parameter	Bedeutung	Hinweise
CellPadding	Zahl in Pixel	Abstand des Zellinhalts vom Zellrand	
CellSpacing	Zahl in Pixel	Abstand der Zellen voneinander	
GridLines	None, Horizontal, Vertical, Both	Zeichnet Linie in der Tabelle	

Tabelle 6.9: Stilattribute für das gesamte Element

Attribut	Parameter	Bedeutung	Hinweise
BackImageUrl	URL	Hintergrundbild	
BorderStyle	dashed, dotted etc.	Art des Randes des gesamten Steuer-elementes	BorderStyle-Struktur, beispielsweise BorderStyle.Dotted
BorderWidth	Pixel	Breite des Randes	
Font-Name, Font-Names	Schrift-artnamen	Name einer oder einer Liste von Schriftarten	Die Eigenschaften der Objekte heißen Font.Name usw. Font stammt von FontInfo ab. Die Größe erwartet Zahlen in Punkt oder Mitglieder von FontUnit, beispielsweise FontUnit.XXLarge.
Font-Size	Schriftgröße	Schriftgröße mit Angabe der Einheit (pt, cm, mm, px usw.)	
Font-Bold	True, False	Modifikation der Schriftart	
Font-Italic			
Font-Underline			
Font-Strike			
CssClass	Klassenname	Name einer Klasse, die in einer externen Stildatei definiert wurde	Zeichenkette
BackColor	Farb-bezeichner	Hintergrundfarbe	Farben müssen der Color-Struktur entstammen, beispielsweise Color.Red oder Color.FromArgb (0xFFCC23)
BorderColor	Farb-bezeichner	Randfarbe	
ForeColor	Farb-bezeichner	Schriftfarbe	
HorizontalAlign	Center, Justify, Left, Right	Horizontale bzw. vertikale Ausrichtung der Zellen im betroffenen Bereich der Tabelle	Die Eigenschaftszuweisung erwartet Elemente der Struktur HorizontalAlign bzw. VerticalAlign, beispielsweise HorizontalAlign.Center
VerticalAlign	Bottom, Middle, Top		

Tabelle 6.9: Stilattribute für das gesamte Element (Forts.)

Gestaltung der Vorlagen

Für die Gestaltung der Vorlagen gibt es zwei Möglichkeiten. Entweder geben Sie im Kopf des Steuerelementes weitere Stilattribute an. Diese unterscheiden sich von den allgemeinen nur durch einen entsprechenden Präfix: `ItemStyle`, `AlternatingItemStyle`, `SelectedItemStyle`, `EditItemStyle`, `FooterStyle`, `HeaderStyle`. Die Hintergrundfarbe des Kopfbereiches wird dann durch `HeaderStyle-BackColor="farbname"` festgelegt. Alternativ gibt es reine Stil-

vorlagen, die denselben Effekt haben. Was Sie verwenden, ist Geschmackssache. Die Stilvorlagen sind etwas übersichtlicher:

```
<HeaderStyle BackColor="red" HorizontalAlign="Center"/>
```

Der Name der Stilvorlage entspricht dem bereits genannten Präfix. Das Attribut `runat="server"` gibt es hier nicht, weil diese Vorlagen nicht alleinstehend sondern nur im Kontext eines anderen Steuerelements auftreten können.

Sie müssen die Vorlage `AlternatingItem` nicht definieren, um `AlternatingStyle` verwenden zu können. Wenn Sie `AlternatingStyle` und `ItemStyle` verwenden, werden die Zeilen abwechselnd damit formatiert, auch wenn nur `ItemTemplate` existiert. In den meisten Fällen reicht dies aus, um den gewünschten Effekt zu erzielen, beispielsweise einen farbig wechselnden Hintergrund.



Das folgende Beispiel nutzt die bereits im letzten Abschnitt erstellten Klassen zum Auflisten von Dateien, diesmal im `DataList`-Steuerelement:

```
<asp:DataList Runat="server" ID="dataList" ␣
    GridLines="Horizontal">
    <HeaderStyle HorizontalAlign="Center" ␣
        BackColor="#ffffcc" ␣
        Font-Name="Verdana" Font-Bold="True"/>
    <ItemStyle Font-Name="Verdana"/>
    <HeaderTemplate>
        Dateien im aktuellen Verzeichnis
    </HeaderTemplate>
    <ItemTemplate>
        <%# Container.DataItem.FileName %>
        (<%# Container.DataItem.FileSize %>)
    </ItemTemplate>
</asp:DataList>
```

Listing 6.41: Datenliste ausgeben (`DataListFiles.aspx`, der Code dazu ist in `DataListFiles.aspx.vb` zu finden)

Die Umsetzung in HTML erfolgt standardmäßig mit einer Tabelle und eingebetteten Stilanweisungen und HTML-Attributen:

HTML

```
<table id="dataList" cellspacing="0" border="0"
    rules="rows" border="1"
    style="border-collapse:collapse;">
<tr>
    <td align="Center"
        style="background-color:#FFFFCC; font-family:Verdana;
```

```

        font-weight:bold;">
        Dateien im aktuellen Verzeichnis
    </td>
</tr><tr>
    <td style="font-family:Verdana;">
        DataBinder.aspx
    </td>
</tr>
...

```

Wenn Sie in den HTML-Teilen innerhalb der Vorlagen weitere Stile oder HTML-Tags definieren, sollten Sie die Einbettung in die Tabellenzellen kennen.

Dateien im aktuellen Verzeichnis
treenodes.xml (243)
region2treeview.xslt (665)
regions.xsd (1249)
universal.xslt (1403)
regions.xml (5155)

Abbildung 6.27: Gestaltete Liste mit dem *DataList*-Steuerelement

Wenn Sie sehr lange Listen ausgeben, kann die Aufteilung auf mehrere Spalten sinnvoll sein. Dazu sind weitere Attribute zuständig:

► RepeatColumns="anzahl"

Hiermit wird die Anzahl der Spalten angegeben. Die Daten der eindimensionalen Datenliste werden dann gleichmäßig über alle Spalten verteilt. Leere Spalten werden nicht angezeigt; wenn nicht genug Daten da sind, erscheinen entsprechend weniger.

► RepeatDirection

Geben Sie hier die Richtung an, in der die Elemente auf die Spalten verteilt werden. Möglich sind die Parameter *Vertical* und *Horizontal*.

1	2	3
4	5	6

Abbildung 6.28: Horizontale Ausrichtung der Elemente: *RepeatDirection.Horizontal*

1	3	5
2	4	6

Abbildung 6.29: Vertikale Ausrichtung der Elemente: *RepeatDirection.Vertical*

Bislang basierte die Ausgabe immer auf Tabellen. In den meisten Fällen ist das auch ausreichend. Wenn Sie eine etwas freiere Gestaltung wünschen, können Sie das Attribut `RepeatLayout` mit dem Parameter `Flow` verwenden. Dann werden Sequenzen aus `<span>`-Tags erzeugt. Zeilenumbrüche, die dazwischen notwendig sind, werden mit `<br>`-Tags umgesetzt. Die Attribute, die für Tabellen typisch sind, wie `GridLines` oder `Cellpadding`, werden ignoriert.

RepeatLayout

In den bisherigen Darstellungen wurden immer nur eindimensionale Listen verarbeitet. Genau genommen ist die Kollektion der `FileObject`-Objekte jedoch zweidimensional, denn jedes Objekt enthält verschiedene Eigenschaften. Eine mehrspaltige Ausgabe ist deshalb angebracht.

Mehrspaltige Ausgaben

Dateiname	Größe
DataBinder.aspx	655
SimpleRepeater.aspx	1019
DataListFiles.aspx	1153
SimpleRepeater2.aspx	1232
DataListFilesExtractRows.aspx	1717
CollectionImplementation.aspx	1782

Abbildung 6.30: Zweispartige Ausgabe mit *DataList*

Der Einbau mehrerer Spalten scheitert auf den ersten Blick. Wenn Sie nur Tabellenelemente ergänzen, stehen diese im Konflikt mit den automatisch erzeugten. Der Inhalt von `<ItemTemplate>` steht immer zwischen `<td></td>`-Paaren. Das Hinzufügen eines weiteren Paares bringt die Tabellen durcheinander. Im fließenden Layout entsteht zwar eine richtige Tabelle, die `<span>`-Tags sind jedoch weiter vorhanden und stehen zwischen den Reihen, was in HTML nicht zulässig ist.

Probleme mit mehreren Spalten

Die Lösung ist in einem weiteren Attribut zu finden: `ExtractTemplateRows`. Wird der Parameter auf `True` gesetzt, extrahiert das Steuerelement vollständige Tabellendefinitionen in den Vorlagen

Umgang mit Tabellen

und setzt daraus eine neue Tabelle zusammen. Die Definition der in Abbildung 6.30 gezeigten Version sieht folgendermaßen aus:

```
<asp:DataList Runat="server" ID="dataList"
    GridLines="Both" ExtractTemplateRows="true">
    <HeaderStyle HorizontalAlign="Center" BackColor="#ffffcc"
        Font-Name="Verdana" Font-Bold="True" />
    <ItemStyle Font-Name="Verdana" />
    <HeaderTemplate>
        <asp:table Runat="server">
            <asp:TableRow Runat="server">
                <asp:TableCell Runat="server" ID="Tablecell14">
                    Dateiname</asp:TableCell>
                <asp:TableCell Runat="server" ID="Tablecell13">
                    Größe</asp:TableCell>
            </asp:TableRow>
        </asp:table>
    </HeaderTemplate>
    <ItemTemplate>
        <asp:table Runat="server" ID="Table1">
            <asp:TableRow Runat="server" ID="Tablerow1">
                <asp:TableCell Runat="server" ID="Tablecell11">
                    <%# Container.DataItem.FileName %>
                </asp:TableCell>
                <asp:TableCell Runat="server" ID="Tablecell12"
                    HorizontalAlign="Right">
                    <%# Container.DataItem.FileSize %>
                </asp:TableCell>
            </asp:TableRow>
        </asp:table>
    </ItemTemplate>
</asp:DataList>
```

Listing 6.42: Ausgabe der Tabelle (DataListFilesExtractRows.aspx, als Code-Datei wurde erneut DataListFiles.aspx.vb verwendet, die etwa dem Code in Listing 6.36 entspricht)

Zwei Dinge sind hier zu beachten: Erstens müssen Sie in den Vorlagen jeweils vollständige Tabellen definieren, die den gesamten zu wiederholenden Bereich enthalten. Zweitens muss das Server-Steuerelement `Table` verwendet werden und darauf aufbauend natürlich `TableRow` bzw. `TableCell`. Die globalen Attribute der Stilvorlagen werden übernommen, eigene Definition in den Tabellentags überschreiben diese aber. Im Beispiel wurde mit `HorizontalAlign="Right"` die Ausrichtung der Zellen der rechten Spalte geändert, sodass die Größenangabe rechtsbündig erscheint.

Gestaltung des Steuerelementes im Code

Die Gestaltung von Steuerelementen mit Attributen ist zwar leistungsfähig, aber nicht besonders flexibel. Wenn Sie Benutzern die Wahl eines Layouts oder nur der Farben ermöglichen wollen, müssen sie im Programm auf die Eigenschaften zugreifen. Dies ist nicht besonders schwer, weil die Attribute dieselben Namen wie die Eigenschaften haben und ähnliche Parameter verlangen. Allerdings beachten die Tag-Attribute Groß- und Kleinschreibung nicht und betrachten alle Parameter als Zeichenketten. Im Code-Teil gilt das nicht mehr, hier wird mit der üblichen Typstrenge vorgegangen.

Eine sehr flexible wenn auch langsame Methode, ist der Austausch der gesamten Vorlage. Sie können den Inhalt jeder Vorlage in einer externen Datei ablegen, die im selben Verzeichnis wie das Formular oder einem Unterverzeichnis gespeichert werden sollte. ASP.NET betrachtet diese Fragmente als Benutzer-Steuerelemente (User Controls). Damit sind folgende Besonderheiten verbunden:

Austauschen von Vorlagen

- ▶ Als Dateierweiterung ist zwingend `.ascx` erforderlich.
- ▶ Der Datenzugriff auf den Container nutzt eine modifizierte Syntax.

Benutzer-Steuerelemente werden von der Klasse `Control` abgeleitet. In dieser Klasse, die in der Hierarchie relativ weit oben steht, ist `Container.DataItem` noch nicht implementiert. Der Zugriff auf Datenelemente erfolgt deshalb mit dem Container alleine:

```
<%# DataBinder.Eval (Container, "DataItem.FileName") %>
```

Zur Auswahl der Datenquelle wird dann während der Laufzeit erst auf `DataItem` – das aktuelle abstrakte Objekt – und dann auf eines seiner Eigenschaften – im Beispiel `FileName` – zugegriffen. Ansonsten entspricht der Inhalt dem der Vorlage, muss also nicht durch eine `@Control`-Direktive ergänzt werden, wie das bei vollständigen Benutzersteuerelementen erforderlich ist.

Das eigentliche `DataList`-Steuerelement kann leer bleiben, wenn die Vorlage extern hinzugefügt werden:

```
<asp:DataList Runat="server" ID="dataList"/>
```

Das Hinzufügen geschieht beim Aufbau der Seite – vorzugsweise in `Page_Load` – und sieht folgendermaßen aus:

```
dataList.ItemTemplate ➤
    = Page.LoadTemplate ("Templates/DataListItem.ascx")
dataList.HeaderTemplate ➤
    = Page.LoadTemplate ("Templates/DataListHeader.ascx")
```

Die Methode `LoadTemplate` erwartet eine virtuelle Pfadangabe. Im Beispiel liegen die Vorlagen in einem Unterverzeichnis `/Templates`. Die Namen der Eigenschaften zum Hinzufügen der Elemente entsprechen denen der Vorlagen selbst. Die Schnittstelle, der die Vorlagen entstammen, heißt übrigens `ITemplate`. Wenn Sie diese implementieren, kann auch der Inhalt programmtechnisch erzeugt werden, was gegenüber externen Dateien einen Leistungsgewinn bewirkt. In Abschnitt 7.6, »Kundenspezifische Steuerelemente (Custom Controls)« ab Seite 706 finden Sie dazu weitere Informationen.

Stile im Programm erzeugen

Der nächste Schritt besteht darin, auch die Gestaltung programmtechnisch zu erzeugen. Es gibt verschiedene Eigenschaften, mit denen die aktuellen Einstellungen – die über Tagattribute vorgenommen wurden – abgerufen werden. Die Namen können wieder aus den Vorlagenbezeichnungen abgeleitet werden, beispielsweise `AlternatingItemStyle`. Über diese Eigenschaften sind die einzelnen Attribute zugänglich. Der folgende Ausschnitt aus `DataListCode.aspx.vb` zeigt, wie verschiedene Einstellungen vorgenommen werden:

```
dataList.Font.Name = "Verdana"
dataList.HeaderStyle.BackColor = Color.FromArgb(0xCCFFFF)
dataList.ItemStyle.BackColor = Color.FromArgb(0xFFCCCC)
dataList.AlternatingItemStyle.BackColor = Color.White
dataList.RepeatLayout = RepeatLayout.Table
dataList.GridLines = GridLines.Both
```



Um typische RGB-Farbangaben einsetzen zu können, benutzen Sie die Hexadezimale Schreibweise von Zahlen mit dem Präfix »&H« und die Methode `FromArgb`.

Da die gesamte Gestaltung im Code stattfindet, nimmt sich das Steuerelement im HTML-Teil bescheiden aus:

```
<asp:DataList runat="server" ID="dataList"/>
```

Bearbeiten von Elementen der Liste

Es ist aus Sicht der Benutzerführung immer optimal, Daten dort zu verändern, wo sie normalerweise angezeigt werden. Im Fall der Datenliste würde es sich anbieten, die Namen der angezeigten Dateien verändern zu können. Dazu sind folgende Maßnahmen notwendig:

1. Auswahl der Zeile, die bearbeitet werden soll
2. Einblenden eines Eingabefeldes anstelle des Namens
3. Bereitstellung von Bestätigungs- und Abbruchmöglichkeiten

Die Möglichkeit, derartige Elemente in das `DataList`-Steuerelement einzubauen, besteht standardmäßig. Wenn nur die Bearbeitungsoption von Interesse ist, dann reicht es aus, die Gestaltung des Bearbeitungszustandes in einer weiteren Vorlage zu definieren: `<EditItemTemplate>`. Natürlich muss das Steuerelement wissen, wann Sie wünschen, dass die betreffende Zeile in dieser Form angezeigt wird. Dazu sind weitere Attribute zu setzen. Zuerst muss es eine Option geben, mit der der Benutzer seinen Bearbeitungswunsch mitteilt:

```
<asp:LinkButton Runat="server" ID="EditButton"
    CommandName="edit" Text="Bearbeiten"/>
```

Entscheidend ist an diesem Link, dass das Kommando mit `CommandName` auf die Zeichenkette `edit` festgelegt wird. Das Steuerelement erkennt neben diesem noch die Werte `"cancel"` (Abbruchwunsch), `"delete"` (Löschen), `"select"` (Auswahl) aus. Groß- und Kleinschreibung ist hier wieder interessant. Das allein reicht aber noch nicht, denn die Aktion, die gewünscht wird, muss nun auch ausgeführt werden. Dazu ist Code erforderlich – die Verarbeitung des Klickereignisses. Damit das Steuerelement die passende Methode findet, wird das Attribut `OnEditCommand` definiert:

```
<asp:DataList ... OnEditCommand="EditFile" ... >
```

Der Name `EditFile` ist willkürlich gewählt. Sie müssen jetzt dazu die passende Methode schreiben (je nach Art der Nutzung wäre noch `Handles` zu ergänzen):

```
Public Sub EditFile(ByVal source As Object,
    ByVal e As DataListCommandEventArgs)
    dataList.EditItemIndex
    Dim As = dataList.DataBind()
End Sub
```

Eigenschaften Der gewünschte Effekt kommt zustande, indem Sie dem Steuerelement über die Eigenschaft `EditItemIndex` mitteilen, für welche Reihe die Vorlage `<EditItemTemplate>` genutzt werden soll. Wenn der Index auf -1 gesetzt wird, verschwindet die Vorlage wieder und `ItemTemplate` oder, wenn vorhanden, `AlternatingItemTemplate` wird wieder angezeigt. Der Index kann auf einen fortlaufenden Wert zwischen 0 und der Anzahl der Reihen minus Eins gesetzt werden. die ausgewählte Reihe kann durch Zugriff die Argumente des Ereignisses erfolgen, im Beispiel das Objekt *e*. Drei Eigenschaften sind verfügbar:

► **Item**

Diese Eigenschaft erlaubt den Zugriff auf das Datenelement dieser Zeile, dessen Eigenschaft `ItemIndex` ergibt dann den Index der ausgewählten Zeile.

► **CommandArgument**

Wenn Sie mit dem Attribut `CommandArgument` ein Argument zum Kommando übergeben haben, steht es in dieser Eigenschaft. Das nächste Beispiel zeigt die Verwendung.

► **CommandName**

Der Name des Kommandos, im Beispiel ist dies die Zeichenkette "edit".

Das Bearbeiten erschöpft sich natürlich nicht im Anzeigen eines Eingabefeldes. Nach der Änderung muss der Benutzer seinen Wunsch bestätigen oder abbrechen können. Der Inhalt der Vorlage zeigt, wie weitere Kommandos eingeführt werden:

```
<asp:TextBox Runat="server" ID="FileEditor"
    Font-Name="Verdana" Width="280"
    Text='<%# ((Addison.VBNet.WebForm.File)
        Container.DataItem).FileName %>' />
<asp:LinkButton Runat="server" ID="Button1"
    CommandName="update" Text="Aktualisieren"/>
<asp:LinkButton Runat="server" ID="Button2"
    CommandName="cancel" Text="Abbrechen"/>
```

Für beide Kommandos sind auch die entsprechenden Attribute im Kopf des Steuerelementes zu definieren:

```
<asp:DataList ...
    OnCancelCommand="CancelFile"
    OnUpdateCommand="UpdateFile"
... >
```


**Ereignisbe-
handlung**

Dazu sind wieder passende Methoden zu definieren, die die Klickereignisse bedienen. Zuerst soll `CancelFile` vorgestellt werden. Diese Methode muss nichts verarbeiten, lediglich der Index ist auf -1 zu setzen:

```
Public Sub CancelFile(ByVal source As Object, 
                        ByVal e As DataListCommandEventArgs)
    dataList.EditItemIndex = -1
    dataList.DataBind()
End Sub
```

Ähnlich ist auch `UpdateFile` aufgebaut, nur wird hier neben dem Rücksetzen des Index der Name der Datei verändert. Freilich hat dies nichts mit `DataList` zu tun, sondern ist nun Code aus dem eigentlichen Programm. Sie ahnen vielleicht schon, was hier entsteht. Es handelt sich um eine frühe Phase eines serverseitigen Dateieexplorers.

Das eigentlich knifflige ist jedoch der Zugriff auf die alten und neuen Werte der Liste. Zu dem Zeitpunkt, an dem die Ereignisse ausgelöst werden, wurde die Seite schon verarbeitet. Zuerst benötigen Sie eine Methodik, mit der auf das `TextBox`-Objekt zugegriffen werden kann. Dazu wird die Methode `FindControl` eingesetzt, die in der Ereignisquelle nach einem Steuerelement sucht:

```
Dim tb As TextBox = CType(e.Item.FindControl("FileEditor"), TextBox)
```

Jetzt steht das Objekt zur Verfügung und damit können Sie auch auf den neuen (!) Inhalt zugreifen:

```
tb.Text
```

Im Beispiel *DataListEditor.aspx.vb* wird zur Demonstration ein Label-Steuerelement mit dem alten und neuen Wert beschrieben. Bleibt also noch zu klären, wie auf den alten Wert zugegriffen werden kann. Eine Möglichkeit besteht darin, die Liste der Dateien im `FileCollection`-Objekt global zu verwalten. Dann ist der Zugriff innerhalb der Ereignisbehandlungsmethoden kein Problem, denn diese sind ja Mitglieder derselben Klasse:

```
Dim oldName As String = fc(e.Item.ItemIndex).FileName
```

Da die Anzahl der Reihen im `DataList`-Steuerelement exakt den Einträgen in `fc` entspricht (`fc` ist die globale Instanz der `FileCollection`-Klasse), kann der Index direkt verwendet werden. Die

Schreibweise ist zulässig, weil die Klasse, die im gesamten Abschnitt Verwendung findet, einen Indexer besitzt. Dies hat wiederum nichts mit `DataList` zu tun, sondern spart nur etwas Tipparbeit.

Umgang mit dem Anzeigestatus und zwei Alternativen dazu

Es ist sehr ineffizient, mit jedem Zugriff immer wieder das gesamte Verzeichnis zu lesen. Das gilt für jede Art Datenzugriff, der anstelle des Dateizugriffs erfolgt. Die folgenden Maßnahmen zur Leistungssteigerung sind vor allem bei kleineren Datenmengen, wie sie in `DataList`- oder `DataGrid`-Objekten zur Ansicht gelangen, von großer Bedeutung. Denn im Verhältnis zu der relativ geringen Datenmenge auf der Webseite ist der Aufwand zum Abrufen der Daten erheblich. Denken Sie daran, dass im laufenden Betrieb nicht ein Benutzer zugreift, sondern möglicherweise sehr viele.

Es gibt zwei grundlegende Lösungen für das Problem:

- Nutzung des Anzeigestatus (`ViewState`)
- Speicherung der Daten in Sitzungsvariablen

Benutzerabhängige Status- erhaltung mit `ViewState`

Normalerweise werden Benutzer individuelle Daten abrufen. Das heißt, Sie müssen eine Speicherung der Abfrage für jeden Benutzer vornehmen. Da oben bereits festgestellt wurde, dass der Inhalt einer Datenbankabfrage (oder der Verzeichnisliste) meist überschaubar groß ist, bietet sich der Browser als Speicherobjekt an. Sie können dazu direkt auf das `ViewState`-Objekt zugreifen. Das Speichern erfolgt nach der Fertigstellung des Objekts:

```
ViewState("dl") = fc
```

Das funktioniert immer dann so direkt, wenn die Instanzen der betroffenen Klasse – hier `fc` als Instanz von `FileCollection` – serialisierbar sind. Dies wird mit dem Attribut `<Serializable>` erreicht. Was passiert nun intern? Beim Aufbau der Seite wird der Zustand des Anzeigestatus in einer versteckten Variablen gespeichert. Schauen Sie sich den Quellcode der HTML-Seite im Browser an, sehen Sie den codierten Inhalt.



Abbildung 6.31: Die Liste im ViewState – auch so kann man Daten erhalten

Jetzt bleibt noch die Frage, wie die Daten wieder zurückgeholt werden. Bei den Server-Steuerelementen passiert das automatisch. Ihre selbstdefinierten ViewState-Einträge müssen Sie auch selbst wieder zurückholen. Im Else-Zweig, wo der Teil von Page_Load ausgeführt wird, wenn die Anforderung per POST kam, steht dann Folgendes:

```
dataList.DataSource = ViewState("dl")
dataList.DataBind()
```

Der Name »dl« ist übrigens willkürlich gewählt, er muss nur eindeutig sein.

Nachteilig bei dieser Methode ist der aufgeblähte Seiteninhalt. Die Daten werden quasi doppelt übertragen, einmal sichtbar und einmal kodiert. Da die Kodierung auf Base64 basiert, vergrößert sich der Code dabei nochmals um ca. 30%. Bei kleinen Datenmengen ist das aber noch unkritisch und belastet nicht Ihren Server.

Als zweite Variante sollen Sitzungsvariablen verwendet werden. Auf die Feinheiten wird in Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778 eingegangen. Der Ort, wo die Variablen gespeichert werden, variiert je nach Implementierung. Im einfachsten Fall ist dies der Hauptspeicher, was gegenüber den statischen Objekten

**Anzeigestatus
zurückholen**

**Statuserhaltung
mit Sitzungs-
variablen**

nur wenig Vorteile in Bezug auf den Speicherverbrauch bietet. Der große Unterschied ist die Art der Implementierung; Sitzungsvariablen sind sehr einfach verwendbar. Der Unterschied zur letzten Version ist minimal:

```
dataList.DataSource = Session("dl")
dataList.DataBind()
```

Die Speicherung erfolgt ebenso:

```
Session("dl") = fc
```

Ein Trick noch zum Abschluss. Synchronisieren Sie die Änderungen an der Datenquelle mit dem aktuellen `Session`-Objekt. Dann sparen Sie das erneute Lesen auch dann, wenn tatsächlich neue Daten auftreten. Allerdings sollte sich dieses Verfahren auf einfache Beispiele beschränken, weil die Gefahr einer mangelnden Synchronisation zwischen Datenquelle und Datensinke besteht. Für das letzte Beispiel würde der Zugriff folgendermaßen aussehen:

```
fc(e.Item.ItemIndex + 1).Filename = tb.Text
```

Der Zugriff auf die Kollektion `fc` in der gezeigten Form setzt voraus, dass eine Standardeigenschaft definiert wurde. Dies ist in der Klasse `FileCollection` die Eigenschaft `Item`. Vergleichen Sie dazu die Ausführungen zu `FileCollectionClass.vb` weiter vorn.

**Vollständige
Programme
finden Sie auf
der Buch-CD
und im Web**

Für die Aktualisierung der Sitzungsvariablen sorgt das Sitzungsmanagement automatisch. Ein Grund mehr, Sitzungsvariablen ernsthaft in Projekte mit einzubeziehen. Den vollständigen Code dieses Beispiels finden Sie in `DataListEditorSession.aspx` bzw. `DataListEditorSession.aspx.vb`.

Weitere Verfahren im Zusammenhang mit Ereignissen

In den letzten Beispielen wurden die eingebauten Ereignisse »Edit« oder »Update« verwendet. Nicht immer werden Datenlisten so eingesetzt. Um individuelle Ereignisse verarbeiten zu können, gibt es das universelle Attribut `OnItemCommand`. Die damit definierte Ereignisbehandlungsmethode reagiert auf alle Klickereignisse, die eine Schaltfläche, ein Kontrollkästchen oder Optionsfeld auslöst. Schaltflächen sind besonders geeignet, denn sie bieten, wenn sie von der Klasse `Button` abstammen, gleich zwei Steuerattribute.

Folgende Definition finden Sie im Abschnitt `<ItemTemplate>` des Beispiels *DataListEditorCommand.aspx*:

```
<asp:button CommandName="ToFileList" ⚡
    CommandArgument="add" ⚡
    Runat="server" ID="AddButton" ⚡
    Text="Hinzufügen"/>
<asp:button CommandName="ToFileList" ⚡
    CommandArgument="remove"
    Runat="server" ID="RemoveButton" ⚡
    Text="Entfernen"/>
```

Mit `CommandName` wird ein beliebiger Name als »Kommando« festgelegt. Dem Kommando kann dann noch mit `CommandArgument` ein Parameter mitgegeben werden. Im Beispiel heißt das Kommando »ToFileList«. Beide Schaltflächen lösen also dasselbe Kommando aus. Damit unterschiedliche Aktionen starten, wird das Kommandoargument verwendet. Für ein so einfaches Muster ist das nicht zwingend erforderlich, es dient hier nur der Demonstration. In der Praxis würde man bei wenigen Aktionen besser gleich verschiedene Kommandonamen verwenden.

**Kommandos
verarbeiten**

Die Definition der Ereignisbehandlungsmethode sieht folgendermaßen aus:

```
<asp:DataList ... OnItemCommand="DoCommand" ...>
```

Es muss nun noch eine Methode erstellt werden, die den Namen »DoCommand« trägt. Der Name ist natürlich wieder willkürlich gewählt. Das Beispiel nutzt die Kommandos, um ausgewählte Namen der Dateiliste in eine weitere Liste zu übernehmen, die unterhalb der Tabelle erscheint. Wie schon im letzten Abschnitt gezeigt, werden Sitzungsvariablen verwendet, um den Zustand der Liste zu erhalten. Die Ausgabe erfolgt mit einem `Repeater-Steuerelement` mit dem Namen `Selection`, das hier nicht weiter erläutert werden soll.

```
Public Sub DoCommand(ByVal source As Object, ⚡
    ByVal ec As DataListCommandEventArgs)
    fcSelect = CType(Session("fcSelect"), ArrayList)
    fc = CType(Session("fc"), FileCollection)
    Select Case ec.CommandName
        Case "ToFileList"
            Select Case ec.CommandArgument.ToString()
                Case "add"
                    fcSelect.Add(fc(ec.Item.ItemIndex).FileName)
                Case "remove"
```

```

        fcSelect.Remove(fc(ec.Item.ItemIndex).FileName)
    End Select
End Select
Selection.DataSource = fcSelect
Selection.DataBind()
Session("fcSelect") = fcSelect
End Sub

```

Listing 6.43: Verarbeitung der Kommandos der Schaltflächen (Ausschnitt aus `DataListEditorCommand.aspx.vb`)

Die `Page_Load`-Methode entspricht wieder in etwa der Variante aus Listing 6.36. Lediglich das Befüllen der Sitzungsvariablen mit den aktuellen Werten kommt hinzu:

```

Session("fc") = fc
Session("fcSelect") = fcSelect

```

Die Arbeitsweise basiert wesentlich auf der Verarbeitung von Kommandos. Dies wird nachfolgend genauer diskutiert.

Eigenschaften der Kommandos

Bei der Definition des Methodenkopfes ist zu beachten, dass das Ereignis vom Typ `DataListCommandEventArgs` sein muss. Dieses Ereignis besitzt mehrere Eigenschaften, die den Zugriff auf die entsprechenden Parameter der Schaltflächen haben:

► `CommandName`

Dies ist der Name des Kommandos – als Zeichenkette (Typ `String`) – wie er mit `CommandName` im `Button`-Steuerelement definiert wurde.

► `CommandArgument`

Dies ist ein optionales Arguments des Kommandos – als Zeichenkette (Typ `String`) – wie er mit `CommandArgument` im `Button`-Steuerelement definiert wurde.

► `CommandSource`

Dies ist die Datenquelle selbst, bei einer Schaltfläche vom Typ `System.Web.UI.WebControls.Button`. Sie können darüber auf Eigenschaften des Steuerelementes zugreifen, wenn dies erforderlich sein sollte.

► `Item`

Hiermit erlangen Sie Zugriff auf die Datenreihe des `DataList`-Steuerelementes. Im Beispiel wird die Eigenschaft `ItemIndex` der aktuellen Datenreihe benutzt, um auf das Element selbst

zugreifen zu können, dessen Daten wiederum der ursprünglichen und nun in der Sessionvariablen gespeicherten Kollektion entnommen werden.

Wenn Sie bisher mit `LinkButton` gearbeitet haben und nun auf `Button` umstellen, wird Ihr Programm möglicherweise nicht mehr funktionieren. Die fertigen Beispiele dieses Buches funktionieren, weil in jedem Fall der Aufbau der Liste nur dann erfolgt, wenn die Daten mit HTTP-GET angefordert werden. Beim folgenden Zurücksenden des Formulars (PostBack) wird nur auf die gespeicherte Liste zugegriffen. Für die Analyse und Ausgabe von Schaltflächen ist dies zwingend erforderlich. Der Render-Prozess kann bei einem erneuten Aufbau die ursprünglichen `Button`-Definitionen nicht mehr erkennen und erstellt einfache `<input type="submit">`-Schaltflächen daraus. Damit geht die PostBack-Funktionalität verloren. Der Browser sendet das Formular zwar ab, die Verknüpfung mit der Ereignisbehandlungsmethode ist jedoch weg.



Verkürzt dargestellt sieht das in der Methode `Page_Load` folgendermaßen aus:

```
Public Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    If Not IsPostBack Then
        Dim hc As HttpContext
        Dim fi As FileInfo
        Dim currentpath As String = Server.MapPath(path)
        Dim filename As String
        For Each filename In Directory.GetFiles(currentpath)
            fi = New FileInfo(filename)
            Dim f As FileObject = New FileObject(fi.Name)
            f.FilePath = fi.DirectoryName
            f.FileExtension = fi.Extension
            f.FileSize = fi.Length
            fc.AddFile(f)
        Next
        dataList.DataSource = fc
        Selection.DataSource = fcSelect
        Session("fc") = fc
        Session("fcSelect") = fcSelect
        Page.DataBind()
    Else
        fc = CType(Session("fc"), FileCollection)
        fcSelect = CType(Session("fcSelect"), ArrayList)
    End If
End Sub
```

Anwendung Der Effekt des gezeigten Programms besteht darin, dass Sie aus einer dynamischen Liste von Dateinamen mittels Schaltfläche welche auswählen können und diese einer weiteren Liste hinzugefügt werden. Ebenso einfach ist das Entfernen möglich. Ein typischer Einsatz wäre ein kleiner Shop mit nur einer Warengruppe, dessen Artikel durch eine »Ein-Klick-System« in den Warenkorb übernommen werden sollen.

DataList

Zugriff auf Einträge zur Laufzeit und Kommandoschaltflächen

Datei	Aktion	
DataBinder.aspx	Hinzufügen	Entfernen
DataListCodeDesign.aspx	Hinzufügen	Entfernen
DataListEditor.aspx	Hinzufügen	Entfernen
DataListEditorCommand.aspx	Hinzufügen	Entfernen
DataListEditorSession.aspx	Hinzufügen	Entfernen
DataListEditorViewState.aspx	Hinzufügen	Entfernen
DataListFiles.aspx	Hinzufügen	Entfernen
DataListFilesExtractRows.aspx	Hinzufügen	Entfernen
DataListTemplate.aspx	Hinzufügen	Entfernen
SimpleRepeater.aspx	Hinzufügen	Entfernen
SimpleRepeater2.aspx	Hinzufügen	Entfernen

- DataBinder.aspx
- DataListEditorSession.aspx
- DataListFiles.aspx
- DataListEditorViewState.aspx
- DataListEditorCommand.aspx
- SimpleRepeater.aspx
- SimpleRepeater.aspx

Abbildung 6.32: Steuerung einer Liste über Kommandoschaltflächen im *DataList*-Element



Wenn Sie mit Steuerelementen arbeiten, die keine *CommandName*-Attribute kennen, greifen Sie auf die Eigenschaften über *CommandSource* zu. Denken Sie daran, das Element vorher in den korrekten Typ zu konvertieren:

```
Dim cb As CheckBox = CType(ec.CommandSource, CheckBox)
```

Zusammenfassung

Dieser Abschnitt hat in umfassender Weise das *DataList*-Steuerelement und typische Techniken im Umgang damit vorgestellt. Das nächste Element, *DataGrid*, basiert auf ähnlichen Prinzipien, ist jedoch noch komplexer. Viele Eigenschaften sind mit *DataList* identisch, diese sollen nicht in dieser Ausführlichkeit wiederholt

werden. Blättern Sie also gegebenenfalls einfach zurück. Eine weitere Eigenschaft, die im Zusammenhang mit `DataGrid` beschrieben wird, gilt auch für `DataList`: Die zeilenweise Bindung der Daten kann ebenfalls ein Ereignis auslösen. Mehr dazu finden Sie im Abschnitt »Ereignisse zeilenweise an die Daten binden« ab Seite 614.

6.6.5 DataGrid

Das `DataGrid`-Steuerelement basiert, ebenso wie `DataList`, auf Vorlagen. Diese sind allerdings nicht nur zeilenweise orientiert, sondern erlauben auch eine spaltenweise Ausrichtung. Damit ergibt sich eine besonders hohe Flexibilität im Zusammenhang mit komplexeren Datenbankabfragen. Allerdings gilt zur Art der Datenquelle das zuvor bereits Gesagte – neben jeder Klasse, die `IEnumerable` implementiert, kann gebunden werden.

Die folgende Abbildung zeigt, wie das `DataGrid` prinzipiell definiert werden kann. Abgesehen vom `<Columns>`-Tag, dass die oberste Ebene bildet, können alle anderen so oft wiederholt werden, wie es die Darstellung des Gitters erfordert. Die Struktur der Vorlagen des `DataGrid`-Steuerelements finden Sie in Abbildung 6.24.

Aus der Komplexität ergibt sich, dass das `DataGrid`-Steuerelement intern immer auf einer Tabelle aufbaut. Wenn Sie für eine oder mehrere Spalten die Vorlagensteuerung benötigen, wie sie `DataList` bietet, wird als Spaltenvorlage `<TemplateColumn>` verwendet.

Elementare Eigenschaften

Das Steuerelement besitzt einige elementare Eigenschaften, die eine Menge Programmierarbeit sparen, wenn man sie richtig einsetzt:

- ▶ Seitenweise Darstellung der Datenquelle

Wenn Ihre Daten sehr viele Zeilen enthalten, kann die Anzeige seitenweise erfolgen. Das muss nicht programmiert werden; es genügt, dem Steuerelement die Art der Weiterschaltung und natürlich die Anzahl Zeilen pro Seite mitzuteilen.

- ▶ Sortierung von Spalten

Wenn die Datenquelle es erlaubt, kann eine Sortierung der Spalten bei der Ausgabe erfolgen.

Automatisches DataGrid

Beispiel Das Beispiel *DataGridSimplest.aspx* zeigt, wie einfach DataGrid verwendet werden kann. Die Definition ist auf das unbedingt Notwendige reduziert:

```
<asp:DataGrid Runat="server" ID="dataGrid" />
```

Die Verknüpfung mit der bekannten Datenquelle führt zu einem erstaunlichen Ergebnis.

DataGrid			
Einfachste Anwendung			
FilePath	FileExtension	FileName	FileSize
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataBinder.aspx	666
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataGridBoundColumns.aspx	970
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataGridSimplest.aspx	753
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListCodeDesign.aspx	746
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListEditor.aspx	2732
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListEditorCommand.aspx	2477
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListEditorSession.aspx	2828
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListEditorViewState.aspx	2831
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListFiles.aspx	1208
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListFilesExtractRows.aspx	1763
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	DataListTemplate.aspx	811
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	SimpleRepeater.aspx	1019
c:\inetpub\wwwroot\aspnet_csharp\Examples.aspx	.aspx	SimpleRepeater2.aspx	1232

Abbildung 6.33: Aufbau der Datenausgabe ohne Parameter – dies kann DataGrid allein

Es ist absehbar, dass Sie sich damit nicht zufrieden geben müssen. Wenn ein Steuerelement bereits ohne jede Vorgabe eine so komplexe Ausgabe erzeugt, kann man mit weiteren Techniken erstaunlich viel erreichen.

Auswahl und Aufbereiten der Spalten

**Automatische
Spalten unter-
drücken**

In der Standardform erzeugt DataGrid die Spalten automatisch. Wenn Sie weitere Spalten definieren, werden die automatisch generierten dennoch angehängt. Damit das nicht passiert, schreiben Sie das Attribut `AutoGenerateColumns="False"` in den Kopf des Tags. Das setzt dann voraus, dass Sie selbst Spalten definieren. Dazu wird die Vorlage `<Columns>` verwendet, die Spaltendefinitionen

umschließt. Als Definition folgt darin – jede Spalte einzeln aufgelistet – eine der folgenden Versionen:

- ▶ `<asp:BoundColumn>`
- ▶ `<asp:ButtonColumn>`

Einfache automatische Spalten mit `<asp:BoundColumn>`

Der Einsatz von `<asp:BoundColumns>` erfolgt nach folgendem Muster:

```
<asp:DataGrid ...  
  <Columns>  
    <asp:BoundColumn .../>    ...  
  </Columns>  
</asp:DataGrid>
```

Lassen Sie sich nicht davon täuschen, dass es sich offensichtlich um ein Server-Steurelement handelt. Sie können dieses nur im Zusammenhang mit `DataGrid` verwenden. Das Binden der Datenquelle ist sehr einfach, da sie auf die übliche Datenbindungssyntax verzichtet, solange Sie nicht von der Standardbindung abweichen. Allerdings entfallen damit auch alle Gestaltungsmöglichkeiten. Abgesehen davon können Sie alles selbst definieren und damit maximale Flexibilität in der Gestaltung erreichen.

Nur im `DataGrid` verwendbar

Folgende Attribute können eingesetzt werden:

- ▶ `DataField`

Hiermit wird die Spalte in der Datenquelle bestimmt. Bei Objekten, wie im Beispielcode, kann dies der Name einer Eigenschaft sein, bei SQL-Abfragen entspricht es den Spaltennamen der Tabelle oder Sicht.

- ▶ `DataFormatString`

Falls die Ausgabe formatiert werden muss, kann hier eine Formatzeichenfolge stehen. Hinweise dazu finden Sie in der Referenz im Anhang. Der dynamische Parameter hat die Nummer 0.

- ▶ `DataKeyField`

Dieses Feld bestimmt die Schlüsselspalte, die meist mit dem Primärschlüssel der unterliegenden Tabelle übereinstimmt. Dadurch kann bei der Auswahl einer Datenreihe die Zuordnung zur Datenquelle beim Löschen oder Aktualisieren bestimmt werden.

- HeaderText

Die Spaltenüberschrift wird mit diesem Attribut festgelegt.

- HeaderStyle-XXX, ItemStyle-XXX, FooterStyle-XXX

Eine ganze Palette von Stilattributen, die Kopf- oder Fußzeile bzw. die Reihen formatieren.



Es ist nicht möglich, alternierende Zeilen in der Spaltendefinition vorzunehmen. Das ist deshalb nicht zulässig, weil sich die Definitionen innerhalb einer Spalte mit denen anderer widersprechen könnten. Damit das von vornherein nicht passiert, steht nur das zentrale Attribut `<AlternatingItemStyle/>` zur Verfügung, das bereits bei `DataList` beschrieben wurde.

Das folgende Beispiel zeigt die Anwendung:

```
<asp:DataGrid Runat="server" ID="dataGrid" ⚡
    AutoGenerateColumns="False" ⚡
    Font-Name="Verdana">
  <AlternatingItemStyle BackColor="#ffffcc"/>
  <Columns>
    <asp:BoundColumn DataField="FileName" ⚡
      HeaderText="Dateiname"/>
    <asp:BoundColumn DataField="FileSize" ⚡
      HeaderText="Größe"
      HeaderStyle-BackColor="#eeeeee"/>
  </Columns>
</asp:DataGrid>
```

Listing 6.44: Einfaches `DataGrid`-Steuerelement mit zwei Spalten
(Ausschnitt aus `DataGridBoundColumns.aspx`)

Offensichtlich ist damit bereits ein beeindruckendes Ergebnis möglich.

FooterText, FooterStyle

Standardmäßig ist der Kopfbereich sichtbar und der Fußbereich unsichtbar. Wenn Sie eine Fußzeile anzeigen möchten, reicht es nicht aus, `FooterText` zu setzen. Sie müssen im Kopf des Tags die Eigenschaft `ShowFooter` explizit einschalten:

```
<asp:DataGrid ... ShowFooter="true" ... >
```



Wenn Sie die Fußzeile nicht benötigen, können Sie diese trickreich zur Übergabe spaltengebundener Daten verwenden, indem Sie die Informationen an Spalten binden und die Zeile dann unsichtbar machen. Ein Anwendungsbeispiel dafür finden Sie im Abschnitt 2.3.3, »Ein erstes Datenbankprojekt« ab Seite 114.

DataGrid

Verwendung von <BoundColumns>

Dateiname	Größe
DataBinder.aspx	666
DataGridBoundColumns.aspx	1166
DataGridSimplest.aspx	753
DataListCodeDesign.aspx	746
DataListEditor.aspx	2732
DataListEditorCommand.aspx	2477
DataListEditorSession.aspx	2828
DataListEditorViewState.aspx	2831
DataListFiles.aspx	1208
DataListFilesExtractRows.aspx	1763
DataListTemplate.aspx	811
SimpleRepeater.aspx	1019
SimpleRepeater2.aspx	1232

Abbildung 6.34: Ausgabe einer Datenliste mit *DataGrid*

Mit Vorlagen gestaltete Spalten

Mit dem Steuerelement `<asp:TemplateColumn>` erhalten Sie Zugriff auf die Gestaltung mit Vorlagen. Innerhalb des Tags stehen die von *DataList* bekannten Attribute zur Verfügung:

- ▶ `<HeaderTemplate>`
- ▶ `<ItemTemplate>`
- ▶ `<EditItemTemplate>`
- ▶ `<FooterTemplate>`

Auch hier fehlt – im Unterschied zu *DataList* – die Definition der alternativen Zeilen. Sie können diese nur über die Stile gestalterisch absetzen. Individuell kann dagegen für jede Spalte der Kopf- und Fußbereich gestaltet werden. In Fußzeilen lassen sich beispielsweise Summen platzieren. Wenn Ihre Datenquelle den direkten Abruf nicht gestattet, können Sie auch eine Methode direkt aufrufen. Das folgende Beispiel zeigt eine solche Version:

**Keine alternativen
Zeilen**

```
<asp:DataGrid Runat="server" ID="dataGrid"
    AutoGenerateColumns="False"
    Font-Name="Verdana"
    ShowFooter="True">
    <AlternatingItemStyle BackColor="#ffffcc"/>
```

```

<FooterStyle BackColor="#0000cc" ForeColor="white"/>
<Columns>
    <asp:TemplateColumn>
        <HeaderStyle BackColor="#cccccc" Font-Bold="True"/>
        <HeaderTemplate>
            Dateiname
        </HeaderTemplate>
        <ItemTemplate>
<%# Container.DataItem.FileName %>
        </ItemTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderText="Größe">
        <ItemStyle HorizontalAlign="Right"/>
        <ItemTemplate>
<%# Container.DataItem.FileSize %>
        </ItemTemplate>
        <FooterTemplate>
            <%# FileSizeSummary() %>
        </FooterTemplate>
    </asp:TemplateColumn>
</Columns>
</asp:DataGrid>

```

Listing 6.45: Ein *DataGrid* mit Vorlagen und der Fußzeile mit Spaltensumme (*DataGridTemplateColumn.aspx*)

Der Abruf der Daten erfolgt unverändert; neu ist lediglich die Methode *FileSizeSummary*:

```

Public Function FileSizeSummary() As String
    Return fc.FileSum.ToString()
End Function

```

Listing 6.46: Ausschnitt mit der Erweiterung in *DataGridTemplateColumn.aspx.vb*

Sie basiert, wie alle anderen Datenquellen in diesem Kapitel auch, auf der Klasse *FileCollection* und dort der Eigenschaft *FileSum*:

```

ReadOnly Property FileSum() As Long
    Get
        Dim i As Integer
        Dim sum As Long = 0
        For i = 0 To Me._length - 1
            sum += _files(i).FileSize
        Next
        Return sum
    End Get
End Property

```

Damit verfügt die Anzeige nun bereits über eine Spaltensumme:

DataGrid	
Verwendung von <TemplateColumns>	
Dateiname	Größe
region2treeview.xslt	665
regions.xml	5155
regions.xsd	1249
regions.xsx	212
treenodes.xml	243
universal.xslt	1403
	8927

Abbildung 6.35: Platzierung der Spaltensumme durch direkten Aufruf einer Funktion im Datenbindungs Ausdruck

Spaltenweises Sortieren von Daten

Sortiermöglichkeiten sind im `DataGrid` standardmäßig vorhanden. Das bedeutet nicht, dass beliebige Daten automatisch sortiert werden. Für die eigentliche Sortierfunktionalität müssen Sie selbst sorgen. Die Klasse `FileCollection`, die in diesem Kapitel verwendet wird, ist dafür schon vorbereitet worden. Die Dateiliste kann auf- und absteigend nach der Größe der Dateien sortiert werden.

Sortieren von Daten

Die Sortierfunktion wird im `DataGrid` durch eine Veränderung des Spaltenkopfes angezeigt. Statt der einfachen Ausgabe der Überschrift erscheint diese als Link. Wird der Link vom Benutzer angeklickt, wird ein damit verknüpft Ereignis ausgelöst. Die Vorgehensweise ist mit den anderen Ereignisfunktionen identisch. Zuerst ein Blick auf den Kopf der Zeile:

```
<asp:DataGrid ...
    AllowSorting="True"
    OnSortCommand="DoSort"
...>
```

Zwei Aktionen sind notwendig. Zuerst müssen Sie die Sortierfunktion global erlauben; dies erledigt `AllowSorting="True"`. Dann wird die Ereignisbehandlungsmethode `DoSort` angegeben. Der Na-

me ist frei wählbar. Das Attribut `OnSortCommand` verknüpft jeden Link in der Kopfzeile, der das entsprechende Attribut trägt, mit der Methode.

Nach diesem Schritt sind noch die Spalten zu bestimmen, die die Sortierlinks tragen. Prinzipiell kann dies jede Spalte sein. Es obliegt Ihnen dafür Sorge zu tragen, dass die vom Benutzer erwartete Aktion auch ausgeführt wird. Das `DataGrid`-Steuerelement prüft nicht, ob die Sortierung der fiktiven Spalte A auch tatsächlich A und nicht C sortiert. Im nächsten Beispiel werden drei Spalten definiert, zwei mit Sortierlinks und eine mit der Anzeige des Sortierwertes. Dadurch können gleich zwei verschiedene Kommandos abgesetzt werden: Auf- und Abwärtssortierung. Zuerst das `DataGrid` auf einen Blick:

```
<asp:DataGrid Runat="server" ID="dataGrid"
    AutoGenerateColumns="False" Font-Name="Verdana"
    ShowFooter="True" AllowSorting="True"
    OnSortCommand="DoSort"
    GridLines="Horizontal">
<AlternatingItemStyle BackColor="#ffffcc"/>
<FooterStyle BackColor="#0000cc" ForeColor="white"/>
<Columns>
    <asp:TemplateColumn>
        <HeaderStyle BackColor="#cccccc" Font-Bold="True"/>
        <HeaderTemplate>
            Dateiname
        </HeaderTemplate>
        <ItemTemplate>
            <%# Container.DataItem.FileName %>
        </ItemTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderText="[-]"
        SortExpression="Down">
        <HeaderStyle Width="50" HorizontalAlign="Center"/>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderText="Größe">
        <ItemStyle HorizontalAlign="Right"/>
        <ItemTemplate>
            <%# Container.DataItem.FileSize %>
        </ItemTemplate>
        <FooterTemplate>
            <%# FileSizeSummary()%>
        </FooterTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderText="[+]"
        SortExpression="Up">
```



```

        <HeaderStyle Width="50" HorizontalAlign="Center"/>
    </asp:TemplateColumn>
</Columns>
</asp:DataGrid>

```

Listing 6.47: DataGrid mit zwei Spalten, die Sortieroptionen enthalten (DataGridSort.aspx)

Ob die Spalte, die die Sortierung auslöst, Daten zur Anzeige enthält oder nicht, ist irrelevant. Die angezeigten Daten werden nicht zur Sortierung verwendet, sondern die zugrunde liegende Datenquelle. Eine Spaltenüberschrift mutiert zum Link, wenn Sie das Attribut `SortExpression` einbauen. Der Parameter wird als Eigenschaft `SortExpression` des Ereignisarguments `DataGridSortCommandEventArgs` übergeben.

```

Public Sub DoSort(ByVal sender As Object,
                  ByVal e As DataGridSortCommandEventArgs)
    fc = CType(Session("dl"), FileCollection)
    Select Case e.SortExpression
        Case "Up"
            fc.SortSizeUp()
            Exit Sub
        Case "Down"
            fc.SortSizeDown()
            Exit Sub
    End Select
    dataGrid.DataSource = fc
    dataGrid.DataBind()
End Sub

```

Listing 6.48: Definition einer Ereignisbehandlungsmethode für die Sortfunktion (Ausschnitt aus DataGridSort.aspx.vb)

Die Spalte, auf die der Benutzer geklickt hat, wird also anhand des `SortExpression`-Parameters erkannt.

Etwas trickreich ist die Gestaltung bei der gezeigten Form mit mehreren Sortierkriterien pro Spalte. Da pro Spalte nur ein Sortierlink verwendet werden kann, wurden im Beispiel drei Spalten verwendet. Die Links stehen über Spalten ohne Inhalt. Damit es optisch so wirkt, als wäre es eine Spalte, wurden die vertikalen Trennlinien unterdrückt: `GridLines="Horizontal"` im Kopf des Steuerelements sorgt dafür. Um die vertikale Linie neben der linken Spalte zu erhalten, wird dessen Vorlage `ItemStyle` mit einem Rand versehen:

Gestaltungstipps

```

<ItemStyle BorderStyle="Solid" BorderWidth="1"/>

```

DataGrid

Sortieren

Dateiname	[-]	Größe	[+]
DataListEditorSession.aspx		2828	
DataListEditor.aspx		2732	
DataListEditorCommand.aspx		2477	
DataGridSort.aspx		2044	
CollectionImplementation.aspx		1775	
DataListFilesExtractRows.aspx		1761	
DataGridTemplateColumn.aspx		1613	
DataListFiles.aspx		1323	
SimpleRepeater2.aspx		1232	
DataGridBoundColumns.aspx		1106	
SimpleRepeater.aspx		1019	
DataListTemplate.aspx		811	
DataGridSimplest.aspx		753	
DataListCodeDesign.aspx		746	
		25051	

Abbildung 6.36: DataGrid mit Sortioptionen

Ereignisse zeilenweise an die Daten binden

Es gibt ein weiteres Ereignis, das für umfangreichere Darstellungen eingesetzt werden kann. Sie können diese Methode übrigens auch mit `DataList`, `Repeater` und `HtmlSelect` einsetzen. Dabei wird immer dann ein Ereignis ausgelöst, wenn ein Datensatz aus der Datenquelle an das Steuerelement zum Aufbau der nächsten Zeile übergeben wird. Die Definition erfolgt im Kopf mit dem folgenden Attribut:

```
<asp:DataGrid ... OnItemDataBound ... >
```

Die Technik eignet sich dazu, auf bestimmte Daten zeilenweise zu reagieren. Der naive Ansatz bestünde darin, die Daten bei der Ausgabe mit Programmcode zu modifizieren. Das scheitert aber daran, dass zur Laufzeit die Daten erst gelesen und dann an die Vorlagen übergeben werden. Ereignisse sind der richtige und bessere Weg – sie entsprechen dem Stil objektorientierter Programmierung. Das hier verwendete Ereignis wird ausgelöst, wenn die Daten der Datenquelle entnommen wurden und mit der Vorlage verknüpft sind. Unmittelbar danach ist die Zeile nicht mehr zugänglich.

In der Dateiliste könnte man beispielsweise auf die Größe reagieren und alle Dateien markieren, die kleiner als 1 KByte sind. Eine häufige Anwendung ist die Anzeige von fortlaufenden Zahlen am Anfang der Spalte – vor allem dann, wenn Ihre Datenquelle diese Information nicht bietet. Die folgende Abbildung zeigt die Vorgehensweise.

DataGrid

Umgang mit Datenbindungsereignissen

Dateiname	Größe
0. DataGridBoundColumns.aspx	1106
1. DataGridItemBind.aspx	1925
2. DataGridSimplest.aspx	753
3. DataGridSort.aspx	2044
4. DataGridTemplateColumn.aspx	1613
5. DataListCodeDesign.aspx	746
6. DataListEditor.aspx	2732
7. DataListEditorCommand.aspx	2477
8. DataListEditorSession.aspx	2828
9. DataListEditorViewState.aspx	2831
10. DataListFiles.aspx	1323
11. DataListFilesExtractRows.aspx	1761
12. DataListTemplate.aspx	811
13. SimpleRepeater.aspx	1019
14. SimpleRepeater2.aspx	1232
	26976

Abbildung 6.37: Dynamisches Erzeugen von Reihendaten zur Laufzeit

Die Definition der Spalte ist sehr einfach. Ein `Label`-Steuerelement soll später die Daten aufnehmen und für die Anzeige sorgen:

```
<asp:TemplateColumn>
  <ItemStyle HorizontalAlign="Right" Font-Bold="True"/>
  <ItemTemplate>
    <asp:Label Runat="server" ID="number"/>.
  </ItemTemplate>
</asp:TemplateColumn>
```

Listing 6.49: Spalte, die zur Laufzeit mit Daten außerhalb der Datenquelle gefüllt wird (Ausschnitt aus `DataGridItemBind.aspx`)

Im Kopf des `DataGrid`-Steuerelementes muss noch die Verknüpfung des Ereignisses mit einer Ereignisbehandlungsmethode erfolgen:

```
<asp:DataGrid ... OnItemDataBound="ModifyRow" ... >
```

Bleibt als Letztes die Definition der Methode:

```
Public Sub ModifyRow(ByVal sender As Object, 
                    ByVal e As DataGridItemEventArgs)
    If e.Item.ItemType = ListItemType.Item 
    Or e.Item.ItemType = ListItemType.AlternatingItem Then
        Dim number As Label = CType(e.Item.FindControl("number"),
Label)
        number.Text = e.Item.ItemIndex.ToString()
    End If
End Sub
```

*Listing 6.50: Zeilenweise dynamische Modifikation
(Ausschnitt aus DataGridItemBind.aspx.vb)*

**Wie es
funktioniert**

Hier wird zuerst überprüft, welchen Typ die gerade verarbeitete Reihe hat. Der Zugriff erfolgt über die Eigenschaft `ItemType` des Objekts `Item`, das die Zeile repräsentiert. Die Aufzählung `ListItemType` kann zum Vergleich verwendet werden. Auf diesem Wege sind natürlich auch die Kopf- und Fußzeilen zugänglich, und zwar genau in der Reihenfolge, wie sie auf der Seite erscheinen. Das `Label`-Steuerelement wird über `FindControl` ermittelt. Sie können dann an dieser Stelle beliebige Daten zuweisen. Für fortlaufende Nummern ist `ItemIndex` ideal.

6.7 Kontroll-Steuerelemente (Validation Controls)

In den vorangegangenen Abschnitten wurden Formularelemente, bestehend aus HTML Server- oder Web Server-Steuerelementen, bereits intensiv genutzt. Es liegt in der Natur der Sache, dass der von Benutzern erfasste Inhalt nicht immer kritiklos angenommen werden kann. Angefangen von einfachen Tippfehlern bis hin zu mutwilligen Falschangaben wird ein Programm viele falsche Daten verkraften müssen. Natürlich ist es möglich, die gesendeten Daten zu überprüfen und dann entsprechend zu reagieren. In der Praxis enthalten aber Formulare viele Felder, die teilweise miteinander in Beziehung stehen. Denken Sie an eine vergleichsweise einfache Kennwortänderung. Hier müssen Sie zum einen die Bedingungen des Kennworts selbst kontrollieren, beispielsweise eine Mindestanzahl Zeichen. Außerdem müssen das Kennwortfeld und ein Kontrollfeld miteinander verglichen werden. Das kann in

einer größeren Applikation einigen Programmieraufwand verursachen.

6.7.1 Prinzipien der Feldkontrolle

In ASP.NET werden zur Überprüfung von Feldinhalten so genannte Kontroll-Steuerelemente eingesetzt. Diese stammen aus der Gruppe der Web Server-Steuerelemente und sind vergleichsweise einfach einsetzbar. Einige Grundlagen sollten Sie vorher kennen lernen, denn dies erleichtert die Entscheidung für die eine oder andere Option bei der Programmierung.

Prinzipiell werden in der Formularprogrammierung zwei Prüfungen unterschieden:

Prüfung: Client oder Server?

- ▶ Clientseitige Feldprüfung mit JavaScript
- ▶ Serverseitige Feldprüfung

Die clientseitige Prüfung nutzt JavaScript und verhindert üblicherweise, dass nicht korrekt ausgefüllte Formulare gar nicht erst abgesendet werden. Dies verringert die Netzlast auf dem Server und beschleunigt die Abarbeitung, denn das Senden an den Server und die Verarbeitung der Antwort entfällt. Nachteilig ist, dass die Präsentation der Fehler im Formular mit JavaScript und DHTML aufwändig zu programmieren ist. Außerdem funktioniert die Applikation möglicherweise überhaupt nicht mehr, wenn der Benutzer aus Sicherheitsgründen das Scripting im Browser grundsätzlich unterbindet.

Clientseitig

Bei der serverseitigen Prüfung werden die Formulardaten empfangen, ausgewertet und dann wird eine entsprechende Reaktion an den Browser gesendet, beispielsweise ein neues Formular mit detaillierten Ausfüllhinweisen.

Serverseitig

Auswahl der Feldkontrolle

Normalerweise würden sich an dieser Stelle endlose Debatten um die Vorzüge von JavaScript und DHTML anschließen, bzw. um handfeste Gründe, warum der Einsatz unbedingt vermieden werden muss. ASP.NET nimmt Ihnen die Entscheidung eigentlich ab – wenngleich Sie dennoch die letzte Kontrolle darüber behalten. Prinzipiell realisieren die Kontroll-Steuerelemente immer serverseitige Funktionen. Wenn ein Browser jedoch DHTML unter-

stützt, werden zusätzlich clientseitige Prüfungen eingebaut. Damit wird sichergestellt, dass Ihre Applikation in jeder Umgebung und unter allen Bedingungen funktioniert. Wenn es möglich ist, Bandbreite einzusparen, wird dies auch getan. Allerdings kann die Verwendung von DHTML explizit unterdrückt werden. Dazu wird einfach die Direktive @Page um ein entsprechendes Attribut erweitert:

```
<% @Page ClientTarget="Downlevel" %>
```

ClientTarget

Die serverseitige Kontrolle kann dagegen nicht unterdrückt werden. Es mag in Anbetracht einer funktionierenden clientseitigen Überprüfung der Felder unnötig erscheinen. Allerdings gehört für einen halbwegs begabten Hacker nicht viel dazu, das Formular auszulesen, den JavaScript-Code zu analysieren, zu verändern und dann das Formular mit fehlerhaften Daten an Ihren Server zu senden. Fatal, wenn dann keine zweite Kontrollmauer existiert, die für Hacker definitiv nicht erreichbar ist.

Ablauf der Prüfung

Im Objekt `Page` gibt es eine Eigenschaft `IsValid`, die anzeigt, ob Fehler im Formular auftraten (`False`) oder nicht (`True`). Dabei spielt es keine Rolle, ob eine oder mehrere Prüfungen nicht bestanden wurden, um das Formular ungültig werden zu lassen. Wie Sie auf den Ausgang reagieren, können Sie selbst entscheiden, an der Bereitstellung der Formulardaten in der `Form`-Kollektion oder als Eigenschaft der Server-Steuerelemente ändert sich nichts.

6.7.2 Typische Kontroll-Steuerelemente

RequiredField-Validator

Die häufigste Prüfung besteht in der Feststellung, ob ein Feld überhaupt ausgefüllt wurde. Dazu wird das Steuerelement `RequiredFieldValidator` eingesetzt. Das folgende Listing zeigt, wie es verwendet wird. Realisiert wird der erste Teil eines Eingabeformulars für den Shop, wo Kunden ihre Adressen eingeben können. Verständlich ist sicher, dass bestimmte Angaben, wie Name und Straße, unbedingt erforderlich sind.

```
<% @Page Language="vb" debug="true" %>
<html>
  <head>
    <title>Validation Control - 1</title>
```

```

</head>
<body>
  Bitte geben Sie die Lieferadresse an:<br/>
  <form runat="server">
    <table border="1">
      <tr>
        <td>Name</td>
        <td>
          <asp:textbox runat="server"
            id="f_name" />
          </td>
        <td>
          <asp:requiredfieldvalidator runat="server"
            id="v_name"
            controltovalidate="f_name"
            display="dynamic">
            Geben Sie einen Namen ein!
          </asp:requiredfieldvalidator>
        </td>
      </tr>
      <tr>
        <td>Strasse</td>
        <td>
          <asp:textbox runat="server" id="f_street" />
        </td>
        <td>
          <asp:requiredfieldvalidator runat="server"
            id="v_street"
            controltovalidate="f_street"
            display="dynamic">
            Geben Sie eine Stra&szlig;e an!
          </asp:requiredfieldvalidator>
        </td>
      </tr>
    </table>
    <asp:button runat="server" id="send" text="Bestellen"/>
  </form>
</body>
</html>

```

Listing 6.51: Formular mit zwei kontrollierten Eingabefeldern
(ValidationControlRequired.aspx)

Dieses Formular verfügt – vor allem in Anbetracht des fehlenden hinterlegten Codes – bereits über beträchtliche Funktionen. Beide Felder werden auf den Zustand »Feld ausgefüllt« hin überprüft. Bleibt ein Feld leer, wird rechts daneben eine Fehlermeldung in roter Schrift eingeblendet. Wenn Sie Ihren Browser genau beobachten, werden Sie außerdem feststellen, dass das Formular nicht

**Wie es
funktioniert**

gesendet wurde. Das erstaunliche Ergebnis basiert auf einer clientseitigen Prüfung. Die gesamte Funktionalität steckt in folgendem Kontroll-Steuerelement:

```
<asp:requiredfieldvalidator runat="server"
    id="v_street"
    controltovalidate="f_street" display="dynamic">
Geben Sie eine Stra&szlig;e an!</asp:requiredfieldvalidator>
```

Der Name `asp:requiredfieldvalidator` bestimmt die Art der Prüfung: Das Feld muss mindestens ein Zeichen enthalten. Dann wird mit dem Attribut `controltovalidate` festgelegt, welches Feld überhaupt überwacht werden soll. Damit sind die unbedingt notwendigen Angaben bereits erfolgt – abgesehen von den obligatorischen Attributen `id` und `runat`.

Steuerung der Anzeige

Das Attribut `display` ist eine nähere Betrachtung wert. Die Ausgabe von Fehlermeldungen inmitten eines Formulars ist bei sorgfältig gestalteten Seiten keine triviale Aufgabe. Schließlich soll die zusätzlich erscheinende Meldung das Layout nicht zerstören. Andererseits soll die am Anfang und hoffentlich auch später fehlende Fehlernachricht nicht weiße Flecken hinterlassen. Kreative Webdesigner werden sicher einen vernünftigen Weg finden; deren Vorgabe umzusetzen, ist Aufgabe des Attributs `display`:

Wert des Attributs <code>display</code>	Verhalten der Fehlernachricht
Dynamic	Der Text erscheint dynamisch, wenn erforderlich. Fehlt die Nachricht, wird kein Platz in Anspruch genommen.
Static	Der Text erscheint dynamisch, wenn erforderlich. Der dafür erforderliche Platz wird immer in Anspruch genommen.
None	Der Text erscheint niemals.

Tabelle 6.10: Wirkungsweise des Attributes `display`

Die Abbildung 6.38 zeigt, wie sich das Kontroll-Steuerelement mit dem Attribut `display="Dynamic"` verhält.

Abbildung 6.38: Dynamische Anzeige einer Fehlermeldung

Die Tabelle wird durch das in der rechten Spalte platzierte Kontroll-Steurelement dynamisch erweitert. Das kann, wenn die Tabelle ein wichtiges Gestaltungselement der Seite ist, den gesamten Aufbau zerstören. Würde als Attribut `display="static"` verwendet werden, sähe die Tabelle folgendermaßen aus:

Abbildung 6.39: Tabelle mit reserviertem Platz für Meldungen

Interessant ist natürlich, was ASP.NET eigentlich an den Browser gesendet hat. Ein Blick in den Quelltext lohnt sich auch hier.

Wie die clientseitige Prüfung funktioniert

Die clientseitige Prüfung basiert auf JavaScript. Dazu stellt ASP.NET Skriptbibliotheken zur Verfügung, die im IIS unter dem virtuellen Verzeichnis `/aspnet_client` zu finden sind. Das Absenden des Formulars wird durch einen entsprechenden Eintrag kontrolliert:

```
<form ... action="vc_required.aspx" ⚡
      onsubmit="ValidatorOnSubmit();">
```

JavaScript-Bibliothek

Diese Funktion führt nach einer Versionsprüfung in die Bibliothek *WebUIValidation.js*. Von dort werden die Fehlermeldungen auf der Seite kontrolliert. Diese werden in `<span>`-Tags platziert, die dann über DHTML sichtbar oder unsichtbar gemacht werden können:

```
<span id="v_name" controltovalidate="f_name" ⚡
      evaluationfunction="RequiredFieldValidatorEvaluateIsValid"
      initialvalue="" style="color:Red;visibility:hidden;">
Geben Sie einen Namen ein!
</span>
```

Mit `visibility:hidden` wird die Anzeige am Anfang unterdrückt, der vom Text in Anspruch genommene Platz ist jedoch reserviert. Es wurde also `display="static"` verwendet. Wenn dagegen `display="dynamic"` verwendet wird, entsteht folgendes Tag:

```
<span id="v_name" controltovalidate="f_name"
      display="Dynamic"
      evaluationfunction="RequiredFieldValidatorEvaluateIsValid"
      initialvalue="" style="color:Red;display:none;">
Geben Sie einen Namen ein!
</span>
```

Mit dem CSS-Attribut `display:none` wird die Anzeige also völlig unterdrückt.

Diese Aussagen treffen für alle Kontroll-Steuerelemente zu, denn die Art der Fehlerausgabe ist für alle identisch.

Serverseitige Prüfungen verwenden

Warum Sie serverseitig prüfen sollten

Die Einzelprüfung ist zwar praktikabel und zuverlässig, eine serverseitige Prüfung kann jedoch noch mehr bieten. Im Sinne einer guten Benutzerführung wäre es empfehlenswert, dem Benutzer an zentraler Stelle gut sichtbar einen allgemeinen Hinweis auf das Problem zu geben. Das folgende Beispiel zeigt ein solches Programm. Hier wird ausschließlich serverseitig geprüft und eine zusätzliche Meldung bei Bedarf angezeigt:

```
<body>
  Bitte geben Sie die Lieferadresse an:<br/>
  <form runat="server">
```

```
<div style="background-color:red; color:white"
    id="errorpanel" runat="server">
    Es traten Fehler beim Ausfüllen
    des Formulars auf.
</div>
```

// ... Rest des Listings entspricht dem vorherigen

Listing 6.52: Serverseitige Prüfung mit zentraler Fehlermeldung
(ValidationControlsRequiredIsValid.aspx)

In der Code-Datei wird dann der Fehlerbereich ein- oder ausgeschaltet:

```
Public Sub checkform(ByVal sender As Object, ByVal e As
EventArgs)
    If Page.IsValid Then
        errorpanel.Visible = False
    Else
        errorpanel.Visible = True
    End If
End Sub
```

Listing 6.53: Reaktion auf die interne Prüfung (Ausschnitt aus
ValidationControlsRequiredIsValid.aspx.vb)

Zuerst wird hier die clientseitige Prüfung mit ClientTarget="Downlevel" unterdrückt. Dann wird ein Klickereignis definiert, das das Absenden des Formulars aufnimmt. Hier wird die Eigenschaft IsValid geprüft und ein HTML Server-Steuerelement, das als <div>-Tag vorliegt, wird entsprechend zur Anzeige gebracht, wenn Fehler auftraten. Beachten Sie, dass Sie diesen Teil nicht in Page_Load schreiben können, denn zum Zeitpunkt, wo die Seite geladen wurde, hat die Prüfung noch nicht stattgefunden. Die Kontroll-Steuerelemente müssen in der Lage sein, auf die Seitengestaltung Rücksicht zu nehmen bzw. alle Elemente der Seite kennen. Deshalb wird die Seite erst soweit abgearbeitet, dass Page_Load ausgeführt werden kann und dann erfolgt die Prüfung.

Wie es funktioniert

6.7.3 Weitere wichtige Kontroll-Steuerelemente

Dieser Abschnitt stellt anhand einfacher Beispiele die wichtigsten Kontroll-Steuerelemente und deren Anwendung vor.

Das Vergleichs-Steuerelement (Compare Validator)

Das Vergleichs-Steuerelement (Compare Validator) vergleicht ein Feld mit einem konstanten Wert, einem Datentyp oder einem anderen Feld. Da es möglich ist, mehr als ein Kontroll-Steuerelement an ein Feld zu binden, können damit auch Bereiche eingeschränkt werden. Die Kombination mit dem Steuerelement `RequiredFieldValidator` ist erforderlich, denn das Kriterium »Vorhandensein eines Wertes« erzwingt es nicht. Das folgende Beispiel erweitert die Adressabfrage und enthält zusätzlich die Festlegung eines Kennwortes.

```
<tr>
  <td>Kennwort</td>
  <td>
    <asp:textbox runat="server" id="f_password1"
      textmode="password"/><br/>
    <asp:textbox runat="server" id="f_password2"
      textmode="password"/>

  </td>
  <td>
    <asp:comparevalidator id="Compare1" type="string"
      runat="server"
      controltovalidate="f_password1"
      controltocompare="f_password2">
      Die Kennwortfelder stimmen nicht überein
    </asp:comparevalidator>
  </td>
</tr>
```

Listing 6.54: Mehrere Prüfungen mit dem `CompareValidator` Steuerelement (Ausschnitt aus `ValidationControlsCompare.aspx`)

Das Steuerelement `CompareValidator` verfügt über eine ganze Reihe von Attributen. Dadurch ist der Einsatz sehr oft möglich. Neben den obligatorischen Attributen `runat` und `id` ist der Typ der Daten anzugeben, die das geprüfte Feld enthalten soll. Dazu dient das Attribut `type`. Es kann folgende Parameter enthalten:

Parameter für type	Beschreibung
String	Eine Zeichenkette wird erwartet
Integer	Ein ganzzahliger Wert wird akzeptiert
Double	Das Feld muss eine Zahl enthalten

Tabelle 6.11: Datentypprüfung für `CompareValidator`

Parameter für type	Beschreibung
Date	Ein Datumswert entsprechend dem DateTime wird erwartet
Currency	Eine Währungsangabe – Zahl mit Währungs-symbol – wird erwartet

Tabelle 6.11: Datentypprüfung für CompareValidator (Forts.)

Soll lediglich der Datentyp eines Feldes überwacht werden, ist nur das Attribut operator mit dem Parameter DataTypeCheck erforderlich. Folgende Kombination nutzt dies:

```
... type="Integer" operator="DataTypeCheck" ...
```

Sollen dagegen zwei Felder verglichen werden, sind zwei andere Attribute beteiligt: controltovalidate und controltocompare:

Zwei Felder
vergleichen

```
... controltovalidate="f_password1"
    controltocompare="f_password2" ...
```

Die Kombination mit type ist möglich, dann wird neben der Gleichheit der Felder auch die Übereinstimmung des Datentyps geprüft. Das Attribut operator ist dann optional. Für alle anderen Fälle, in denen ein Vergleich erfolgen soll, können Sie operator mit folgenden Parametern ausstatten:

Parameter für operator	Beschreibung
DataTypeCheck	Datentyp
Equal	Gleichheit (=)
GreaterThan	Größer als (>)
LessThan	Kleiner als (<)
GreaterThanEqual	Größer als oder gleich (>=)
LessThanEqual	Kleiner als oder gleich (<=)
NotEqual	Ungleich (<>)

Tabelle 6.12: Vergleichsoperatoren für CompareValidator

Der Vergleich kann nicht nur zwischen Feldern, sondern auch mit einem konstanten Wert erfolgen. Dazu wird statt controltocompare das Attribut valuetocompare verwendet. Für den Vergleich von Zahlen ist das Attribut type erforderlich; operator mit einem der genannten Parameter sowieso:

```

<asp:textbox runat="server" id="f_age" />
<asp:comparevalidator id="v_age" runat="server"
    controltovalidate="f_age"
    valuetocompare="18"
    type="Integer"
    operator="GreaterThan"
    ErrorMessage="Das Alter muss
    mindestens 18 sein"/>

```

Der Operator ist so zu gestalten, dass er die Bedingung für eine gültige Prüfung ergibt. Im Beispiel wird die Bedingung »f_age > 18« geprüft. Ist dieser Vergleich nicht erfüllt, wird die Meldung ausgegeben.

Ausgabe der Fehlermeldung

Die Ausgabe der Fehlermeldung kann auf zwei Wegen erfolgen: Entweder nutzen Sie das Attribut `ErrorMessage` und schließen das Steuerelement sofort ab:

```
<asp:comparevalidator ErrorMessage=«Feld falsch» ... />
```

Oder Sie bauen ein Container-Tag, das die Meldung enthält:

```
<asp:comparevalidator ... >Feld falsch</asp:comparevalidator>
```

Das Bereichskontroll-Steuerelement (RangeValidator Control)

Manchmal reicht der Vergleich von Feldern mit Festwerten nicht aus. Der `RangeValidator` prüft, ob Werte in einem bestimmten Bereich liegen. So könnte die Prüfung der Postleitzahlen im letzten Beispiel verbessert werden, indem statt der Prüfung des Datentyps eine Einschränkung auf den Bereich zwischen 1.000 und 99.999 erfolgt. Der `RangeValidator` arbeitet ähnlich dem `CompareValidator`, verwendet aber zur Kontrolle zwei spezielle Attribute, die den Wertebereich eingrenzen: `MinimumValue` und `MaximumValue`.

```

<tr>
  <td>PLZ</td>
  <td>
    <asp:textbox runat="server" id="f_zipcode" width="5"/>
  </td>
  <td>
    <asp:rangevalidator runat="server" id="v_zipcode"
      type="Integer"
      minimumvalue="1000"
      maximumvalue="99999"
      controltovalidate="f_zipcode"

```

```

                                display="dynamic">
        Postleitzahl im falschen Wertebereich
    </asp:rangevalidator>
    <asp:requiredfieldvalidator runat="server"
                                id="vr_zipcode"
                                controltovalidate="f_zipcode"
                                display="dynamic">
        Postleitzahl nicht angegeben
    </asp:requiredfieldvalidator>
</td>
</tr>

```

Listing 6.55: Kontrolle eines Wertebereiches (Ausschnitt aus ValidationControlsVcCompare.aspx)

Das Regulärer Ausdruck-Steuererelement (Regular Expression Validator)

In allen Fällen, in denen die Prüfung von Eingabewerten nicht mittels Übereinstimmung, Datentyp oder Bereich erfolgen kann, sind reguläre Ausdrücke ein ideales Mittel. Reguläre Ausdrücke bestehen aus einem definierten Satz von Steuerzeichen, mit denen Suchmuster gebildet werden können. Eine Einführung in die Technik und Anwendung finden Sie im Abschnitt 4.7, »Reguläre Ausdrücke« ab Seite 267. Dort werden spezielle Klassen des Frameworks verwendet, um Text mit solchen Ausdrücken zu durchsuchen. Hier wird dagegen der `RegularExpressionValidator` verwendet, um Werte zu prüfen.

Häufig wird ein Eingabefeld für die E-Mail-Adresse benötigt. Der dazu erforderliche reguläre Ausdruck sieht folgendermaßen aus:

```

^
[_a-zA-Z0-9-]+
(\.
    [_a-zA-Z0-9-]+
)*
@
(
    [_a-zA-Z0-9-]+
    \.
)+
(
    [_a-zA-Z]{2,3}
)
$

```

Listing 6.56: Regulärer Ausdruck zum Prüfen einer E-Mail-Adresse

Das kryptische Gebilde wurde hier auseinandergezogen, um die einzelnen Elemente kommentieren zu können. In der Praxis schreibt man die Zeichen in eine Zeile. Konsultieren Sie Abschnitt 4.7, »Reguläre Ausdrücke« ab Seite 267, um herauszufinden, wie der Ausdruck funktioniert und wie Sie selbst Suchmuster entwerfen können.

Die Anwendung ist einfacher. Das Steuerelement verfügt über ein neues Attribut, `ValidationExpression`. Die Prüfung der E-Mail-Adresse sieht dann folgendermaßen aus:

```
<tr>
  <td>E-Mail</td>
  <td>
    <asp:textbox runat="server" id="f_email" />
  </td>
  <td>
    <asp:regularexpressionvalidator runat="server"
      id="v_email"
      validationexpression =
        "[_a-zA-Z0-9-]+\([[_a-zA-Z0-9-]+\)*@([a-zA-Z0-9-]
        +\.)+([a-zA-Z]{2,3})$"
      controltovalidate="f_email" display="dynamic">
      Diese E-Mail-Adresse ist nicht gültig
    </asp:regularexpressionvalidator>
  </td>
</tr>
```

Listing 6.57: Prüfung einer E-Mail-Adresse (Ausschnitt aus `ValidationControls-Compare.aspx`, der Ausdruck muss in einer Zeile geschrieben werden)

Grenzen regulärer Ausdrücke

Reguläre Ausdrücke können Suchmuster aufbauen. Es gibt zwar prinzipiell die Möglichkeit, Abhängigkeiten zwischen Teilen des Ausdrucks festzulegen, die Werte sind aber statischer Natur. Berechnungen lassen sich nicht anstellen. Ein typisches Beispiel sind ISBN-Nummer, Datumswerte oder Kreditkartennummern. In allen drei Fällen sind die Abhängigkeiten komplizierter. Das Muster einer ISBN-Nummer ist statisch: L-VVV-NNNNN-P. Dabei steht L für das Land, beispielsweise 3 für Deutschland, VVV für den Verlag (dieser Teil kann auch vier- oder fünfstellig sein) und NNNNN ist die eigentliche Buchnummer, wobei die Anzahl so gewählt wird, dass die gesamte Zeichenfolge ohne Bindestriche immer 10 Zeichen umfasst. Bis dahin wäre ein regulärer Ausdruck in der Lage, eine Prüfung vorzunehmen. P ist aber eine Prüfziffer, die auf Basis der anderen Zahlen berechnet werden

muss. Dies kann nicht mit einem Suchmuster erfolgen. Sie müssen dazu eine eigene Funktion schreiben, die die Prüfwert selbst berechnet und vergleicht.

6.7.4 Selbstdefinierte Kontrollelemente (Custom Validation Controls)

Natürlich ist es ohne weiteres möglich, die empfangenen Daten auf dem Server aus der `Form`-Kollektion oder der `Text`-Eigenschaft zu entnehmen und eine Prüffunktion zu schreiben. Im Sinne eines einheitlichen, übersichtlichen und geradlinigen Programmierkonzepts ist aber der Einbau in ein Kontroll-Steuerelement zu empfehlen. Dafür gibt es den `CustomValidator`. Auch dieses Steuerelement ist mit den bisher vorgestellten vergleichbar. Ein zusätzliches Attribut `onservervalidate` dient der Angabe einer Methode, die sie selbst definieren und die die `IsValid`-Eigenschaft setzt.

Das folgende Beispiel zeigt die Prüfung von ISBN-Nummern. Zuerst das Listing des HTML-Codes. Die Prüfung ist in eine hinterlegte Code-Datei ausgelagert, die danach vorgestellt wird:

**Serverseitige
Prüfungen**

```
<html>
  <head>
    <title>Validation Control - 3</title>
  </head>
  <body>
    Nach welcher ISBN möchten Sie suchen?<br/>
    <form runat="server">
      <asp:textbox runat="server" id="f_isbn" />
      <asp:button runat="server" id="send" text="Prüfen"/>
      <br/>
      <asp:customvalidator runat="server" id="v_isbn"
                           onservervalidate="checkisbn"
                           display="dynamic">
        ISBN-Nummer ist falsch
      </asp:customvalidator>
      (Geprüfte Nummer:
      <asp:label id="l_isbn" runat="server"/>)
    </form>
  </body>
</html>
```

Listing 6.58: HTML-Teil mit der Definition des benutzerdefinierten Kontroll-Steuerelements (ValidationControlCompareJS.aspx)

Definiert wird hier neben dem Eingabefeld (`f_isbn`) und der Sendeschaltfläche (`send`) auch ein `Label`, das die gewählte Nummer oder eine Fehlermeldung anzeigt. Alle Berechnungen werden in der Klasse *Checker* definiert:

```
Public Class Checker
    Inherits System.Web.UI.Page
    Protected v_isbn As System.Web.UI.WebControls.CustomValidator
    Protected f_isbn As System.Web.UI.WebControls.TextBox
    Protected send As System.Web.UI.WebControls.Button
    Protected l_isbn As System.Web.UI.WebControls.Label

    Private Function substr(ByVal isbn As String, ↵
                                ByVal pos1 As Integer) As Integer
        Dim i As Integer ↵
            = Convert.ToInt32(isbn.Substring(pos1, 1))
        Return i
    End Function

    Public Sub checkisbn(ByVal sender As Object, ↵
                                ByVal e As ServerValidateEventArgs)
        Dim isbn As String = f_isbn.Text
        Dim isbn10 As String = isbn.Replace("-", "")
        l_isbn.Text = isbn10
        If isbn10.Length <> 10 Then
            l_isbn.Text = "Anzahl Ziffern stimmt nicht = " ↵
                + isbn.Length.ToString()
            e.IsValid = False
        Else
            Dim checkdigit As Int32 = 11 ↵
                - ((10 * substr(isbn10, 0) + 9 * ↵
                    substr(isbn10, 1) + 8 * ↵
                    substr(isbn10, 2) + 7 * ↵
                    substr(isbn10, 3) + 6 * ↵
                    substr(isbn10, 4) + 5 * ↵
                    substr(isbn10, 5) + 4 * ↵
                    substr(isbn10, 6) + 3 * ↵
                    substr(isbn10, 7) + 2 * ↵
                    substr(isbn10, 8)) Mod 11)
            If checkdigit = 11 Then
                checkdigit = 0
            End If
            Dim comparedigit As Int32
            If isbn10.Substring(9, 1) = "X" Then
                comparedigit = 10
            Else
                comparedigit ↵
                    = Convert.ToInt32(isbn10.Substring(9, 1))
            End If
        End If
    End Sub
End Class
```

```

        e.IsValid = (checkdigit = comparedigit)
        l_isbn.Text += " Kontroll-Prüfsumme -> " &
            + comparedigit.ToString()
    End If
End Sub
End Class

```

Listing 6.59: Klasse zur Prüfung von ISBN-Nummern
(ValidationControlCompareJS.aspx.vb)

Die Datei beginnt mit dem Zugriff auf die benötigten Namensräume. Die verwendeten Zeichenketten-Methoden aus der Klasse `System.String` sind im Basisnamensraum `System` definiert. Die Klasse `checker` muss außerdem von `Page` erben und natürlich öffentlich zugänglich sein:

Wie es funktioniert

```
Public Class Checker Inherits System.Web.UI.Page
```

Dann werden die Web Server- bzw. Kontroll-Steuerelemente der Seite bekannt gemacht:

```

Protected v_isbn As CustomValidator
Protected f_isbn As TextBox
Protected l_isbn As Label

```

Im Programm wird sehr häufig eine Zeichenkettenoperation eingesetzt, bei der ein Teil einer Zeichenkette als Zahlenwert benötigt wird. Deshalb wurde dafür eine private Methode *substr* definiert. Die eigentliche Arbeit erledigt die Methode *checkisbn*:

```

Public Sub checkisbn(ByVal sender As Object,
                    ByVal e As ServerValidateEventArgs)

```

Das Argument `e` erlaubt den Zugriff auf das Kontroll-Steuerelement selbst, das beim Aufruf als Objekt übergeben wird.

Zuerst erfolgt der Zugriff auf das Eingabefeld über die Eigenschaft `Text`:

```
Dim isbn As String = f_isbn.Text
```

Dann werden möglicherweise mit eingegebene Bindestriche entfernt, die ISBN-Nummern nur an den sinntrennenden Stellen auflösen, sonst aber für die Berechnung keine Bedeutung haben. Die Zeichenkettenmethode `Replace` wird hier verwendet:

```
Dim isbn10 As String = isbn.Replace("-", "")
```

Dann wird dem Label der Text der ISBN zugewiesen:

```
l_isbn.Text = isbn10
```

Im nächsten Schritt wird die Länge geprüft; ISBN-Nummern haben immer 10 Ziffern. Die tatsächliche Länge wird mit der Zeichenkettenmethode `Length` ermittelt:

```
If isbn10.Length <> 10 Then
```

Stimmt die Länge nicht, wird eine entsprechende Information an das Label-Steuerelement ausgegeben und der Status des Kontroll-Steuerelements so gesetzt, dass die Fehlermeldung angezeigt wird. Dazu wird auf die Eigenschaft `IsValid` zugegriffen:

```
l_isbn.Text = "Anzahl Ziffern stimmt nicht = " &
    + isbn.Length.ToString()
e.IsValid = False
```

Validierung auf Element- oder Seitenebene

Beachten Sie hier, dass Sie nicht nur `IsValid` schreiben, denn dann wird `Page.IsValid` verwendet. Das ist zwar nicht prinzipiell falsch, denn mit der Ungültigkeit eines Kontroll-Steuerelements wird auch der Prüfzustand der gesamten Seite ungültig, aber im Beispiel wird `Page.IsValid` nicht ausgewertet. Um die Fehlermeldung zu provozieren, muss das Objekt `e` angesprochen werden.

Sind alle Prüfungen erfolgt, wird die Berechnung durchgeführt. Die Formel gibt Prüfziffern zwischen 0 und 11 zurück. In der ISBN wird die 11 als »0« und die 10 als »X« dargestellt. Die letzte Stelle der ISBN-Nummer muss zum Vergleich entsprechend umgewandelt werden:

```
Dim comparedigit As Int32
If isbn10.Substring(9, 1) = "X" Then
    comparedigit = 10
Else
    comparedigit = Convert.ToInt32(isbn10.Substring(9, 1))
End If
```

Dann folgt der Vergleich. Das Ergebnis wird gleich für `IsValid` verwendet:

```
e.IsValid = (checkdigit = comparedigit)
```

Clientseitige Prüfungen

Es ist durchaus möglich, auch clientseitige Prüfungen zu programmieren. Dazu müssen Sie die entsprechende Funktion in JavaScript schreiben. Als Attribut für das Kontroll-Steuerelement wird `clientvalidationfunction` verwendet. Werden beide Definitio-

nen verwendet, entscheidet ASP.NET anhand der Möglichkeiten des Browsers, welche eingesetzt wird. Das folgende Beispiel zeigt, wie zusätzlich zur bereits gezeigten serverseitigen Kontrolle die Definition der Prüfziffernberechnung in JavaScript erfolgen kann:

```
<html>
  <head>
    <title>Validation Control - 3</title>
    <script language="JavaScript">
      function checkisbn(sender, e)
      {
        isbn = document.checkform.f_isbn.value;
        isbn10 = isbn.replace(/-/g, "");
        if (isbn10.length != 10)
        {
          alert("Anzahl Ziffern falsch: " +
            + isbn10.length + " (" + isbn10 + ")");
          e.IsValid = false;
        } else {
          checkdigit = 11-((10*isbn10.substr(0,1) + 9 *
            isbn10.substr(1,1) + 8 *
            isbn10.substr(2,1) + 7 *
            isbn10.substr(3,1) + 6 *
            isbn10.substr(4,1) + 5 *
            isbn10.substr(5,1) + 4 *
            isbn10.substr(6,1) + 3 *
            isbn10.substr(7,1) + 2 *
            isbn10.substr(8,1) ) % 11);
          checkdigit = (checkdigit == 11) ? 0 : checkdigit;
          comparedigit = (isbn10.substr(9,1) == "X") ? 10 :
            isbn10.substr(9,1);
          e.IsValid = (checkdigit == comparedigit);
        }
      }
    </script>
  </head>
  <body>
    Nach welcher ISBN möchten Sie suchen?<br/>
    <form runat="server" id="checkform">
      <asp:textbox runat="server" id="f_isbn" />
      <asp:button runat="server" id="send" text="Prüfen"/>
      <br/>
      <asp:customvalidator runat="server" id="v_isbn"
        onservvalidate="checkisbn"
        clientvalidationfunction="checkisbn"
        display="dynamic">
        ISBN-Nummer ist falsch
      </asp:customvalidator>
      (Geprüfte Nummer:
```

```

        <asp:label id="l_isbn" runat="server"/>)
    </form>
</body>
</html>

```

*Listing 6.60: Clientseitige Realisierung einer Prüffunktion
(ValidationControlsCustomJS.aspx)*

Wie es funktioniert

Die Funktion *checkisbn* selbst unterscheidet sich nur wenig von der bereits in VB.NET gezeigten. Beachtenswert ist der Zugriff auf die Feldwerte, der über das DOM-Modell des Browser erfolgt:

```
isbn = document.checkform.f_isbn.value;
```

Damit das funktioniert, wurde das Formular mit der ID *checkform* versehen, die in HTML auch als Attribut *name* verwendet wird.

Im Beispiel erfolgt die Ausgabe der Fehlermeldung bei falscher Anzahl Ziffern über die *alert*-Funktion von JavaScript. Wenn Sie die Ausgabe über das Label erledigen möchten, müssen Sie den Zugriff auf die HTML-Elemente über das Objektmodell des Browsers kennen. ASP.NET erzeugt für *<asp:label>* ein **-Element, das über das Attribut *id* erreicht werden kann. Tauschen Sie die Zeile mit *alert* gegen folgende aus, um dynamisch auf HTML zugreifen zu können:

```
document.getElementById("l_isbn").innerHTML +=
    "Anzahl Ziffern falsch: " + isbn10.length;
```

Mit *getElementById(id)* kann im Browser jederzeit auf alle Steuerelemente der Seite zugegriffen werden, die das Attribut *id* tragen. Das Attribut *runat="server"* kennt JavaScript nicht, es ist nicht notwendig, stört aber auch nicht.

Abbildung 6.40: Reaktion des Programms auf eine falsche ISBN



Im Sinne eines gutes Programmstils sollten Sie immer client- und serverseitige Steuerelemente realisieren. Wenn der Browser JavaScript unterstützt, spart es Bandbreite und die Verarbeitung der Seite wirkt auf den Benutzer schneller. Ist JavaScript nicht einsetzbar, funktioniert es trotzdem.

6.8 Komplexe Steuerelemente (Rich Controls)

ASP.NET liefert nicht nur elementare Steuerelemente, sondern auch außerordentlich komplexe. Dennoch wird in allen Fällen am Ende HTML erzeugt. Die Steuerelemente sind auch gute Beispiele dafür, wie Kunden-Steuerelemente eingesetzt werden können.

6.8.1 Kalender anzeigen mit Calendar

Eines der interessantesten Steuerelemente ist das Kalender-Steuer-element (Calendar). Zuerst eine Präsentation:



Abbildung 6.41: Interaktiver Kalender

Wie lange brauchen Sie, um dies mit herkömmlichen Elementen zu programmieren? Immerhin besitzt der Kalender Blätterfunktionen zum Wechsel der Monate. Der ausgewählte Tag wird grau hinterlegt und erscheint bei Mausklick unterhalb des Kalenders. Das Programm selbst – tatsächlich vollständig:

```
<html lang="de">
  <head>
    <title>Kalender</title>
    <basefont face="Arial">
  </head>
  <body>
    <h1>Kalender</h1>
    <form runat="server">
      <asp:calendar id="kalender" runat="server"
        onSelectionChanged="zeigeDatum"/>
```

```

        <hr noshade="noshade" width="200px" align="left"/>
        <asp:label id="auswahl" runat="server"/>
    </form>
</body>
</html>

```

Listing 6.61: Ein interaktiver Kalender (Calendar.aspx)

```

Public Class Calendar
    Inherits System.Web.UI.Page
    Protected WithEvents kalender ◀
        As System.Web.UI.WebControls.Calendar
    Protected auswahl As Label

    Protected Sub zeigeDatum(ByVal sender As Object, ◀
        ByVal e As EventArgs) _
        Handles kalender.SelectionChanged
        kalender.TodaysDate = kalender.SelectedDate
        auswahl.Text = "Ihre Auswahl: " ◀
            + kalender.TodaysDate.ToShortDateString()
    End Sub
End Class

```

Listing 6.62: Code zu Calendar.aspx (Ausschnitt aus Calendar.aspx.vb)

Wie es funktioniert

Im Kern basiert das Programm auf dem Web Server-Steuerelement Calendar:

```

<asp:calendar id="kalender" runat="server" ◀
    onSelectionChanged="zeigeDatum"/>

```

Hier wird nur eines der sehr zahlreichen Attribute verwendet: onSelectionChanged. Wann immer die Auswahl geändert wird, ruft ASP.NET die Methode *zeigeDatum* auf.

Hier passiert auch nicht viel. Zuerst der Zugriff auf das ausgewählte Datum, das zugleich zum aktuellen gemacht wird:

```
kalender.TodaysDate = kalender.SelectedDate
```

Dann wird diese Angabe in ein passendes Format gebracht und an das Label übergeben:

```

auswahl.Text = "Ihre Auswahl: " ◀
    + kalender.TodaysDate.ToShortDateString()

```

Das war's auch schon. In der Referenz finden Sie zahlreiche Eigenschaften zur Gestaltung und Methoden zur Behandlung der ermittelten Datumswerte.

Das Kalender-Steuerelement in der Praxis

Das Kalender-Steuerelement basiert auf zwei grundlegenden Techniken, die auch in anderen Steuerelementen zum Einsatz kommen:

- ▶ Die Gestaltung erfolgt mit Stil-Vorlagen
- ▶ Die Übernahme der ausgewählten Daten erledigt die Ereignisverarbeitung

Jedes Element des Kalenders kann über einen eigenen Stil gesteuert werden. Die in Abbildung 6.41 gezeigte Version ist nahezu ungestaltet, außer dem global bestimmten Font wurden keine Stilvorlagen verwendet. Die folgende Tabelle zeigt alle zulässigen Stile und deren Bedeutung.

Stilvorlage

Stil	Bedeutung
<TodayDayStyle>	Der aktuelle Tag (Heute)
<WeekEndDayStyle>	Das Wochenende
<DayHeaderStyle>	Überschrift mit Wochentagsnamen
<SelectedDayStyle>	Der ausgewählte Tag
<DayStyle>	Alle anderen Tage
<TitleStyle>	Der Titel des Kalenders
<NextPrevStyle>	Stil der Blättersymbole »<« und »>«
<OtherMonthDayStyle>	Wechsel auf einen anderen Monat
<SelectorStyle>	Auswahl für eine Woche oder den gesamten Monat

Tabelle 6.13: Stilvorlagen zur Gestaltung des *Calendar*-Steuerelementes

Es ist auch möglich, Teile des Kalenders auszublenden, wenn die betreffenden Funktionen nicht benötigt werden. Dazu gibt es vier Attribute sowie – in der Code-Darstellung – vier gleichnamige Eigenschaften:

- ▶ ShowDayHeader
Zeigt, wenn *true*, die Überschrift mit den Wochentagsnamen an.
- ▶ ShowGridLines
Zeigt die Gitterlinien zwischen den Tagen an oder blendet sie aus.

► ShowNextPrevMonth

Schaltet die Blätterfunktion »vorhergehender Monat« und »nächster Monat« ein oder aus.

► ShowTitle

Schaltet den Titel des Kalenders ein oder aus.

Beispiel Der Umgang mit dem Kalender ist in einigen Details nicht trivial, vor allem, wenn bestimmte Designansprüche vorgegeben sind, die das Element nicht direkt unterstützt. Das folgende Beispiel zeigt, wie sie `Calendar` programmieren können und welche Probleme es dabei gibt.

Realisiert werden soll ein einfaches Formular, das eine Datums-eingabe zulässt. Wählt der Benutzer statt der direkten Eingabe einen entsprechenden Link, wird ein Kalender in einem besonderen Fenster aufgeblendet und der Benutzer kann eine Auswahl treffen. Die Auswahl wird sofort in das Formular übernommen.

Zuerst soll das Formular vorgestellt werden:

```
<%@ Page language="vb" Codebehind="GetDateFromCalendar.
aspx.vb" ␣
        AutoEventWireup="false" ␣
        Inherits="Addison.VBNet.WebForm.GetDateFromCalendar" %>
<html>
<head>
<title>GetDateFromCalendar</title>
<script language="javascript">
    if (typeof(oWin) != 'undefined')
    {
        oWin.close ();
    }
    function GetCalendar (winurl)
    {
        self.name = "GetDateFromCalendar";
        oWin = window.open (winurl, "Calendar", ␣
                            "height=100,width=80,menubar=0,␣
                            toolbar=0,status=0,directories=0,␣
                            resizable=0,scrollbars=0,location=0");
    }
</script>
</head>
<body MS_POSITIONING="GridLayout">
<basefont face="Verdana"/>
<h1>Kalenderfunktionen</h1>
<form id="GetDateFromCalendar" name="GetDateFromCalendar" ␣
        method="post" runat="server">
```

```

        <asp:TextBox Runat="server" ID="StartDatum"/>
        <asp:HyperLink Runat="server" ID="StartDatumCalendar"/>
    </form>
</body>
</html>

```

Listing 6.63: Die gestalterische Seite des Formulars (GetDateFromCalendar.aspx)

Hier wird eine JavaScript-Funktion verwendet, um eine andere *aspx*-Datei in einem weiteren Fenster zu öffnen.

Die Code-Datei dazu bietet wenig eigene Funktionalität, lediglich der Link wird mit dem Aufruf der JavaScript-Funktion gefüllt:

```

Public Class GetDateFromCalendar
    Inherits System.Web.UI.Page
    Protected StartDatumCalendar As HyperLink
    Protected StartDatum As TextBox

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        StartDatumCalendar.Text = "Auswahl"
        StartDatumCalendar.NavigateUrl _
            = "javascript:GetCalendar('GetDateCalendar.aspx');"
        If Not Request.QueryString("date") Is Nothing Then
            StartDatum.Text = Request.QueryString("date")
        End If
    End Sub
End Class

```

Listing 6.64: Der Code zum Eingabeformular (GetDateFromCalendar.aspx.vb)

Damit existiert bereits ein primitives Formular. Von einer Weiterverarbeitung der Datumswerte soll hier abgesehen werden. Dazu finden Sie in den vorhergehenden Abschnitten ausreichend Informationen.

Kalenderfunktionen

Mittwoch, 7. August 2002 Auswahl

Abbildung 6.42: Das Formular mit direkter Eingabe eines Datums

Die Datei *GetDateCalendar.aspx* dient nun der Darstellung des Kalenders. Zuerst die vollständige Definition. Sie können hier gut die Verwendung der Vorlagen erkennen:

```

<%@ Page language="vb"
    Codebehind="GetDateCalendar.aspx.vb"
    AutoEventWireup="false"
    Inherits="Addison.VBNet.WebForm.GetDateCalendar" %>

<HTML>
<HEAD>
    <title>GetDateCalendar</title>
</HEAD>
<body MS_POSITIONING="GridLayout">
<form id="GetDateCalendar" name="GetDateCalendar"
    method="post" runat="server">
<TABLE id="Table1" style="Z-INDEX: 103; LEFT: 7px;
    POSITION: absolute; TOP: 5px"
    cellSpacing="1" cellPadding="2"
    width="300" border="0">

<TR>
    <TD><h1>Datumsauswahl</h1>
    </TD>
<TD></TD>
</TR>
<TR>
<TD>
<asp:Calendar id="Calendar1" runat="server"
    BorderStyle="Solid" NextPrevFormat="ShortMonth"
    BackColor="White" Width="330px" ForeColor="Black"
    CellSpacing="1" Height="250px" Font-Size="9pt"
    Font-Names="Verdana" BorderColor="Black"
    ToolTip="Wählen Sie das gewünschte Datum aus">
    <TodayDayStyle ForeColor="White" BackColor="#999999"/>
    <DayStyle BackColor="#CCCCCC"/>
    <NextPrevStyle Font-Size="8pt" Font-Bold="True"
        ForeColor="White"/>
    <DayHeaderStyle Font-Size="8pt" Font-Bold="True"
        Height="8pt" ForeColor="#333333"/>
    <SelectedDayStyle ForeColor="White" BackColor="#333399"/>
    <TitleStyle Font-Size="12pt" Font-Bold="True"
        Height="12pt" ForeColor="White"
        BackColor="#333399"/>
    <OtherMonthDayStyle ForeColor="#999999"/>
    </asp:Calendar></TD>
<TD vAlign="top" align="left">
    <P style="FONT-FAMILY: Verdana">Hier&nbsp;steht&nbsp;
        eine
        Erklärung&nbsp;zu&nbsp;der Verwendung&nbsp;dieser
        Auswahlfunktion</P>
    <asp:HyperLink Runat="server" Target="_self"
        NavigateUrl="GetDateCalendar.aspx"
        ID="SetToDay" Font-Names="Verdana"
        Font-Bold="True">Heute</asp:HyperLink>

```

```

        </TD>
    </TR>
    <TR>
        <TD align="middle">
            <asp:HyperLink Runat="server" ID="TransferDate"
                Target="GetDateFromCalendar" Font-Names="Verdana"
                Font-Bold="True">Noch keine Auswahl getroffen
            </asp:HyperLink>
        </TD>
        <TD>
            <input type="submit" name="Close" onclick="self.
                close()"
                value="Schließen">
        </TD>
    </TR>
</TABLE>
</form>
</body>
</HTML>

```

Listing 6.65: Die Darstellung des Kalenders (*GetDateCalendar.aspx*)

Dargestellt wird hier ein Kalender innerhalb einer Tabelle. Unterhalb des Kalenders gibt es ein `HyperLink`-Steuerelement, das solange deaktiviert ist, wie noch keine Datumsauswahl getroffen wurde. Daneben liegt eine einfache Schaltfläche, die nur zum Schließen des Fensters dient und nicht Bestandteil der Steuerelemente-Kollektion ist. Mit dem Link »Heute« kann der Kalender jederzeit wieder auf das aktuelle Datum gesetzt werden.

Diesem Formular schließt sich eine Code-Datei an:

```

Public Class GetDateCalendar
    Inherits System.Web.UI.Page
    Protected WithEvents Calendar1 As
        System.Web.UI.WebControls.Calendar
    Protected TransferDate As HyperLink
    Protected SetToday As HyperLink
    Protected CurrentDateSelection As HtmlInputHidden
    Protected Header As HtmlGenericControl

    Protected Sub Calendar1_SelectionChanged(ByVal sender As
        Object,
        ByVal e As
        System.EventArgs) _
        Handles Calendar1.SelectionChanged
        Dim NewDate As String =
        Calendar1.SelectedDate.ToLongDateString()
        TransferDate.NavigateUrl

```

```

        = String.Format("GetDateFromCalendar.aspx?date={0}",
NewDate)
        TransferDate.Text = NewDate + " übernehmen?"
        Calendar1.TodaysDate = Calendar1.SelectedDate
    End Sub

    Protected Sub SetDateToDay(ByVal sender As Object,
                                ByVal e As EventArgs)
        Calendar1.TodaysDate = Now
    End Sub
End Class

```

Listing 6.66: Der Code, der den Kalender bedient (GetDateCalendar.aspx.vb)

Wie es funktioniert

Die Funktionalität ist bescheiden. Abgefangen wird das Ereignis `SelectionChanged`, also ein Ändern der aktuellen Datumsauswahl. Nun wird das Datum ermittelt:

```
Dim NewDate As String = Calendar1.SelectedDate.ToLongDateString()
```



Abbildung 6.43: Auswahl eines Datums mit dem Kalender

Dann wird der Link zusammengestellt, der die ursprüngliche Seite erneut aufruft. Dabei wird das Datum per GET übergeben:

```

TransferDate.NavigateUrl
    = String.Format("GetDateFromCalendar.aspx?date={0}",
NewDate)

```

Nun wird der Link zur Übernahme des Datums aktiviert:

```
TransferDate.Text = NewDate + " übernehmen?"
```

Zuletzt wird das aktuelle Datum des Kalenders auf die Auswahl gesetzt, damit die Anzeige nicht zurückspringt, was recht irritierend für den Benutzer ist:

```
Calendar1.TodaysDate = Calendar1.SelectedDate
```

Mit einem Klick auf den mit »Heute« beschrifteten Link kann das aktuelle Datum wieder eingestellt werden. Die Ereignisbehandlungsmethode dafür enthält folgenden Code:

```
Calendar1.TodaysDate = Now
```

Die Steuerung eines zweiten Fensters auf so direktem Wege, mit GET, ist zwar einfach, aber nicht ideal. Denn beim erneuten Aufruf der ersten Seite per GET gehen die Feldwerte und der Anzeige-status verloren. Die Übergabe per POST scheitert aus, weil sich durch den gegenseitigen Aufruf Probleme mit dem Anzeigestatus ergeben. Diese Funktion dient nur der Erhaltung eines Status, wenn sich eine Seite selbst aufruft.

Probleme mit diesem Programm

In der Praxis können Sie die Übergabe der Werte über zwei Wege erreichen:

Lösungen

► Verwendung von *Sitzungsvariablen*

Sitzungen sind ein einfaches Mittel zur Lösung. Sie finden dazu ausführliche Informationen in Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778.

► Referenzierung anderer Seiten über den so genannten *Seitenkontext*

Die Referenzierung ist der »moderne« Weg, der den Paradigmen von ASP.NET folgt. Diese Technik wird in Abschnitt 8.3.8, »Kontext-Handler und Seitenreferenzierung« ab Seite 773 behandelt.

6.8.2 Weitere Steuerelemente

Zwei weitere Steuerelemente sollen hier nur erwähnt werden. Sie sind in der Referenz gut dokumentiert und funktionieren nach den bereits gezeigten Prinzipien.

AdRotator

Dieses Element dient der Speicherung von Werbeinformationen. Über eine Steuerdatei kann das Einblenden von Werbebannern kontrolliert werden. Das Element hat folgenden Namen:

```
<asp:AdRotator>
```

Die Steuerdatei ist selbstredend eine XML-Datei und hat folgenden Aufbau:

```
<Advertisement>
  <Ad>
    <ImageUrl>
      Bildpfad    </ImageUrl>
    ...
  </Ad>
</Advertisement>
```

Neben dem Bild sind weitere Informationen hinterlegbar. Existieren mehrere <Ad>-Elemente, wechselt die Anzeige zwischen diesen. Durch Einstellen einer Priorität kann die Anzeigefrequenz einzelner Banner gesteuert werden.

XML

Dieses Element dient der Anzeige von XML-Daten innerhalb einer Seite. Es wird folgendermaßen benannt:

```
<asp:xml>
```

Als Parameter werden die Angabe der XML-Datei und die einer XSLT-Datei erwartet. Die XSLT-Datei sollte die Daten in ein lesbares Format überführen.



Dieses Element stellt nicht transformierte Daten unverändert dar. Da der Browser damit nichts anfangen kann, unterbleibt die Anzeige dann. Sie müssen also immer eine Transformation vornehmen, wenn Sie etwas sehen wollen.

Mehr Informationen zu XSLT finden Sie im Abschnitt 5.9, »Daten transformieren mit XSLT« ab Seite 478.

7 Erweiterte Web Form-Programmierung

Nach der Einführung in die Welt der »Web Forms« – also der Programmierung nach der Gestaltung der Benutzerschnittstelle – soll nun auf einige erweiterte Techniken eingegangen werden. In diesem Kapitel wird die Erstellung eigener Steuerelemente vorgestellt und gezeigt, wie Sie Steuerelemente nachträglich installieren und verwenden.

7.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

7.1.1 Über dieses Kapitel

Dieses Kapitel führt die im vorhergehenden begonnenen Ausführungen über die Programmierung der Benutzerschnittstelle fort. Vorgestellt wird eine Technik zum Programmieren wieder verwendbarer Steuerelemente in Abschnitt 7.2, »Modularer Code mit Benutzer-Steuerelementen« ab Seite 647.

Das sich die Browser nachwievor in der Art der Darstellung von komplexerem HTML-Code etwas unterscheiden, wird auf die browserabhängige Programmierung eingegangen. Dabei geht es darum, die Leistungen des Browsers zu erkennen und den dazu passenden Code ablaufen zu lassen. Abschnitt 7.3, »Browserorientiert programmieren« ab Seite 662 widmet sich diesem Thema.

Danach finden Sie auch Informationen über *SmartNavigation*, einem speziell für den Internet Explorer entwickelte Funktion.

Zwar nicht im Lieferumfang aber zum freien Download werden vier komplexe Steuerelemente von Microsoft angeboten, die bei umfangreicheren Gestaltungsaufgaben wertvolle Hilfe leisten können: Abschnitt 7.5, »Zusätzliche Steuerelemente: WebControls 1.0« ab Seite 676 geht darauf detailliert ein.

7.1.2 Wegweiser in die Referenz

Für die Programmierung von Benutzer-Steuerelementen wird als Basis die Klasse `UserControl` verwendet:

```
System.Object
├─ System.Web.UI.Control
│   └─ System.Web.UI.TemplateControl
│       └─ System.Web.UI.UserControl
```

Abbildung 7.1: Ableitung der Klasse *UserControl*

Auf gleicher Basis aber auf anderem Weg wird `WebControl` abgeleitet, die Klasse, die als Basis kundenspezifischen Steuerelemente dient:

```
System.Object
├─ System.Web.UI.Control
│   └─ System.Web.UI.BasePartialCachingControl
│       └─ System.Web.UI.DataBindLiteralControl
│           └─ System.Web.UI.HtmlControls.HtmlControl
│               └─ System.Web.UI.LiteralControl
│                   └─ System.Web.UI.TemplateControl
│                       └─ System.Web.UI.WebControls.Literal
│                           └─ System.Web.UI.WebControls.PlaceHolder
│                               └─ System.Web.UI.WebControls.Repeater
│                                   └─ System.Web.UI.WebControls.RepeaterItem
│                                       └─ System.Web.UI.WebControls.WebControl
│                                           └─ System.Web.UI.WebControls.Xml
```

Abbildung 7.2: Ableitung der Klasse *WebControl*

7.2 Modularer Code mit Benutzer-Steuerelementen

ASP.NET bietet bei der Programmierung großer Projekte eine starke Unterstützung für den Entwickler. Eines der wichtigen Instrumente zur Modularisierung sind Benutzer-Steuerelemente (User Controls).

7.2.1 Grundlagen

Benutzer-Steuerelemente sind so einfach und flexibel, dass der Einsatz sich auch schon bei kleineren Projekten lohnen kann. Praktisch werden hier Objekte definiert, die ähnlich wie die Web Server-Steuerelemente einen eigenen Namensraum besitzen. Wenn die Erweiterung von HTML über den Namensraum »asp:« gelingt, sollte es auch möglich sein, eigene Namensräume zu deklarieren. Tatsächlich ist dies kein Trick, sondern vom Systemdesign vorgegeben. Eine solche Erweiterung wird als Benutzer-Steuerelement bezeichnet. Welches Präfix verwendet wird, können Sie selbst bestimmen.

Da Benutzer-Steuerelemente per Definition zur Gruppe der Web Server-Steuerelemente gehören, verhalten sie sich auch ähnlich. Der Inhalt eines Benutzer-Steuerelementes darf nur aus HTML-Tags bestehen, wie er zwischen den `<body>`-Tags stehen kann. `<html>` und `<body>` selbst dürfen nicht auftreten. Ebenso sollte das `<form>`-Tag vermieden werden, da die ASP.NET-Komponente erwartet, das Benutzer-Steuerelemente innerhalb oder außerhalb eines Formulars erscheinen, aber dieses nicht selbst bilden. Benutzer-Steuerelemente sind in ihrem Einsatzspektrum dennoch kaum eingeschränkt. Was immer Sie in Ihrem Projekt mehr als einmal benötigen, können Sie so definieren. Durch die Möglichkeit, auf die enthaltenen Elemente programmatisch zugreifen zu können, ist die Anwendung sehr flexibel möglich.

Benutzer-Steuerelemente sind jedoch kein Allheilmittel. Es gibt in ASP.NET mehr Technologien zur Modularisierung. Einsetzen sollten Sie sie in folgenden Situationen:

- ▶ Elemente einer HTML-Seite werden mehrfach benötigt, beispielsweise bei Menüs, Kopf- und Fußzeilen etc.
- ▶ Zur Reduzierung von Code in einer Seite, das heißt, zur Erhöhung der Übersichtlichkeit

Aufbau und Einsatz

In anderen Fällen gibt es bessere Alternativen:

- ▶ Bereitstellung von Steuerelementen für fremde Applikationen (dafür sind Kundenspezifische Steuerelemente besser geeignet). Informationen dazu enthält Abschnitt 7.6, »Kundenspezifische Steuerelemente (Custom Controls)« ab Seite 706.
- ▶ Kapselung von Code für fremde Applikationen (dafür sind kompilierte Assemblies gedacht)
- ▶ Trennung von Code und Design (dies wird mit Code Behind gemacht, was bereits in Abschnitt 1.5.2, »Code Behind« ab Seite 79 bereits betrachtet wurde).

7.2.2 Wie Benutzer-Steuerelemente entstehen

Erweiterung .ascx Benutzer-Steuerelemente entstehen in zwei Schritten. Zuerst wird eine neue Datei angelegt, die den HTML-Code aufnimmt. Diese Datei muss die Endung *.ascx* bekommen. Dies ist zwingend vorgeschrieben, damit der Baustein beim Übersetzen der Seite korrekt eingebunden wird.

Benutzer-Steuerelemente in Visual Studio .NET

Wenn Sie Visual Studio .NET verwenden, erzeugen Sie ein neues Benutzer-Steuerelement wie folgt:

1. Im PROJEKTMAPPEN-EXPLORER klicken Sie mit der rechten Maustaste auf ein Projekt.
2. Wählen Sie im Kontextmenü HINZUFÜGEN und dann WEB-BENUTZER-STEUERELEMENT HINZUFÜGEN.
3. In der folgenden Liste der Vorlagen belassen Sie die Auswahl auf BENUTZERSTEUERELEMENT.
4. Vergeben Sie einen Namen für das neue Element. Klicken Sie auf OK.

Das Element wird in Visual Studio .NET immer mit einer Code Behind-Datei versehen.

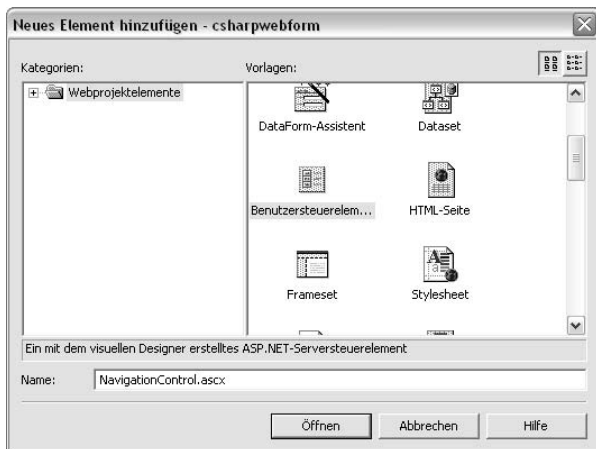


Abbildung 7.3: Ein neues Benutzer-Steuerelement in Visual Studio .NET

Ein einfaches Anwendungsbeispiel

Als Beispiel soll hier eine einfache Navigation aufgebaut werden, die auf jeder Seite der Applikation erscheint. Über den Programmcode soll eine Eigenschaft so verändert werden, dass die aktuelle Seite in den Navigationselementen farbig hervorgehoben wird. Das folgende Beispiel zeigt den Aufbau eines einfachen Benutzer-Steuerelementes:

```
<table border="1">
  <tr>
    <td>
      <a href="home.aspx">Startseite</a>
    </td>
    <td>
      <a href="impressum.aspx">Impressum</a>
    </td>
    <td>
      <a href="produkte.aspx">Produkte</a>
    </td>
  </tr>
</table>
```

Listing 7.1: Einfache Navigation (UserControlsNavigation.ascx)

Im zweiten Schritt muss das Benutzer-Steuerelement nun bekannt gemacht werden. Es soll auf jeder Seite der Applikation eingebaut

@Register

werden. Dazu braucht es einen Präfix¹ und einen Namen. Bislang fehlte noch die Definition des Präfixes, der für das XML-Tag verwendet wird. ASP.NET trifft hier keine Vorgabe. Hier soll »uc:« verwendet werden. Die Anmeldung wird über die Direktive @Register vorgenommen:

```
<% @Register TagPrefix="uc" TagName="navigation" %>
           src="UserControlsNavigation.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
  <head>
    <title>Home</title>
  </head>
  <body>
    <uc:navigation id="nav" runat="server"/>
    <h1>Startseite</h1>
    Das ist die Startseite
  </body>
</html>
```

Listing 7.2: Nutzung des User Controls (UserControlsHome.aspx)

Präfix und Tag-Name

Die Direktive bestimmt mit dem Attribut `TagPrefix`, welches Präfix verwendet wird. `TagName` definiert, wie das Tag auf dieser Seite heißt. Ein- und derselbe Code kann also nicht nur mehrfach verwendet, sondern auch verschieden benannt werden. Natürlich muss noch die Quelle mit dem Attribut `src` benannt werden. Auf der Seite selbst können Sie den Baustein nun folgendermaßen einbauen:

```
<uc:navigation id="nav" runat="server"/>
```

Das Attribut `runat="server"` ist wieder obligatorisch. Da Benutzer-Steuerelemente auch als Objekte im Code zur Verfügung stehen, ist die Benennung mit `id="ucname"` erforderlich. Der Name ist im Rahmen der üblichen Bedingungen frei wählbar. Aber auch ohne Zugriff über Programmcode wird es nun bereits in der Seite angezeigt.

¹ Im Sinne korrekter Auslegung des XML-Standards handelt es sich hier um einen Alias für einen Namensraum. Allerdings sieht das ASP.NET nicht so eng – die Definition des Namensraumes bleibt Ihnen erspart.



Abbildung 7.4: Die Navigationslinks oben entstehen durch ein Benutzer-Steuerelement.

Programmierung von Benutzer-Steuerelementen

Spannender ist freilich der Zugriff über den Programmcode, denn nur so entfalten Benutzer-Steuerelemente ihr gesamtes Leistungsspektrum. Das .NET-Framework besteht aus einer ausgefeilten und komplexen Hierarchie von Klassen. Jedes Objekt stammt in der einen oder anderen Form von einer der Klassen ab. Das trifft zwangsläufig auch auf Benutzer-Steuerelemente zu. Diese stammen aus der Klasse `Controls`, aus der auch HTML Server-Steuerelemente und Web Server-Steuerelemente abgeleitet wurden. Aus `Controls` erbt auch `TemplateControls`; aus dieser Klasse wird die Basisklasse `UserControl` abgeleitet, aus der letztlich auch die Objekte der Benutzer-Steuerelemente instanziiert werden. Damit stehen Baseeigenschaften wie beispielsweise `Visible` sofort zur Verfügung, ohne dass eine einzige Codezeile erforderlich wäre. Der Name des Objekts wird durch das Attribut `id` festgelegt. Auf die Elemente des Steuerelements können Sie nicht so direkt zugreifen – dazu jedoch später mehr. Der Zugriff auf die Elemente vom Programm der aufrufenden Seite ist natürlich möglich, allerdings nicht direkt. Denn als Entwickler eines Benutzer-Steuerelements sollten Sie die Kontrolle darüber behalten, welche Details sich beeinflussen lassen und welche nicht. Es ist nahe liegend, dass der Code dazu im Steuerelement selbst untergebracht wird. Wenn nun aber Code geschrieben wird, sind auch Direktiven notwendig. Tatsächlich gibt es kaum Unterschiede zu einer normalen *aspx*-Seite. Statt der Direktive `@Page` wird das Benutzer-Steuerelement mit `@Control` ausgestattet. Sämtliche Attribute und Parameter sind mit `@Page` identisch, mit Ausnahme von `trace` – diese Option kann nicht im Steuerelement ein- und ausgeschaltet werden. Stehen nun alle Techniken zur Verfügung, die *aspx*-Seiten bieten, ist es

Zugriff aus dem Code

auch leicht möglich, Teile des Benutzer-Steuerelements als Eigenschaften zur Verfügung zu stellen. Eine genaue Auflistung der Attribute von `@Control` finden Sie in Abschnitt 10.2.6, »Die Direktive `@Control`« ab Seite 961.

Die Eigenschaften der Benutzer-Steuerelemente

Eigenschaften der Bausteine

Im Beispiel wäre es nahe liegend, die Hintergrundfarbe der Tabellenzeile einzufärben, deren Link auf die gerade aktive Seite zeigt. Dies soll als neue Eigenschaft des Benutzer-Steuerelements dem aufrufenden Programm bereit gestellt werden. Das folgende Beispiel enthält das derart erweiterte Steuerelement zur Navigation. Beachten Sie, dass der direkte Zugriff nicht möglich ist, Sie müssen die erforderlichen Eigenschaften über Programmcode zur Verfügung stellen. Definiert werden dazu Eigenschaften der das Steuerelement repräsentierenden Klasse. Im Beispiel wird eine nur schreibbare Eigenschaft benötigt; es genügt also, den `Set-`Zweig zu definieren.

```
<%@ Control CodeBehind="UserControlsNavigation2.ascx.vb"
        Language="vb" AutoEventWireup="false"

Inherits="Addison.VBNet.WebForm.UserControlsNavigation2" %>
<table border="1">
  <tr>
    <td id="td1" runat="server">
      <asp:hyperlink id="link1" runat="server" />
    </td>
    <td id="td2" runat="server">
      <asp:hyperlink id="link2" runat="server" />
    </td>
    <td id="td3" runat="server">
      <asp:hyperlink id="link3" runat="server" />
    </td>
  </tr>
</table>
```

Listing 7.3: Umfangreiches Benutzer-Steuerelement mit Steuerung der Eigenschaften (UserControlsNavigation2.ascx)

Der der Eigenschaft zugewiesene Wert ist über das Schlüsselwort *value* erreichbar. *td1* bis *td3* repräsentieren die zum HTML Server-Steuerelement konvertierten Tabellenzellen, in welchen die Links stehen. Wird eine der Eigenschaften mit dem Parameter `True` oder `False` aufgerufen, wird der entsprechenden Link aktiviert bzw. deaktiviert. Wie das funktioniert, zeigt der Code dazu:


```

Public MustInherit Class UserControlNavigation2
    Inherits System.Web.UI.UserControl
    Protected WithEvents link1 As HyperLink
    Protected WithEvents link2 As HyperLink
    Protected WithEvents link3 As HyperLink
    Protected WithEvents td1 As HtmlTableCell
    Protected WithEvents td2 As HtmlTableCell
    Protected WithEvents td3 As HtmlTableCell

    Private Sub Page_Load(ByVal sender As System.Object, 
        ByVal e As System.EventArgs) _
    Handles MyBase.Load
        link1.Text = "Startseite"
        link1.NavigateUrl = "UserControlsHome2.aspx"
        link2.Text = "Impressum"
        link2.NavigateUrl = "UserControlsImpressum2.aspx"
        link3.Text = "Produkte"
        link3.NavigateUrl = "UserCOntrolsProdukte2.aspx"
    End Sub

    Public WriteOnly Property nav_link1() As Boolean
        Set(ByVal Value As Boolean)
            If Value Then
                td1.Style("Background-Color") = "#dddddd"
            Else
                td1.Style("Background-Color") = "#ffffff"
            End If
        End Set
    End Property

    Public WriteOnly Property nav_link2() As Boolean
        Set(ByVal Value As Boolean)
            If (Value) Then
                td2.Style("Background-Color") = "#dddddd"
            Else
                td2.Style("Background-Color") = "#ffffff"
            End If
        End Set
    End Property

    Public WriteOnly Property nav_link3() As Boolean
        Set(ByVal Value As Boolean)
            If (Value) Then
                td3.Style("Background-Color") = "#dddddd"
            Else
                td3.Style("Background-Color") = "#ffffff"
            End If
        End Set
    End Property

```

```

        End Set
    End Property
End Class

```

*Listing 7.4: Kontrolle der Links eines Benutzer-Steuerelements
(UserControlsNavigation2.ascx.vb)*

Das folgende Programm zeigt nun noch, wie der Einbau des Steuerelementes und dessen Nutzung erfolgt:

```

<%@ Page language="vb" debug="true" ␣
    CodeBehind="UserControlsHome2.aspx.vb" ␣
    AutoEventWireup="false" ␣
    Inherits="Addison.VBNet.WebForm.UserControlsHome2" %>
<% @Register TagPrefix="uc" TagName="navigation" ␣
    src="UserControlsNavigation2.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML lang="en">
  <HEAD>
    <title>Home</title>
  </HEAD>
  <body>
    <uc:navigation id="nav" runat="server" />
    <h1>Startseite</h1>
    Das ist die Startseite
  </body>
</HTML>

```

*Listing 7.5: Nutzung des komplexeren Benutzer-Steuerelement
(UserControlsHome2.aspx)*

```

Public Class UserControlsHome2
    Inherits System.Web.UI.Page
    Protected nav As UserControlsNavigation2

    Private Sub Page_Load(ByVal sender As System.Object, ␣
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        nav.nav_link1 = True
    End Sub
End Class

```

Listing 7.6: Code zum Benutzer-Steuerelement (UserControlsHome2.aspx.vb)

Wie es funktioniert

Über `nav.nav_link1` wird auf eine der Eigenschaften zugegriffen. Das erscheint auf den ersten Blick umständlich, immerhin wäre es einfacher, beispielsweise `nav.td1.Style("Background-Color") = "#ddddd"` zu schreiben, um ein Teil des Steuerelements farblich zu verändern. Allerdings wären dann alle Teile öffentlich zugänglich, was im Sinne eines guten Programmierstils nicht gut ist. Sie

sollten immer genau die Eigenschaften exakt definieren, die auch tatsächlich benötigt werden. .NET erzwingt das kurzerhand. Das verbesserte Benutzer-Steuerelement ist auch in allen anderen Dateien zu finden, wobei jeweils eine andere Eigenschaft verwendet wird. Benutzer-Steuerelemente können also nicht nur flexibel eingesetzt werden, sondern auch umfangreiche Aktionen ausführen. Dazu gehören auch Datenbankabfragen, Zugriffe auf das Dateisystem oder Interaktionen mit der aktuellen Seite.

Wenn Sie in ASP.NET programmieren, sollten Sie sich intensiv mit Web Server-Steuerelementen oder Benutzer-Steuerelementen auseinandersetzen. Es sind Kernelemente der Programmierung und ebnen den Weg zu großen und stabilen Anwendungen.



7.2.3 Spezielle Techniken der Benutzer-Steuerelemente

Die Philosophie, die dahinter steckt, ist grundlegend anders als die im alten ASP vermittelte. Anstatt HTML und Code wild zu vermischen, ist eine strenge Trennung nun möglich und empfehlenswert. Die Auslagerung von Code in externe Dateien mit Code Behind ist Ihnen sicher bereits vertraut, die letzten Beispiele nutzen das auch für Benutzer-Steuerelemente. Hier sollen nun ein paar Hintergründe dieser Technik behandelt werden.

Code der Benutzer-Steuerelemente auslagern @Control

Da die @Control-Direktive den gesamten Umfang der @Page-Direktive bietet, ist dies auch mit *ascx*-Dateien möglich. Eine genaue Auflistung der Attribute von @Control finden Sie in Abschnitt 10.2.6, »Die Direktive @Control« ab Seite 961. Wenn Sie externen Code verwenden, leiten Sie Ihre eigenen Klassen normalerweise von der Klasse *Page* ab, um über die Eigenschaften und Methoden der Seite verfügen zu können. Benutzer-Steuerelemente erben auch auf diese Weise, allerdings von der Klasse *UserControl*:

```
Public Class NavigationClass Inherits UserControl
```

Das Beispiel mit der Navigation lässt sich mit dieser Technik gut erweitern. So stehen die Zellen der Tabelle im Moment nebeneinander, ideal für eine Navigation am oberen Rand der Seite. Soll dieselbe Navigation links erscheinen, ist es besser, die Zellen untereinander anzuordnen. Da sich der HTML-Code völlig vom bereits gezeigten unterscheidet, ist ein neues Benutzer-Steuerelement zu entwerfen. Der VB.NET-Code, der den Zugriff auf die

Eigenschaften gestattet, ist dagegen mit dem bereits gezeigten identisch. Es ist mehr als nur guter Programmierstil, den Code mehrfach zu verwenden. In der Kopfzeile wird dazu folgende Direktive geschrieben: `<% @Control src=" " ... %>`.

Der Code entspricht dem in *UserControlHome2.aspx.vb* gezeigten; für dieses Beispiel heißt die Datei *userControlHome3.aspx*. Auf dieser Basis sind jetzt zwei Benutzer-Steuerelemente verfügbar, die denselben Code verwenden. Der Einbau in einer HTML-Seite kann folgendermaßen aussehen:

```
<%@ Page language="vb" CodeBehind="UserControlsHome3.aspx.vb"
    AutoEventWireup="false"
    Inherits="Addison.VBNet.WebForm.UserControlsHome3" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML lang="en">
  <HEAD>
    <title>Home</title>
    <% @Register TagPrefix="uc" TagName="topnavigation"
        src="UserControlsNavigationTop.ascx" %>
    <% @Register TagPrefix="uc" TagName="dwnnavigation"
        src="UserControlsNavigationDwn.ascx" %>
  </HEAD>
  <body>
    <table border="2">
      <tr>
        <td><h1>LOGO</h1>
        </td>
        <td>
          <uc:topnavigation id="topnav" runat="server" />
        </td>
      </tr>
      <tr>
        <td>
          <uc:dwnnavigation id="dwnnav" runat="server" />
        </td>
        <td>
          <asp:label id="show" runat="server" />
        </td>
      </tr>
    </table>
  </body>
</HTML>
```

Listing 7.7: Zwei interaktiv gesteuerte Benutzer-Steuerelemente
(*UserControlsHome3.ascx*)

Die Steuerung erfolgt hier über einen GET-Parameter. Dies dient mehr der Demonstration als der praktischen Nutzung, denn Sie

können so nicht problemlos in Formularen arbeiten. Es gibt aber viele Wege, den Auswahlparameter zu übertragen, beispielsweise über Sitzungsvariablen oder im Anzeigestatus registrierte Werte. Der folgende Code zeigt, was hinter den Kulissen passiert:

```
Public Class UserControlHome3
    Inherits System.Web.UI.Page
    Protected show As Label
    Protected topnav As UserControlNavigationTopDown
    Protected dwnnav As UserControlNavigationTopDown

    Private Sub Page_Load(ByVal sender As System.Object, 
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Select Case Convert.ToInt32(Request.QueryString("i"))
            Case 2
                topnav.nav_link2 = True
                dwnnav.nav_link2 = True
                show.Text = "<h1>Impressum</h1>"
            Case 3
                topnav.nav_link3 = True
                dwnnav.nav_link3 = True
                show.Text = "<h1>Produkte</h1>"
            Case Else
                topnav.nav_link1 = True
                dwnnav.nav_link1 = True
                show.Text = "<h1>Startseite</h1>"
        End Select
    End Sub
End Class
```

Listing 7.8: Steuerung der Anzeige durch GET-Parameter (UserControlsHome3.ascx.vb)

Die folgende Abbildung zeigt, wie die fertige Seite aussieht. Die um die Navigation erleichterte Seite ist nicht nur schlanker und übersichtlicher, sondern auch einfacher zu warten. Änderungen an der Navigation wirken sich automatisch an allen Stellen aus, an denen sie verwendet wird. Dies kann auch bei kleineren Projekten erheblich Zeit sparen und die Qualität der Seiten verbessern. Durch die Trennung des Codes sind überdies auch die Änderungen selbst einfacher ausführbar.

Aus den mit festen Links versehenen Steuerelementen lassen sich leicht universelle Elemente entwickeln. Die Tags in den bereits gezeigten Beispielen sind fest codiert. Da Sie aus jedem HTML-Element ein HTML Server-Steuerelement machen können und die Links ohnehin bereits Web Server-Steuerelemente sind, bietet sich

Benutzer-Steuerelemente perfektionieren

eine Erweiterung der Liste über Eigenschaften an. Hier kann über ein Formular die Navigation dynamisch erweitert werden, ähnlich den persönlichen Menüs in MS Office. Tatsächlich verursacht der Einbau der erweiterten Benutzer-Steuerelemente kaum Aufwand. Entsprechend sind Änderungen – auch solche tief gehenden wie bei der Navigation – bei geschickter Planung der Applikation sehr einfach ausführbar. Webseiten sind dynamische Gebilde. Der Erfolg einer Site hängt wesentlich davon ab, wie schnell und flexibel sich diese an ändernde Benutzergewohnheiten anpassen lassen. Eine der aktuellen Entwicklungen sind so genannte My-Sites, bei denen die Gestaltung der Seite und die Anzahl verfügbarer Optionen an die Wünsche eines Benutzers angepasst werden kann. Benutzer-Steuerelemente sind das ideale Werkzeug dafür. Wie beim Navigationsbeispiel gezeigt, können Sie leicht jedes veränderbare Teil der Seite damit definieren. Die vom Benutzer wählbaren Eigenschaften werden auch programmtechnisch als Eigenschaften abgelegt. Es ist nun möglich, jedem Steuerelement die Möglichkeit zu geben, aus der Session-ID den aktuellen Benutzer zu ermitteln und »sich selbst« so einzustellen, wie es beispielsweise in einer Datenbank hinterlegt wurde. Der Programmierer der Seite muss dann nicht auf die Belange der Benutzer-Steuerelemente Rücksicht nehmen, sondern kann diese wie Web Server-Steuerelemente einsetzen. Die Sammlung solcher universeller Steuerelemente ist eine ideale Basis für eine Bibliothek. Wie in Bibliotheken üblich, werden nur Teile »ausgeliehen«.

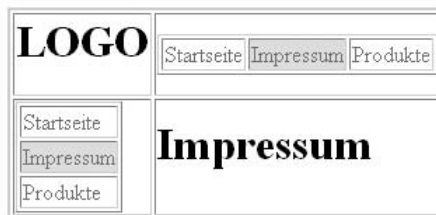


Abbildung 7.5: Benutzer-Steuerelemente, die denselben Code nutzen

Dynamische Benutzer-Steuer- elemente

In den vorherigen Beispielen wurden die dynamisch veränderlichen Navigations-Elemente selbst statisch implementiert. Es wäre im Sinne einer guten Benutzerführung aber durchaus praktisch, wenn die An- und Abwahl programm- und damit benutzer-gesteuert erfolgen kann. Die bereits mit HTML Server- und Web Server-Steuerelemente verwendeten Techniken lassen sich mit Be-

nutzer-Steuerelementen gut kombinieren. Die folgende Datei zeigt die Nutzung einer direkt von der Klasse `UserController` geerbten Eigenschaft – `Visible`. Damit lassen sich die Benutzer-Steuerelemente gezielt ein- und ausschalten. Das Formular dient der Steuerung der Navigationsanzeige durch den Benutzer:

```
<%@ Page language="vb" CodeBehind="UserControlsHome4.aspx.vb"
    AutoEventWireup="false"
    Inherits="Addison.VBNet.WebForm.UserControlsHome4" %>
<% @Register TagPrefix="uc" TagName="topnavigation"
    src="navigation_top.ascx" %>
<% @Register TagPrefix="uc" TagName="dwnnavigation"
    src="navigation_dwn.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
  <head>
    <title>Home</title>
  </head>
  <body>
    <table border="2">
      <tr>
        <td><h1>LOGO</h1></td>
        <td>
          <uc:topnavigation id="topnav" runat="server" />
        </td>
      </tr>
      <tr>
        <td>
          <uc:dwnnavigation id="dwnnav" runat="server" />
        </td>
        <td>
          <asp:label id="show" runat="server"/>
        </td>
      </tr>
    </table>
    <form runat="server">
      <asp:checkbox autopostback="true" id="settop"
        runat="server" text="Top Navigation"
        groupname="nav_set"/>
      <asp:checkbox autopostback="true" id="setdwn"
        runat="server" text="Down Navigation"
        groupname="nav_set"/>
    </form>
  </body>
</html>
```

Listing 7.9: Interaktive Aktivierung und Deaktivierung von Benutzer-Steuerelementen (`UserControlsHome4.aspx`)

In der Methode `Page_Load` wird dann die Veränderung der Einstellung abgefragt und die Sichtbarkeit der Navigationselemente gesteuert. *topnav* und *dwnnav* sind die Instanzen der Benutzer-Steuerelemente, *settop* und *setdown* die Web Server-Steuerelemente, mit denen die Anzeige gesteuert wird. Es handelt sich hierbei um die Kontrollkästchen.

```
Public Class UserControlHome4
    Inherits System.Web.UI.Page
    Protected WithEvents show As Label
    Protected WithEvents settop As CheckBox
    Protected WithEvents setdown As CheckBox
    Protected topnav As UserControlNavigationAdmin
    Protected dwnnav As UserControlNavigationAdmin

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Select Case Convert.ToInt32(Request.QueryString("i"))
            Case 2
                topnav.nav_link2 = True
                dwnnav.nav_link2 = True
                show.Text = "<h1>Impressum</h1>"
            Case 3
                topnav.nav_link3 = True
                dwnnav.nav_link3 = True
                show.Text = "<h1>Produkte</h1>"
            Case Else
                topnav.nav_link1 = True
                dwnnav.nav_link1 = True
                show.Text = "<h1>Startseite</h1>"
        End Select
        If Page.IsPostBack Then
            If settop.Checked Then
                topnav.Visible = True
            Else
                topnav.Visible = False
            End If
            If setdown.Checked Then
                dwnnav.Visible = True
            Else
                dwnnav.Visible = False
            End If
        Else
            settop.Checked = True
        End If
    End Sub
End Class
```



```

        setdwn.Checked = True
    End If
End Sub

```

```
End Class
```

Listing 7.10: Steuerung der Anzeige der Steuerelement (*UserControlsHome4.aspx.vb*)

Abbildung 7.6: Formular zum Ein- und Ausschalten von Benutzer-Steuerelementen

Steuerung von Eigenschaften der Benutzer-Steuerelemente

Das nächste Beispiel zeigt eine zusätzliche Erweiterung, bei der über ein drittes Kontrollkästchen dynamisch ein weiterer Link angezeigt werden kann. Neben der Definition der Tabellenzelle bzw. Tabellenreihe wird der Code aus dem letzten Beispiel um die Steuerung der entsprechenden Eigenschaft dieser Zelle erweitert. Startpunkt des Beispiels ist die Datei *UserControlsHome5.aspx*. Darin ist ein weiteres Kontrollkästchen zu finden:

**Code Behind
effektiver nutzen**

```

<asp:checkbox autopostback="true" id="setadm" 
           runat="server" text="Admin Area" 
groupname="nav_set" />

```

Mit diesem Kontrollkästchen kann ein zusätzlicher Link aktiviert werden. In der Code-Datei *UserControlsHome4.aspx.vb* wird – gegenüber dem letzten Beispiel – Folgendes ergänzt:

```

If (setadm.Checked) Then
    topnav.nav_link4 = True
    dwonav.nav_link4 = True
Else
    topnav.nav_link4 = False
    dwonav.nav_link4 = False
End If

```

Die beiden Benutzer-Steuerelemente unterscheiden sich nur durch die Anordnung der Links in der Tabelle und dies entspricht, abgesehen vom vierten Link, den bereits gezeigten Beispielen. Ebenso erweitert worden ist die Code-Datei, die sich aber funktional nicht unterscheidet.

Die Abbildung zeigt den Effekt, wenn der zusätzliche Link eingeschaltet ist:



Abbildung 7.7: Komplexes Benutzer-Steuerelement, interaktiv erweiter- und aktivierbar

Auch den Code dieses Steuerelements könnte man noch auslagern, was in der Praxis auch zu empfehlen ist. Wenn Sie Visual Studio .NET einsetzen, wird diese Form grundsätzlich verwendet, wenn Sie eine neue Vorlage für Benutzer-Steuerelemente auswählen.

7.3 Browserorientiert programmieren

»Uplevel«
»Downlevel«

Grundsätzlich unterscheidet ASP.NET zwischen zwei Arten von Browsern: »Uplevel« und »Downlevel«. Welcher Typ erkannt oder verlangt wird, entscheidet darüber, wie die Steuerelemente erstellt werden. Dabei wird als »Uplevel« ein Browser definiert, der in etwa dem Internet Explorer entspricht. Dennoch können auch andere Browser verwendet werden, wenn sie die folgenden Eigenschaften haben:

- ECMAScript ab Version 1.2 (auch bekannt unter den Marken JavaScript und JScript)
- HTML Version 4.0 wird unterstützt.
- Das DOM nach der Definition von Microsoft wird unterstützt.
- Stylesheets nach CSS 1.0, die zusätzlich mindestens die absolute Positionierung aus CSS 2.0 erlauben

Der Internet Explorer verfügt über eine ganze Reihe von Funktionen, die bei der Einstellung »Uplevel« benutzt werden und die nicht mit HTML 4.0 oder CSS 2.0 konform sind. Dies betrifft insbesondere:

- ▶ Das Attribut `accesskey`, das eine Taste zur Anwahl eines Formularfeldes erlaubt
- ▶ Das Attribut `tabindex` zur Festlegung der Reihenfolge der Felder beim Durchlaufen mit der `[Tab]`-Taste
- ▶ Das Attribut `tooltip`, das ein kleines gelbes Pop-Up-Fenster erzeugt

Wenn Sie diese Funktionen benutzen, kann es mit Browsern zu Problemen mit der Darstellung kommen, auch wenn diese als »Uplevel« erkannt wurden.

Prinzipiell schränkt die Verwendung von Downlevel-Browsern die Nutzung einiger gestalterischer Funktionen ein. Probleme gibt es unter anderem bei folgenden häufig verwendeten Eigenschaften:

- ▶ `BackColor`, `Width` und `BorderWidth` werden nur dann korrekt ausgeführt, wenn das betreffende Steuerelement mit einer HTML-Tabelle dargestellt wird. Fließende Layouts (Flow-Layout) ignorieren die Angaben. Die Verwendung von `Height` wird für manche Steuerelemente ebenfalls ignoriert.
- ▶ `BorderStyle`, `Font-OverLine`, werden generell nicht unterstützt.
- ▶ `Fore-Color` wird als `<font>`-Tag umgesetzt, nicht als Stylesheet.

7.3.1 Umgang mit Browsern im Zusammenhang mit Clientskripten

Wenn anhand der Signatur eines Browsers erkannt wird, dass dieser in der Lage ist Clientskripte mit JavaScript auszuführen, wird ASP.NET die entsprechenden Skriptbibliotheken verwenden. Das Erkennen der Browserfunktionen basiert jedoch auf den prinzipiellen Möglichkeiten des Browsers. Wenn der Benutzer die Funktion abschaltet, kann dies nicht sicher automatisch erkannt werden. Sie sollten daran denken, dass dieser Fall eintreten kann. Folgende Steuerelemente und Funktionen werden nicht funktionieren, wenn JavaScript nicht aktiviert oder verfügbar ist:

Clientskripte mit JavaScript

- ▶ `LinkButton` und `HtmlButton` benötigen JavaScript, da sie den Klick auf den Link in ein »Form-Submit« wandeln müssen.
- ▶ Die Eigenschaft `AutoPostBack` benötigt generell JavaScript.

Außerdem kann für viele Kontroll-Steuerelemente eine clientseitige Prüfung definiert werden, die natürlich auch auf eine Skriptumgebung angewiesen ist. Dies ist aber relativ unkritisch, wenn Sie sich an die Fallback-Regeln halten und eine äquivalente Funktion auch serverseitig bereitstellen.



Die Prüf- und Formularfunktionen benötigen lediglich ECMAScript, nicht jedoch das DOM (Document Object Model) des Browsers und DHTML. Sie laufen zuverlässig auch auf Opera und Netscape. Lediglich für einige gestalterische Funktionen ist DHTML hilfreich, jedoch nicht zwingend notwendig (Im Klartext: Es sieht nicht gut aus, aber es funktioniert).

7.3.2 Feststellen des Browsertyps und seiner -funktion

Fähigkeiten des Browsers feststellen

Basis für den Umgang mit Browser-Informationen bildet die Klasse `HttpBrowserCapabilities`. Sie müssen zuerst eine Instanz aus dem `Request`-Objekt der aktuellen Anforderung mit der Eigenschaft `Browser` erstellen:

```
Dim browser As HttpBrowserCapabilities = Request.Browser
```

Sie können nun auf eine Reihe von Eigenschaften zurückgreifen, die über den Browser Auskunft geben. Das folgende Programm erstellt eine Liste einiger Eigenschaften. Die Darstellung basiert auf einem `<ul>`-Tag, das dynamisch erweitert wird:

```
<h1>Browseereigenschaften</h1>
Der Browser kennt folgende Funktionen:<br/>
<ul Runat="server" ID="Capabilities"/>
```

Listing 7.11: Vorbereitung der Liste (Ausschnitt aus `BrowserCapabilities.aspx`)

Die Liste wird nun durch dynamisches Hinzufügen von `<li>`-Tags erstellt. Das Layout bleibt unverändert.

```
Public Class BrowserCapabilities
    Inherits System.Web.UI.Page
    Protected Capabilities As HtmlGenericControl
    Private HtmlListElement As HtmlGenericControl
```

```

Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles MyBase.Load
    Dim CurrentBrowser As HttpBrowserCapabilities 
        = Request.Browser
    HtmlListElement = New HtmlGenericControl("li")
    HtmlListElement.InnerText = "Browser = "
    HtmlListElement.InnerText += CurrentBrowser.Browser
    HtmlListElement.InnerText += CurrentBrowser.Version
    Capabilities.Controls.Add(HtmlListElement)
    HtmlListElement = New HtmlGenericControl("li")
    HtmlListElement.InnerText = "JavaScript Version = "
    HtmlListElement.InnerText += String.Format("{0}.{1}", ,
        CurrentBrowser.EcmaScriptVersion.Major.ToString(), ,
        CurrentBrowser.EcmaScriptVersion.Minor.ToString())
    Capabilities.Controls.Add(HtmlListElement)
    HtmlListElement = New HtmlGenericControl("li")
    HtmlListElement.InnerText = "BetaVersion = "
    If CurrentBrowser.Beta = True Then
        HtmlListElement.InnerText += " JA "
    Else
        HtmlListElement.InnerText += " NEIN "
    End If
    Capabilities.Controls.Add(HtmlListElement)
    HtmlListElement = New HtmlGenericControl("li")
    HtmlListElement.InnerText = ".NET CLR-Version = "
    HtmlListElement.InnerText +=
CurrentBrowser.ClrVersion.ToString()
    Capabilities.Controls.Add(HtmlListElement)
End Sub
End Class

```

Listing 7.12: Ermittlung der Browsereigenschaften (Ausschnitt aus BrowserCapabilites.aspx.vb)

Zuerst erfolgt die Anforderung der Browserdaten:

```
Dim CurrentBrowser As HttpBrowserCapabilities = Request.Browser
```

Dann wird ein neues -Tag erzeugt:

```
HtmlListElement = New HtmlGenericControl("li")
```

Diesem werden dann über die Eigenschaft InnerText die Daten zugewiesen und anschließend wird das fertige Tag an die -Kollektion angefügt:

```
Capabilities.Controls.Add(HtmlListElement)
```

**Wie es
funktioniert**

Beachten Sie, dass Boolesche oder numerische Werte mit `ToString` in Zeichenketten konvertiert werden müssen. Die Ausgabe zeigt, wie aktuelle Browser erkannt werden:



Abbildung 7.8: Ausgabe des Programms für Internet Explorer 6.0 (oben) und Netscape 6.2 (unten)

Wie die Informationen tatsächlich intern beschafft werden, zeigt der folgende Abschnitt.

7.3.3 Interne Arbeitsweise der Browsererkennung

ASP.NET im Vergleich zu ASP

Im alten ASP wurde eine Datei benutzt, die zu allen bekannten Browserkennungen eine Liste der verfügbaren Funktionen lieferte. Dies ist notwendig, weil der Browser selbst nicht alle Informationen preisgibt, die für die Gestaltung von Seiten gebraucht werden.

In ASP.NET ist das Prinzip nur ein wenig modifiziert worden. Nachwievor sind die eigentlichen Daten über die Browser fest hinterlegt. Allerdings ist die Art der Erkennung verbessert worden. Der Browser liefert, neben anderen Informationen, eine bestimmte Zeichenkette, die ihn identifiziert:

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)

Sie können diese Zeichenkette sehr einfach in einer *aspx*-Seite ermitteln:

```
<% = Request.ServerVariables("HTTP_USER_AGENT") %>
```

ASP.NET versucht nun beim Abruf des `HttpBrowserCapabilities`-Objekts aus diesen Daten und weiteren Servervariablen durch Mustervergleiche mit regulären Ausdrücken die tatsächlichen Bedingungen auf dem Client zu ermitteln.

Beachten Sie, dass die Methode generell unzuverlässig ist, weil die Identifizierungszeichenkette im Browser von fachkundigen Benutzern gefälscht werden kann. Allerdings ist zu vermuten, dass derartige Manipulationen nur dann ausgeführt werden, wenn der Empfang normaler Webseiten mit vollem Funktionsumfang ohnehin nicht geplant ist.



Die Datenbasis in der Datei »machine.config«

Die Bedeutung und Lage der Datei *machine.config* wird in Abschnitt 10.4, »Konfiguration des gesamten Systems machine.config« ab Seite 966 vorgestellt. Sie finden in dieser Datei im Abschnitt `<system.web>` den Eintrag `<browserCap>`. Dort sind Filter definiert, die auf bestimmte Einträge in `HTTP_USER_AGENT` reagieren. Die Filter bestehen aus regulären Ausdrücken und können so leicht erweitert oder modifiziert werden. Allerdings wird dies nur selten notwendig sein. Gegenüber der starren Verknüpfung in der alten *browscap.ini* ist diese Variante sehr zuverlässig bei zukünftigen Änderungen der Erkennungszeichenfolge.

machine.config

Als Beispiel sei die Erkennung des Internet Explorers angeführt:

```
<case match="^Mozilla[^(]*\ (compatible; MSIE
(? 'version' (? 'major' \d+ ) (? 'minor' \
.\d+ ) (? 'letters' \w* ) ) (? 'extra' .* ) ">
    browser=IE
    version=${version}
    majorversion=${major}
    minorversion=${minor}
<case match="[5-9]\.\" with="${version}">
    frames=true
    tables=true
    cookies=true
    backgroundsounds=true
    vbscript=true
    javascript=true
```

```

        javaapplets=true
       activexcontrols=true
        tagwriter=System.Web.UI.HtmlTextWriter
        ecmaascriptversion=1.2
        msdomversion=${major}${minor}
        w3cdomversion=1.0
        css1=true
        css2=true
        xml=true
        <filter with="{letters}" match="^b">
            beta=true
        </filter>
        <filter with="{extra}" match="Crawler">
            crawler=true
        </filter>
    </case>
    ...

```

Das erste Tag `<case match=" ">` erkennt die Ausgabe von `HTTP_USER_AGENT`, alle folgenden Zuweisungen oder weitere Tests von Teilmustern können auch auf die erkannten Gruppen zugreifen, die in Variablen vom Typ `$(name)` abgelegt sind.

Einsatzmöglichkeiten

- Applikationen** Wenn Sie Applikationen entwickeln, werden Sie verschiedene Funktionen möglicherweise nur für bestimmte Clients bereit stellen wollen. Es wäre nun denkbar, ein Programm so universell zu schreiben, dass es sich auf die Angaben der Browserbedingungen einstellt. Wenn Sie die Datei *machine.config* um intelligente Muster erweitern, können Sie auch auf eigene Clients reagieren. So wäre es möglich, Außendienstmitarbeitern einen speziellen Browser zu geben, beispielsweise unter Windows CE. Wird dieser erkannt, schaltet die Anwendung in eine andere Darstellform um oder erlaubt den Zugriff auf bestimmte Daten.
- Sicherheit** Aus Sicht der Systemsicherheit ist dies nur ein schwacher, aber zusätzlicher Schutz und das Verfahren trägt damit insgesamt zur Verbesserung der Sicherheit bei.

7.4 SmartNavigation

Bislang wurde immer betont, dass die von Server-Steuerelementen erzeugten Codes HTML-konform sind und mit jedem Browser

angezeigt werden können. Wenn Sie jedoch sicherstellen können, dass Ihre Benutzer den Internet Explorer einsetzen, verzichten Sie auf eine Reihe von interessanten Eigenschaften. Die DHTML-Funktionen des Internet Explorers erlauben unter anderem eine spezielle Methode zur Aktualisierung von Inhalten der Seite. Normalerweise löscht der Browser die Seite, wenn Sie ein Formular absenden, wartet dann auf den Rücklauf vom Server und baut die Seite neu auf. Auch wenn der Vorgang sehr schnell stattfindet, bleibt der Effekt des Seitenab- und -aufbaus sichtbar. Darüber hinaus gehen Einstellungen des Benutzers bezüglich des angezeigten Formulars verloren. Das ist besonders dann sichtbar, wenn sich an der Seite nur wenig ändert.

Die Funktion SmartNavigation benötigt den Internet Explorer ab Version 5.



7.4.1 Einsatzmöglichkeiten der Eigenschaft SmartNavigation

Es gibt einige typische Probleme bei der Seitendarstellung, die von Windows-Anwendungen her nicht bekannt sind. Benutzer, die den Browser für die Bedienung einer Applikation nutzen sollen und wenig Erfahrung im Umgang mit Browsern haben, werden davon irritiert sein. Wenn Sie festlegen können, dass der Internet Explorer verwendet wird, lassen sich die folgenden Effekte vermeiden:

**Lösung für
typische Darstel-
lungsprobleme**

- ▶ Der sichtbare Seitenab- und -aufbau, spürbar als Flackern.
- ▶ Seiten mit Scrollbalken vergessen beim erneuten Laden des Formulars ihre Position.
- ▶ Eingabefelder verfügen über einen so genannten Fokus zu Markierung des aktuellen Feldes. Dieser Fokus geht beim erneuten Laden verloren.
- ▶ Wenn die Seite immer wieder geladen wird, füllt sich der »History«-Speicher mit den Seiteninformationen. Wenn der Benutzer die Zurück-Schaltfläche im Browser betätigt, wird das nicht erwartungsgemäß funktionieren. Stattdessen führt dieser Aufruf nur zu einem der vorherigen Formularaufrufe und der sehr lästigen Meldung, dass die Seite nicht mehr aktuell ist.

Der sinnvolle Einsatz beschränkt sich natürlich darauf, wenige Inhalte der Seite elegant auszutauschen. Wenn sich der Aufbau der Seite nach dem Absenden des Formulars grundlegend ändert, hilft die SmartNavigation-Funktion natürlich nicht.



Perfekt ist die Umsetzung allerdings nicht. Die Festsetzung des Fokus funktioniert nicht, wenn der Benutzer die Felder mit der `Tab`-Taste wechselt.

7.4.2 Aktivierung und praktische Nutzung

Für die praktische Nutzung ist lediglich die entsprechende Funktion zu aktivieren. Dafür gibt es zwei praktikable Wege, die nachfolgend beschrieben werden. Die dritte Möglichkeit ist die Aktivierung im Code.

SmartNavigation global aktivieren

Wenn Sie festlegen möchten, die *SmartNavigation* für das gesamte Projekt zu verwenden, tragen Sie die entsprechende Option in die Konfigurationsdatei *web.config* folgendermaßen ein:

```
<configuration>
  <system.web>
    <pages SmartNavigation="true"/>
    ...
  </system.web>
</configuration>
```

SmartNavigation seitenweise aktivieren

Generell ist die Idee, *SmartNavigation* einzusetzen, an eine ganze *aspx*-Seite gebunden. Sie können dies nicht für einen Teil abbilden, wie beispielsweise ein einzelnes Benutzer-Steuerelement. Der einfachste Weg, *SmartNavigation* zu aktivieren, besteht deshalb im Setzen einer entsprechenden Option der `@Page`-Direktive:

```
<% @Page SmartNavigation="true" ... %>
```

SmartNavigation praktisch nutzen

Der Erhalt der Position des Scrollbalkens mag kein hinreichender Grund sein, *SmartNavigation* einzusetzen. Es gibt jedoch im Zusammenhang mit dem Erhalt des Fokus eine clevere Anwendung. Wenn Sie für viele Felder eine sehr strenge Feldprüfung vornehmen müssen oder nachfolgende Felder in Abhängigkeit von vorherigen Eingaben setzen möchten, ist ein zwischenzeitliches Senden des Formulars zum Server eine sinnvolle Möglichkeit. Dann wäre es allerdings fatal, wenn der Benutzer jedes Mal wieder das letzte Feld mit der Maus oder der `[Tab]`-Taste aufsuchen müsste.

Bei Listen mit automatischem Absenden (*AutoPostBack*) wird dieses Verhalten besonders deutlich. Wenn Sie zwei Listen präsentieren, die in Abhängigkeit voneinander stehen, ist der Erhalt des Fokus ein ganz entscheidender Punkt in der Benutzerführung.

AutoPostBack

Die folgende Abbildung zeigt die Idee. Die linke Liste enthält Regionen der Erde. Die rechte enthält Länder, wobei jeweils nur die angezeigt werden, die zur ausgewählten Region passen.

EUROPE	ALBANIA
EUROPE	
AFRICA	
ASIA AND PACIFIC	
LATIN AMERICA AND CARRIBBEAN	

Abbildung 7.9: Die rechte Liste enthält Ländernamen in Abhängigkeit vom Zustand der linken.

Wenn Sie einen solchen Entwurf mit ASP.NET umsetzen, können Sie die linke Liste mit der Maus anwählen und mit dem Auslösen der Option wird die rechte Liste aktualisiert. Den Quelltext für diesen Effekt finden Sie im folgenden Beispiel:

Wählen Sie ein Region aus:

```
<form runat="server">
    <asp:DropDownList Runat="server" ID="region"
        AutoPostBack="True" />
    <asp:DropDownList Runat="server" ID="country"
        Width="350px" />
    <asp:Button Runat="server" ID="select"
        OnClick="SelectCountry" Text="Auswahl" />
</form>
```

Ihre Wahl: <asp:Label Runat="server" ID="selectedCountry" />

Listing 7.13: Aufbau von dynamischen Liste und optimierte Navigation mit *SmartNavigation* (Ausschnitt *SmartNavigation.aspx*)

Das setzt jedoch voraus, dass das entsprechende Feld immer mit der Maus angewählt wird. Nun kann es bei der Suche nach einem Land sinnvoll sein, die linke Liste zügig zu durchblättern. Vermutlich wird der Benutzer dann nicht die Maus verwenden, sondern die Liste auswählen und nun mit der -Taste die Liste durchlaufen. Bei jedem Antippen der Taste wird das Formular gesendet:

```
<asp:DropDownList Runat="server" ID="region" AutoPostBack="True"/>
```

Ohne *SmartNavigation* geht der Fokus nun sofort verloren, das heißt, ein Durchblättern der Liste ist eigentlich unmöglich. Wenn Sie dagegen mit *SmartNavigation* arbeiten, bleibt der Fokus in der Liste und der Blättereffekt wird möglich.

Ergänzend sei an dieser Stelle eine kleine Applikation präsentiert, die den *SmartNavigation*-Effekt mit einer ansprechenden Datenbindung kombiniert. Die Liste der Regionen und Länder wird dazu in einer XML-Datei gespeichert:

```
<?xml version="1.0" encoding="utf-8" ?>
<regions xmlns="http://tempuri.org/regions.xsd">
  <region name="EUROPE">
    <country>ALBANIA</country>
    <country>BELGIUM</country>
    <country>BOSNIA AND HERZEGOVINA</country>
```

Listing 7.14: Die Datenquelle ist eine kleine XML-Datei mit den Regionen und Ländern (Ausschnitt aus data/regions.xml).

Datenbindung von Listen

Mit der automatischen Erstellung eines XML-Schemas lässt sich die Übertragung in ein *DataSet*-Objekt sehr leicht erledigen. Das XML-Schema liegt bereits fertig im Verzeichnis *data* als *regions.xsd*. Wenn Sie es selbst erstellen möchten und über Visual Studio .NET verfügen, wählen Sie in der XML-Ansicht im Menü XML die Option SCHEMA ERSTELLEN. Das Schema hilft Ihnen beim Entwurf solcher XML-Quellen und bei der Zuordnung der Daten zu Tabellen. Aus der Darstellung des *DataSet* mit Hilfe des Schemas ergeben sich auch die generierten Feldnamen, die in der folgenden Applikation zum Einsatz kommen:

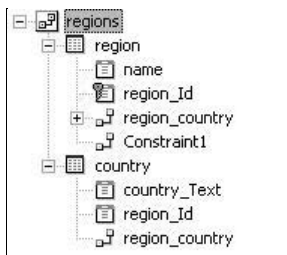


Abbildung 7.10: Struktur des aus den XML-Daten gewonnenen DataSet-Objekts

Die komplette Applikation zeigt, wie die Listen mit Daten gefüllt werden:

```

Public Class SmartNavigation
    Inherits System.Web.UI.Page
    Protected region As DropDownList
    Protected WithEvents country As DropDownList
    Protected WithEvents selectedCountry As Label
    Private currentIndex As Integer

    Protected Sub SelectCountry(ByVal sender As Object,   

                                ByVal e As System.EventArgs)
        selectedCountry.Text = "Bitte wählen Sie ein Land"
        If Not country.SelectedItem Is Nothing Then
            country.SelectedIndex = currentIndex
            selectedCountry.Text = country.SelectedItem.Value
        End If
    End Sub

    Private Sub Page_Load(ByVal sender As Object,   

                          ByVal e As System.EventArgs) _
    Handles MyBase.Load
        Dim dsRegion As DataSet = New DataSet()
        dsRegion.ReadXml(Server.MapPath("data/regions.xml"))
        If Not Page.IsPostBack Then
            region.DataSource = dsRegion
            region.DataMember = "region"
            region.DataTextField = "name"
            region.DataValueField = "region_Id"
            region.DataBind()
        Else
            currentIndex = country.SelectedIndex
            Dim dvCountry As DataView = New DataView()
            dvCountry.Table = dsRegion.Tables("country")
            dvCountry.RowFilter = String.Format("region_Id='{0}'",   

                                                region.SelectedItem.Value)
        End If
    End Sub
End Class

```

```

country.DataSource = dvCountry
country.DataMember = "country"
country.DataTextField = "country_Text"
country.DataValueField = "country_Text"
country.DataBind()
End If
End Sub
End Class

```

Listing 7.15: Ereignisgesteuerte Verarbeitung der Listen (*SmartNavigation.aspx.vb*)

Wie es funktioniert

Die Basis der Datenhaltung bildet ein `DataSet`-Objekt:

```
Dim dsRegion As DataSet = New DataSet()
```

Beim Start werden zuerst die Daten aus der XML-Datei in ein `DataSet`-Objekt eingelesen:

```
dsRegion.ReadXml(Server.MapPath("data/regions.xml"))
```

Wird die Seite das erste Mal aufgerufen, wird nun die Liste `region` gefüllt. Für alle folgenden Formularaufrufe sorgt der Anzeigestatus (*ViewState*) für die Erhaltung des Listeninhalts, sodass die Zuweisungen beim *PostBack* nicht mehr erfolgen müssen:

```
If Not Page.IsPostBack Then
```

Das Füllen der Liste geschieht durch Zuweisung der Datenquelle:

```
region.DataSource = dsRegion
```

Die Verknüpfung mit der Ländertabelle *country* wird über das Feld *region_Id* programmiert. Wie die Felder heißen, können Sie Abbildung 7.10 entnehmen. Die Anwendung sieht folgendermaßen aus:

```
region.DataValueField = "region_Id"
```

Wenn das Formular gesendet wird – zuerst durch die Wahl der Region, ausgelöst durch die *AutoPostBack*-Funktion – muss nun die zweite Liste in Abhängigkeit von der Auswahl aufgebaut werden. Außerdem muss der gewählte Status erhalten bleiben:

```
currentIndex = country.SelectedIndex
```

Da die Tabelle *country*, die die Ländernamen enthält, nun gefiltert werden muss, wird die Klasse `DataView` verwendet:

```
Dim dvCountry As DataView = New DataView()
```

Die Sicht beschränkt sich auf die Tabelle *country*:

```
dvCountry.Table = dsRegion.Tables("country")
```

Auf diese Tabelle wird ein Filter definiert, der der Auswahl der ersten Liste entspricht:

```
dvCountry.RowFilter ←  
    = String.Format ("region_Id='{0}'", ←  
        region.SelectedItem.Value)
```

Es folgen im Code die Zuweisungen der gefilterten Daten zum DropDownList-Steuerelement *country*.

Damit man mit den Daten auch noch etwas anfangen kann, ist an die Schaltfläche ein Ereignis gebunden, das die Methode *SelectCountry* auslöst:

```
<asp:Button Runat="server" ID="select" ←  
    OnClick="SelectCountry" Text="Auswahl" />
```

In der Ereignisbehandlungsmethode erfolgt dann der Zugriff auf den mit der Auswahl getroffenen Index und die Zuweisung an die Liste. Das ist notwendig, weil das wiederholte Aufbauen der Datenliste zum Verlust des Anzeigestatus führt:

```
country.SelectedIndex = currentIndex
```

Zur Kontrolle wird dieser Wert dann an das *Label*-Steuerelement übergeben:

```
selectedCountry.Text = country.SelectedItem.Value
```

Im Ergebnis entsteht eine ungewöhnlich gut bedienbare, zweiteilige, datengebundene Auswahl:

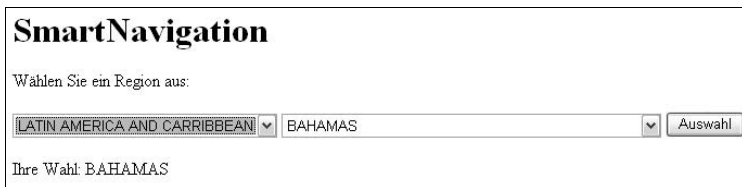


Abbildung 7.11: Auswahl aus größeren Listen trotz *AutoPostBack* allein mit der *Cursor-Taste* (linke Liste)

7.5 Zusätzliche Steuerelemente: WebControls 1.0

Beschaffung Zu ASP.NET gibt es eine Erweiterung um einige häufig benötigte komplexe Steuerelemente, WebControls 1.0 genannt. Diese sind als zusätzliches Modul unter folgender Adresse ladbar:

http://msdn.microsoft.com/downloads/default.asp?url=/downloads/samples/internet/asp_dot_net_servercontrols/webcontrols/default.asp

Einsatz Insgesamt besteht das Paket aus folgenden Steuerelementen:

► MultiPage

Dieses Element dient der überlagerten Darstellung mehrerer Seiten im Browser, sodass zwischen den Seiten sehr schnell umgeschaltet werden kann. Darüber hinaus sind alle Seiten in derselben *aspx*-Datei gespeichert.

► TabStrip

Das TabStrip-Steuerelement ermöglicht die Darstellung von Registerkarten im Browser.

► Toolbar

Dieses Steuerelement dient der Darstellung einer Leiste von Schaltflächen.

► TreeView

Mit Hilfe des Steuerelements kann eine hierarchische Datenmenge in einer Baumstruktur abgebildet werden. Dies eignet sich besonders für die Navigation.

Der Fokus der Steuerelemente ist klar erkennbar. Komplexere, für Desktop-Applikationen typische Funktionen werden hier mit Hilfe von HTML und DHTML nachgebildet. Alle Steuerelemente nutzen DHTML, wenn dies möglich ist, und liefern HTML 3.2, wenn der Browser DHTML bzw. JavaScript nicht versteht.



Bei Tests mit Netscape 6.2 erwies sich das Steuerelement TabStrip als schwer handhabbar in Bezug auf die Reaktion auf CSS. Die Anzeige und Nutzung der erstellten Applikation gelingt zwar, allerdings nur mit schwer wiegenden Abstrichen am Design. Auch der Einsatz des Attributes `ClientTarget="Downlevel"` der `@Page`-Direktive brachte keinen Fort-

schritt. Sie sollten, wenn Sie browserunabhängig programmieren müssen, jedes einzelne Element sehr sorgfältig testen und ein möglichst einfaches Design verwenden. Das schlechte Resultat liegt übrigens nicht am Browser; mit dem durch `ClientTarget="Downlevel"` erzeugten Code kam auch der Internet Explorer 6.0 aus dem Konzept und stellt die Seiten verzerrt dar.

Das Modul besteht aus einer Installationsdatei, die die Installation und Registrierung der Komponente im Global Assembly Cache (GAC) erledigt. Dennoch müssen Sie bei der Arbeit mit Visual Studio .NET einen Verweis einrichten, wenn die Erstellung einer Assembly für Ihr Projekt gewünscht wird.

7.5.1 Installation und Anwendungsbeispiel

Die sehr umfangreichen Eigenschaften, Methoden und Ereignisse können Sie der online verfügbaren und mitgelieferten Dokumentation entnehmen. Die vollständige Darstellung wäre eher langweilig. Interessanter ist es, ein Projekt zu entwerfen, das die Komponenten intensiv nutzt. Voraussetzung dafür ist eine einwandfreie Installation.

Installationsanleitung

Damit Sie die Komponenten auch aus Visual Studio .NET verwenden können, gehen Sie folgendermaßen vor:

1. Öffnen Sie in Ihrem Projekt den PROJEKTMAPPEN-EXPLORER und dort Ihr Projekt und darunter den Zweig VERWEISE.
2. Wählen Sie dann im Kontextmenü HINZUFÜGEN.
3. Im folgenden Dialog suchen Sie auf der Registerkarte .NET die Komponente `Microsoft.IE.WebControls` (siehe Abbildung 7.12).
4. Schließen Sie den Dialog mit OK und prüfen Sie den Eintrag in der Verweisliste (siehe Abbildung 7.13).
5. Nun sollte auch Intellisense auf den Eintrag reagieren und die entsprechende Auswahl zulassen (siehe Abbildung 7.14).

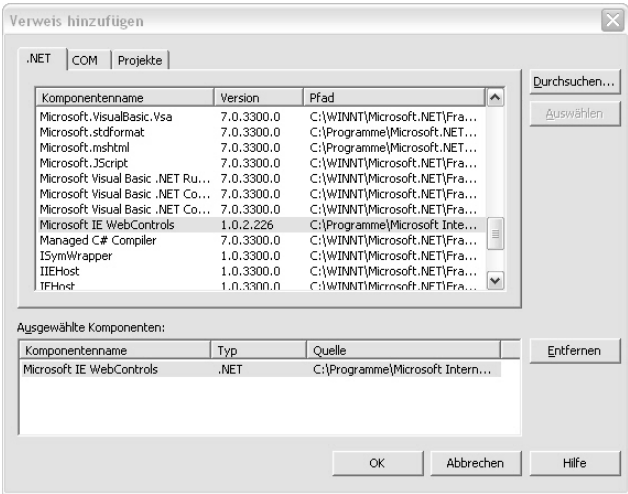


Abbildung 7.12: Auswahl der zusätzlichen Steuerelementebibliothek

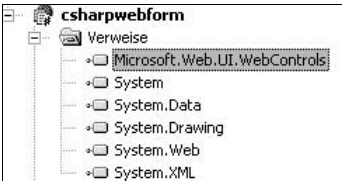


Abbildung 7.13: So erscheint die Komponente in der Verweisliste.

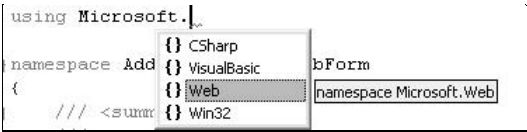


Abbildung 7.14: Intellisense erkennt die neue Komponente im Namensraum Microsoft.

Namensraum einbinden

Der vollständige Namensraum kann wie folgt eingebunden werden:

```
Imports Microsoft.Web.UI.WebControls
```

Registrierung der Komponenten auf der Seite

@Register

Damit Sie die Komponenten tatsächlich verwenden können, müssen Sie diese wie alle anderen Benutzer-Steuerelemente auch auf der Seite mit der @Register-Direktive registrieren. Da alle vier Ele-

mente in derselben Assembly enthalten sind, genügt eine Anweisung:

```
<%@ Register TagPrefix="iewc"
    Namespace="Microsoft.Web.UI.WebControls"
    Assembly="Microsoft.Web.UI.WebControls,
        Version=1.0.2.226,
        Culture=neutral, PublicKeyToken=31bf3856ad364e35" %>
```

Wenn Sie eine neuere Version geladen haben, schauen Sie im GAC nach, was installiert wurde. Geben Sie dazu an der Eingabeaufforderung aus dem Visual Studio .NET-Paket Folgendes ein (das »l« steht für »list«):

```
gacutil -l
```

Suchen Sie den Eintrag in der Liste, der mit »Microsoft.Web« beginnt. Die gesamte Zeile gehört dann in das Assembly-Attribut.

GAC

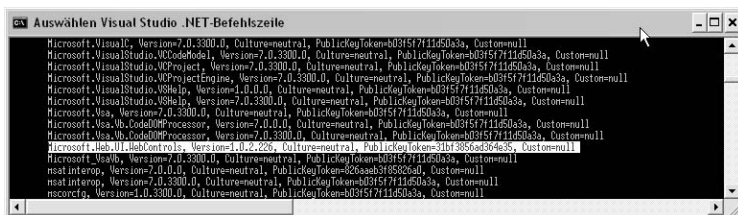


Abbildung 7.15: Ausschnitt aus der Liste, die `gacutil -l` erzeugt

Stellen Sie das Fenster der Eingabeaufforderung auf 300 Zeichen Breite und 500 Zeilen Puffergröße ein, damit Sie die Liste einigermaßen lesbar erhalten.



Durch die Wahl des Parameters des Attributes `TagPrefix` wird das Präfix festgelegt. In allen folgenden Beispielen wird, übereinstimmend mit der Dokumentation, »iewc« verwendet. Das Steuerelement `<TabStrip>` wird deshalb im Quellcode der HTML-Seite `<iewc:TabStrip>` geschrieben.

Das Projekt »Server Explorer«

Wenn Sie Ihre Website bequem verwalten möchten, gibt es mehrere Wege. Sie können per FTP zugreifen, den Ordner als Webordner mit WebDAV öffnen oder sich einer serverseitigen Applikation bedienen. Der zuletzt genannte Weg ist der aufwändigste,

aber auch attraktivste, weil Sie die Art der Bedienung und die Funktionalität frei bestimmen können. Die IE Webcontrols sind eine gute Basis für ein solches Projekt.

Der hier vorgestellte »Server Explorer« ist ein guter Ausgangspunkt für eine richtige Applikation. Es ist kein perfektes Programm, bietet aber eine solide Grundlage dafür. Fehlerfunktionen und robuste Eingabeprüfungen fehlen noch. Es ist eine gute Übung, diese zu ergänzen.



Sie finden das Programm unter dem Namen `IeWebControls.aspx` im Projektordner `vbnetwebform`.

Bei einem solchen Projekt sollten Sie sich zuerst über die Funktionalität im Klaren sein. Für die Navigation innerhalb eines Verzeichnisbaumes wird eine Verzeichnisansicht benötigt. Dann brauchen Sie eine Auswahlmöglichkeit für Dateien und schließlich ein Anzeigemodul dafür. Als Ergänzung dient noch ein Hochlademodul, mit dem Dateien per HTTP auf den Server übertragen werden.

Es ist eine gute Idee, den ersten Entwurf auf einem Blatt Papier zu machen und sich über die Lage und Funktion der Steuerelemente klar zu werden. Das gewünschte Ergebnis zeigen die folgenden Abbildungen. Die Funktionen werden über Registerkarten ausgewählt, die durch `TabStrip` dargestellt werden. Die Applikation besteht nur aus einer einzigen Seite, die einzelnen Teile werden durch `MultiPage` verwaltet und den Registerkarten zugeordnet.

**Verzeichnis
auswählen:
TreeView**

Die Auswahl des Verzeichnisses ist am bequemsten für den Benutzer, wenn er die vom Windows Explorer bekannte Baumstruktur nutzen kann. Das Steuerelement `TreeView` erledigt das in hervorragender Weise.

Toolbar

Mit der Dateiliste kann dann auf die einzelnen Dateien eines Verzeichnisses zugegriffen werden. Die Ausgabe erfolgt mit einem `CheckBoxList`-Steuerelement. Da hier relativ viele Funktionen benötigt werden (auch wenn in der Musterapplikation nicht alle kodiert sind), bietet sich ein `Toolbar`-Steuerelement zur Darstellung von Schaltflächen an.



Abbildung 7.16: Auswahl eines Verzeichnisses aus einem Verzeichnisbaum

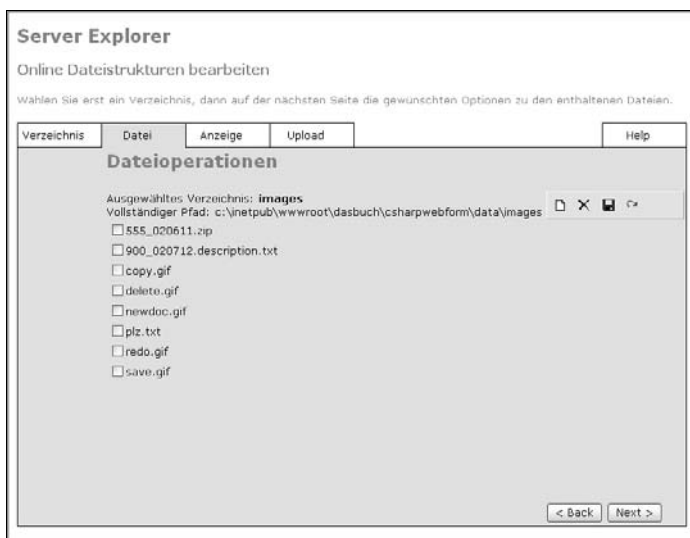


Abbildung 7.17: Anzeige einer Dateiliste und Funktionswahl über Toolbar (rechts)

Eine der bereits fertigen Funktionen ist die Anzeige von Daten über eine ausgewählte Datei. Dazu wird automatisch eine Registerkarte weitergeschaltet, wo normaler Dateizugriff erfolgt. Der Aufbau der Steuerelemente dieser Seite erfolgt weitgehend dynamisch.

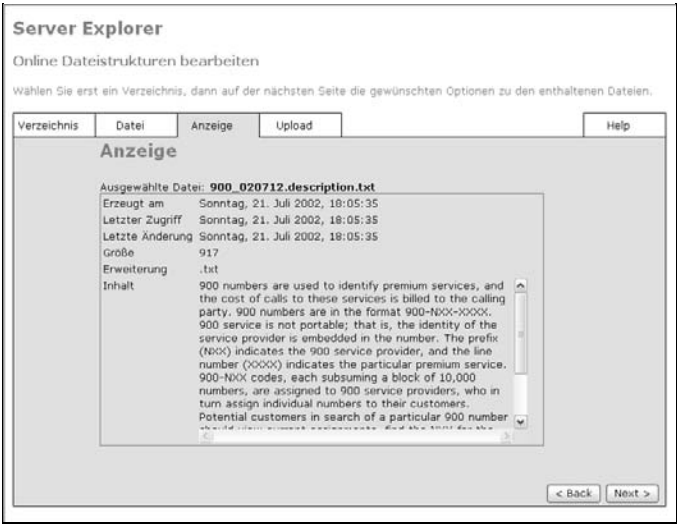


Abbildung 7.18: Anzeige von Dateinformationen und -inhalt

Hochladen von Dateien

Das letzte Modul dient dem Hochladen von Dateien. Neben der einfachen Dateiübertragung besteht die Möglichkeit, die Anzahl der Dateien vorher festzulegen.

Zusätzlich wurde eine Hilfeseite vorbereitet und an den rechten Rand der Registerkarten gesetzt.

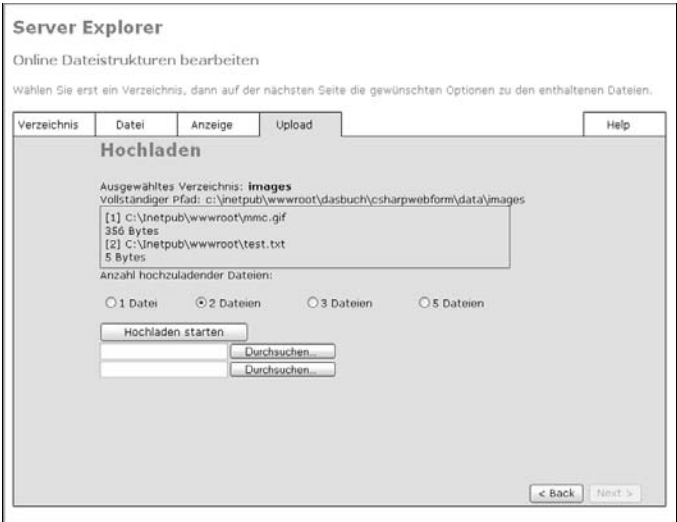


Abbildung 7.19: Komfortables Hochladen mit Wahl der Eingabefelder

7.5.2 TabStrip

Das `TabStrip`-Steuerelement erstellt Registerkarten. Dabei werden nur die Kopfelemente selbst erzeugt. Der zugeordnete Text im unteren Teil der Seite ist davon nicht betroffen. Wie Sie diesen Teil mit `MultiPage` füllen können, wird im nächsten Abschnitt gezeigt. Die Definition der Registerkarten sieht im Beispiel folgendermaßen aus:

```
<iwcc:TabStrip id="CentralTab" runat="server"
    TabDefaultStyle ="border-bottom:1px solid black; ↵
        padding:2px; ↵
        border-left:1x solid black; ↵
        border-top:1x solid black; ↵
        background-color:white" ↵
    TabHoverStyle  =" border-bottom:1px solid black; ↵
        border-left:1x solid black; ↵
        border-top:1x solid black; ↵
        background-color:#FFFFC0;" ↵
    TabSelectedStyle="border-bottom:none; ↵
        border-left:1x solid black; ↵
        border-top:1x solid black; ↵
        background-color:#FFFFC0" ↵
    autopostback="true" ↵
    onselectedIndexChange="CentralTab_SelectedIndexChange"↵
    Width="796px" Height="30px">
<iwcc:Tab Text="Verzeichnis" ID="TabDir" ↵
    DefaultStyle="Width:80px; Text-align:center">
</iwcc:Tab>
<iwcc:Tab Text="Datei" ID="TabFile" ↵
    DefaultStyle="Width:80px; Text-align:center">
</iwcc:Tab>
<iwcc:Tab Text="Anzeige" ID="TabEdit" ↵
    DefaultStyle="Width:80px; Text-align:center">
</iwcc:Tab>
<iwcc:Tab Text="Upload" ID="TabUpload" ↵
    DefaultStyle="Width:80px; Text-align:center">
</iwcc:Tab>
<iwcc:TabSeparator DefaultStyle
    = "border-bottom:1px solid black; ↵
        border-left:1x solid black; ↵
        width:250px">
</iwcc:TabSeparator>
<iwcc:Tab Text="Help" ID="TabHelp" ↵
```

```

DefaultStyle="border-right:1x solid black;
width:80px; text-align:center">
</iewc:Tab>
</iewc:TabStrip>

```

*Listing 7.16: Definition des Steuerelements `TabStrip`
(Ausschnitt aus `leWebControls.aspx`)*

Wie es funktioniert

Der prinzipielle Aufbau stützt sich auf drei Tags:

```

<iewc:TabStrip>
  <iewc:Tab>
  <iewc:TabSeparator>
</iewc:TabStrip>

```

`<TabStrip>` ist das umschließende Element, das auch die grundlegenden Style-Attribute enthält. Dann wird jede einzelne Registerkarte mit einem `<Tab>`-Element erzeugt. Auch hier kann der Style wieder modifiziert werden. Im Beispiel wird dies genutzt, um Breite, Textausrichtung und Ränder einzurichten. Wenn zwischen den Elementen größere Lücken benötigt werden, setzen Sie `<TabSeparator>` ein. Hier wird der Separator verwendet, um die Registerkarte »Hilfe« an den rechten Rand zu verschieben.

Code hinterlegen

Die so definierten Elemente funktionieren schon. Wenn mit der Maus eine Registerkarte angeklickt wird, wechselt der Fokus dahin. Allerdings hat dies keinen praktischen Effekt, wenn nicht Code hinterlegt wird. In der Musterapplikation wird ein Ereignis verwendet, um die jeweils passende Seite aus `MultiPage` zu aktivieren. Das Ereignis reagiert auf einen Wechsel des Index:

```
onselectedIndexChange="CentralTab_SelectedIndexChange"
```

Die Ereignisbehandlungsmethode sieht folgendermaßen aus:

```

Public Sub CentralTab_SelectedIndexChange(ByVal sender As
                                         Object,
                                         ByVal e As EventArgs)
    CentralPage.SelectedIndex = CentralTab.SelectedIndex
    If CentralTab.SelectedIndex = 3 Then
        ModifyFormForUpload()
    End If
End Sub

```

Listing 7.17: Reaktion auf einen Wechsel der Registerkarte (Ausschnitt aus `leWebControls.aspx.vb`)

Die Auswahl des Ereignisses funktioniert, weil die Eigenschaft `autopostback` auf `True` gesetzt wurde. Andernfalls wird das Formu-

lar nicht sofort gesendet. Der Zugriff auf die Registerkarte erfolgt über ihren Index. Im Programm wird der Name `CentralTab` verwendet, die Eigenschaft `SelectedIndex` enthält die aktuelle Auswahl. Im Beispiel wird das verwendet, um sofort den Index der passenden `MultiPage`-Seite zu setzen:

```
CentralPage.SelectedIndex = CentralTab.SelectedIndex
```

Im Beispiel ist auf der Seite zum Hochladen (vierte Registerkarte, Index 3) noch eine besondere Aktion auszuführen. Deshalb erfolgt im entsprechenden Fall der Aufruf der Methode *ModifyFormUpload*. Mehr Informationen dazu finden Sie im Abschnitt 7.5.6, »Weitere Funktionen der Applikation »Server Explorer« ab Seite 703.

7.5.3 MultiPage

Viel mehr passiert auf den Seiten, die durch das Steuerelement `MultiPage` verwaltet werden. Tatsächlich werden mehrere, sich überlagernde Seiten in einer einzigen *aspx*-Datei definiert. Zuerst ein Ausschnitt aus der Applikation:

```
<iwcc:MultiPage style="BORDER-TOP:medium none"
    selectedIndex="0"
    id="CentralPage" runat="server"
    BorderStyle="Solid" BorderWidth="1px"
    Height="450px"
    BackColor="#FFFFC0" Width="796px">
    <iwcc:PageView ID="PageList">
        <!-- Inhalt von Seite 1 -->
    </iwcc:PageView>
    <!-- Weitere Seiten -->
</iwcc:MultiPage>
```

*Listing 7.18: Definition des Steuerelements MultiPage
(Ausschnitt aus leWebControls.aspx)*

Das eigentliche Geheimnis der Seiten steckt weniger in der Definition. Tatsächlich definiert `MultiPage` nur einen Rahmen, in den normales HTML und natürlich ASP.NET-Server-Steuerelemente eingebettet werden können.

Wie es funktioniert

Interessant ist die Navigation mit den Schaltflächen »Next« und »Back«, die von einer Registerkarte zur nächsten springen und am Anfang und Ende die nicht verfügbare Richtung durch deaktivierte Schaltflächen anzeigen. Selbstverständlich soll die Darstellung der Seite außerdem mit den Registerkarten korrespondieren. Ein Blick auf den Code zeigt, wie das funktioniert.

MultiPage praktisch steuern

Schauen Sie sich zuerst die Definition der Schaltflächen an. Dies sind Button-Steuerelemente, die mit dem entsprechenden Ereignis verknüpft sind:

```
<asp:Button id="Button1" ↵
           runat="server" Text="< Back" Enabled="False" />
<asp:Button id="NextBtn2" ↵
           runat="server" Text="Next >" OnClick="NextClick2" />
```

Die erste Schaltfläche ist gesperrt, weil sie auf der ersten Seite steht und dort kein Rückschritt mehr möglich ist. Die zweite ist mit einem `OnClick`-Ereignis verbunden, das durch die Ereignisbehandlungsmethode `NextClick2` bedient wird:

```
Public Sub NextClick2(ByVal sender As Object, ByVal e As
EventArgs)
    CentralPage.SelectedIndex = 1
    CentralTab.SelectedIndex = 1
End Sub
```

Hier werden zwei Funktionen ausgelöst: Wechsel der Registerkarte und Wechsel der Seite auf einen festen Index. Hier ist es die zweite Seite mit dem Index 1. Der Index ist nullbasiert. Die übrigen Definitionen auf den anderen Seiten sind entsprechend aufgebaut und nutzen für jedes Ziel – insgesamt vier für die vier Seiten – eine eigene Ereignisbehandlungsmethode.

Die eigentliche Anzeige auf der ersten Seite erfolgt durch das Steuerelement `TreeView`. Es zeigt die gesamte Verzeichnisstruktur Ihres Webs an. Der nächste Abschnitt zeigt, wie es funktioniert.

7.5.4 TreeView

`TreeView` dient der Darstellung von hierarchischen Strukturen mit Hilfe einer Baumansicht. Sie können es auf drei Wegen erstellen:

- ▶ Definition mit entsprechenden Tags in der HTML-Seite
- ▶ Erzeugen der Knoten über ein Programm
- ▶ Übergabe einer XML-Datei mit einer Strukturdefinition

TreeView definieren

In der Musterapplikation kommt die zweite Form zur Anwendung. Zuerst ein Blick auf die Definition des Steuerelements:

```
<iewc:TreeView id="DirView" runat="server"
    OnSelectedIndexChanged="DirView_SelectedIndexChanged"
    DefaultStyle="font-family:Verdana; font-size:10pt" />
```

*Listing 7.19: Definition des Steuerelements `TreeView`
(Ausschnitt aus `leWebControls.aspx`)*

Da die Knoten im Programm erzeugt werden, ist der Aufbau recht einfach. Weitere Möglichkeiten werden am Ende des Abschnitts gezeigt. Damit das ausgewählte Verzeichnis im Programm erkannt werden kann, wird wieder ein Ereignis definiert:

```
OnSelectedIndexChanged="DirView_SelectedIndexChanged"
```

Zuvor muss jedoch der Aufbau der Knoten erfolgen. Diese Funktion wird nur beim ersten Start des Programms aufgerufen. Nachfolgend werden die Daten im Anzeigestatus der Seite gehalten. Dies sollten Sie beachten, wenn Sie diese Technik mit sehr großen Verzeichnisstrukturen verwenden. Zwar wird beim erneuten Laden der Seite verhindert, dass immer wieder auf den Verzeichnisbaum zugegriffen wird, aber der Anzeigestatus, der als verstecktes Feld in der Seite gespeichert wird, wächst sehr schnell an. Im Ergebnis werden die Seiten sehr groß, was zu langen Ladezeiten führt und für Benutzer möglicherweise irritierend ist. Die `Page_Load`-Methode sieht nun folgendermaßen aus:

```
Public Sub Page_Load(ByVal sender As Object, ByVal e As
EventArgs) _
    Handles MyBase.Load
    If Not Page.IsPostBack Then
        Dim di As DirectoryInfo = New
DirectoryInfo(Server.MapPath("."))
        ' Application Virtual Root
        DirViewNode = New TreeNode()
        DirViewNode.Text = di.Name
        DirView.Nodes.Add(DirViewNode)
        AddDirectories(di, DirViewNode)
    End If
End Sub
```

*Listing 7.20: Füllen des Steuerelements `TreeView`
(Ausschnitt aus `leWebControls.aspx.vb`)*

Zuerst wird auf die Verzeichnisinformationen zugegriffen:

```
Dim di As DirectoryInfo = New DirectoryInfo(Server.MapPath("."))
```

**Wie es
funktioniert**

Dann wird ein neuer Knoten erzeugt – dies wird der Wurzelknoten:

```
DirViewNode = new TreeNode()
```

Dem Knoten wird der Verzeichnisname zur Anzeige zugewiesen:

```
DirViewNode.Text = di.Name
```

Jetzt wird dem Baum der neue Knoten als erstes Element hinzugefügt:

```
DirView.Nodes.Add(DirViewNode)
```

Alle weiteren Knoten werden nun durch eine rekursive Funktion an diesen Wurzelknoten angehängt:

```
AddDirectories(di, DirViewNode)
```

Die Methode *AddDirectories* durchläuft nun alle tiefer liegenden Verzeichnisse und erzeugt jedes Mal neue Knoten, die an den Baum angehängt werden:

```
Private Sub AddDirectories(ByVal di As DirectoryInfo, ByVal tn As
TreeNode)
    Dim subdir As DirectoryInfo
    For Each subdir In di.GetDirectories
        DirViewNode = New TreeNode()
        DirViewNode.Text = subdir.Name
        tn.Nodes.Add(DirViewNode)
        If Not subdir.GetDirectories Is Nothing Then
            AddDirectories(subdir, DirViewNode)
        End If
    Next
End Sub
```

*Listing 7.21: Rekursives Füllen des Steuerelements **TreeView**
(Ausschnitt aus **leWebControls.aspx.vb**)*

**Wie es
funktioniert**

Hier wird nun für jedes Verzeichnis ein Knoten erzeugt. Enthält das aktuelle Verzeichnis *subdir* Unterverzeichnisse, wird die Funktion rekursiv aufgerufen:

```
If Not subdir.GetDirectories Is Nothing Then
```

Dem rekursiven Aufruf wird der aktuelle Knoten übergeben, damit die tiefer liegenden Verzeichnisse dort angehängt werden:

```
AddDirectories(subdir, DirViewNode)
```

Die Funktionen des Steuerelements, wie Auf- und Zuklappen, müssen nicht programmiert werden. Mit dem Absenden des Formulars wird die aktuelle Auswahl über die Ereignisbehandlungsmethode `DirView_SelectedIndexChange` übernommen.

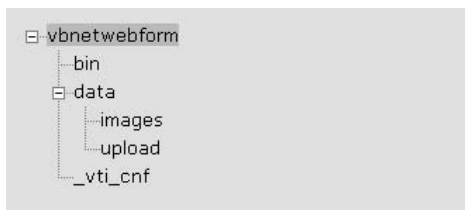


Abbildung 7.20: Das Auf- und Zuklappen der Zweige wird im Client ausgeführt.

Hier zeigt sich nun der Vorteil der DHTML-Lösung: Das Auf- und Zuklappen führt nicht zum Senden des Formulars, sondern wird komplett im Browser ausgeführt. Die Übernahme der getroffenen Auswahl, in Abbildung 7.20 ist dies der grau hinterlegte Name, erfolgt über die Ereignisbehandlungsmethode:

```
Public Sub DirView_SelectedIndexChange(ByVal sender As Object,   
                                     ByVal e As TreeViewSelectEventArgs)   
    CentralPage.SelectedIndex = 1   
    CentralTab.SelectedIndex = 1   
    Dim CurrentViewNode As TreeNode   
        = DirView.GetNodeFromIndex(e.NewNode)   
    CurrentDirectory.Text = CurrentViewNode.Text   
    Dim di As DirectoryInfo   
        = New DirectoryInfo(GetCurrentPath(e.NewNode))   
    CurrentPath.Text = di.FullName   
    If Not di.GetFiles() Is Nothing Then   
        FileListView.DataSource = di.GetFiles()   
    Else   
        FileListView.Controls.Add(New LiteralControl("Keine   
                                                         Datei"))   
    End If   
    FileListView.DataBind()   
End Sub
```

Listing 7.22: Die Auswahl des Verzeichnisses bestimmt, wie die Dateiliste gefüllt wird (Ausschnitt aus `leWebControls.aspx.vb`).

Wie es funktioniert

Das Absenden der Verzeichnisauswahl führt zuerst auf die Seite mit der Dateianzeige, in dem der Index für `Tabstrip` und `MultiPage` gesetzt wird. Dann wird der ausgewählte Knoten ermittelt:

```
CurrentViewNode As TreeNode = DirView.GetNodeFromIndex (e.NewNode)
```

Übergeben wird ein spezielles Ereignisargument, `TreeViewSelectEventArgs`. In diesem sind die Eigenschaften `NewNode` (aktuelle Auswahl) und `OldNode` (letzte Auswahl) vorhanden. Über die Eigenschaft `Text` kann auf den Verzeichnisnamen zugegriffen werden. Wenn – wie im Beispiel – der gesamte Pfad benötigt wird, stellt sich die Sache leider nicht so einfach dar.

Indizierung der Knoten

Der Index des `TreeView`-Steuerelements zählt nicht fortlaufend, wie bei `MultiPage`, sondern bildet die Hierarchie ab. `e.NewNode` enthält den Index in Form einer Zeichenkette, beispielsweise: »0.1.0«. Dies ist das erste Element der obersten Ebene, das zweite Element der zweiten Ebene und das erste Element der dritten Ebene. Der in Abbildung 7.20 gezeigt Baum hat folgende Indexstruktur:

```
0           = vbnetwebform
0.0         = bin
0.1         = data
0.1.0       = images
0.1.1       = resources
0.1.2       = _vti_cnf
0.2         = templates
0.2.0       = img
0.2.0.0     = swf
0.2.1       = includes
0.3         = _vti_cnf
```

Pfad ermitteln

Es besteht also die Aufgabe, aus der Angabe »0.2.0.0« den Pfad »\vbnetwebform\templates\img\swf« zu berechnen. Dies übernimmt die Methode *GetCurrentPath*:

```
Private Function GetCurrentPath(ByVal NodeCode As String) As String
    Dim NodePath() As String = NodeCode.Split(".")
    Dim FullPathInfo As StringBuilder = New StringBuilder()
    FullPathInfo.Append(Request.ApplicationPath.Substring(0, Request.ApplicationPath.LastIndexOf("/")))
    Dim ss As String = String.Empty
    Dim i As Integer
    For i = 1 To NodePath.Length
        DirViewNode = DirView.GetNodeFromIndex(String.Join(".", ss, NodePath(i)))
```

```

                                NodePath, 0, i))
    FullPathInfo.AppendFormat("\{0}", DirViewNode.Text)
    Next
    Return Server.MapPath(FullPathInfo.ToString())
End Function

```

Listing 7.23: GetCurrentPath ermittelt aus dem TreeView-Objekt vollständige Pfade

Übergeben wird der Code des Knotens, beispielsweise »0.1«; das TreeView-Objekt ist öffentlich sichtbar, weshalb direkt zugegriffen werden kann. Nun wird der »Knotenpfad« zuerst in ein Array zerlegt:

Wie es funktioniert

```
Dim NodePath() As String = NodeCode.Split(".")c)
```

Da eine längere Zeichenkette entsteht, wird die Klasse `StringBuilder` eingesetzt:

```
Dim FullPathInfo As StringBuilder = New StringBuilder()
```

Den Anfang des Pfades bildet der Stammpfad zu Ihrem Web, ohne das Wurzelverzeichnis. Dies ist zwar eine etwas umständliche Berechnung, vereinfacht aber wesentlich die Extraktion der Namen aus dem TreeView-Objekt:

```
FullPathInfo.Append (Request.ApplicationPath.Substring(0,
                                                         Request.ApplicationPath.LastIndexOf("/")
```

Dann wird das Array von vorn beginnend durchlaufen:

```
For i = 1 To NodePath.Length
```

In jedem Schritt wird der komplette Knotenpfad durch Auswahl der passenden Anzahl Arrayelemente ermittelt und dann direkt auf den Knoten zugegriffen. Der folgende Teil extrahiert einen Knotenpfadabschnitt, von vorn beginnend:

```
String.Join(".", NodePath, 0, i)
```

Es werden also bei dem Pfad »0.0.2.1« nacheinander folgende Knotenpfade erstellt:

```

0
0.0
0.0.2
0.0.2.1

```

Diese werden dann zum Aufruf der Methode `GetNodeFromIndex` verwendet. Aus dem so ermittelten Knoten wird der Pfadteil entnommen und zum Gesamtpfad zusammengesetzt:

```
FullPathInfo.AppendFormat("{0}", DirViewNode.Text)
```

Zuletzt bleibt nur die Rückgabe des vollständigen, physischen Pfades, wofür wieder `MapPath` zum Einsatz kommt:

```
Return Server.MapPath(FullPathInfo.ToString())
```



Dieselbe Technik ist auch einsetzbar, wenn Sie ein Menüsystem mit `TreeView` aufbauen. Es ist hilfreich für den Benutzer, neben der Baumdarstellung immer einen Zugriff auf den Pfad zu haben, sich also jederzeit über die Ebene des gerade geöffneten Menüs in der Struktur erkundigen zu können.

Dateiliste erzeugen

Nach erfolgreicher Pfadermittlung kann die Dateiliste erzeugt werden, wenn Dateien vorhanden sind:

```
If Not di.GetFiles() Is Nothing Then
```

Die Dateiliste wird der Auflistung `CheckBoxList` aus der Datenquelle zugewiesen:

```
FileListView.DataSource = di.GetFiles()
```

Die Anzeige erfolgt durch Bindung der Daten:

```
FileListView.DataBind()
```

Die Auswahl der Funktionen für Dateien erfolgt nun über das `Toolbar`-Steuerelement. Lesen Sie dazu in Abschnitt 7.5.5, »Toolbar« ab Seite 699 weiter. Nachfolgend finden Sie ergänzende Informationen über `TreeView`, die im Projekt nicht verwendet wurden.



Abbildung 7.21: Dateiliste mit Kontrollkästchen, rechts das `Toolbar`-Steuerelement

TreeView statisch definieren

Soll das Element nur statisch genutzt werden, erzeugen Sie die Knoten mit <treenode>-Elementen. Die Verschachtelung bestimmt dann die Struktur:

```
<iewc:TreeView runat="server" id="mainmenu">
  <iewc:TreeNode text="Hauptmenü">
    <iewc:TreeNode text="Aktivierung"></iewc:TreeNode>
    <iewc:TreeNode text="Konfiguration"></iewc:TreeNode>
      <iewc:TreeNode text="Datenbank"></iewc:TreeNode>
      <iewc:TreeNode text="Basisdaten"></iewc:TreeNode>
    </iewc:TreeNode>
    <iewc:TreeNode text="Benutzerdaten"></iewc:TreeNode>
    <iewc:TreeNode text="Hilfe"></iewc:TreeNode>
  </iewc:TreeNode>
</iewc:TreeView>
```

Nach der Definition des Gitters müssen Sie überlegen, wie die Gestaltung erfolgt und wie die Elemente an Interaktion gewinnen.

Die Gestaltung kann durch Styles (CSS) und Festlegung eigener Bilder für die Knotenpunkte erfolgen. Die Bilder müssen zwei Zustände abdecken. In der Standarddarstellung erscheinen geschlossene Knoten als Plus-Zeichen, offene als Minus-Zeichen. Beim Klick darauf wechselt der Zustand. Zwei Attribute erlauben die Angabe zusätzlicher Bilder vor dem Text, beispielsweise Ordnersymbole:

```
<iewc:TreeView runat="server" id="mainmenu"
  ExpandedImageUrl="./images/folderopen.gif"
  ImageUrl="./images/folderclosed.gif"
>
```

Sie können auf jeder Ebene, also sowohl in TreeView als auch Tree Node, Bilder definieren. Weil das recht umständlich werden kann, erben untergeordnete Elemente die Angabe automatisch. Wenn Sie mit verschiedenen Bildern in verschiedenen Ebenen arbeiten, können Sie zwei weitere Attribute nutzen, um die Vererbung zu steuern. Definieren Sie zusammen mit den Bildern ein Attribut Child Type="name". In den Kindelementen tragen Sie dann Type="name" ein und die Bilder des Elternelements werden übernommen.

Als Gestaltungsattribute stehen DefaultStyle (Standardanzeige), HoverStyle (Aktiv, wenn die Maus über dem Element schwebt) und SelectedStyle (das ausgewählte Element) zur Verfügung. Ver-

Gestaltung

Vererben von Bilddaten

Gestaltungsattribute

wenden Sie als Parameter CSS-Werte. Es ist außerdem möglich, in den Text HTML zu schreiben:

```
<iewc:TreeNode text="<b>Hilfe</b>"></iewc:TreeNode>
```

TreeView mit Hilfe einer XML-Datei füllen

XML als Knoten- quelle

Wenn Sie strukturierte Daten anzeigen und darin navigieren möchten, kann `TreeView` ebenfalls sehr hilfreich sein. Es ist möglich, eine XML-Datei als Datenquelle anzugeben. Das Format ist fest vorgegeben:

```
<?xml version="1.0"?>
<TREENODES>
  <treenode text="Hauptmenü"/>
</TREENODES>
```



Beachten Sie, dass das umschließende Tag `<TREENODES>` tatsächlich groß geschrieben werden muss, während die inneren Tags Kleinbuchstaben erlauben.

Nun liegen die Daten nicht immer in dieser Form vor. Es bietet sich an, bestehende XML-Daten mit XSLT für den Zweck der Anzeige in `TreeView` zu transformieren. Dazu geben Sie die XSLT-Datei mit dem Attribut `TreeNodeXsltSrc` an.



Informationen über XSLT und eine kompakte Einführung finden Sie im Abschnitt 5.9, »Daten transformieren mit XSLT« ab Seite 478.

TreeView mit beliebigen XML-Daten füllen

Die folgende XML-Datei soll als Baum dargestellt werden:

```
<?xml version="1.0" encoding="utf-8" ?>
<regions>
  <region name="EUROPE">
    <country>ALBANIA</country>
    <country>BELGIUM</country>
    ...
  </region>
  <region name="AFRICA">
    <country>ALGERIA</country>
```

```

        <country>ANGOLA</country>
        ....
    </region>
</regions>

```

Listing 7.24: Ausschnitt aus der Datei data/regions.xml

Um die Darstellung in das Schema `<TREENODES><TreeNode>` zu überführen, soll XSLT verwendet werden. Die XSLT-Datei, die die Transformation übernimmt, sieht folgendermaßen aus:

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <TREENODES>
      <xsl:apply-templates/>
    </TREENODES>
  </xsl:template>
  <xsl:template match="//regions/region">
    <xsl:element name="TreeNode">
      <xsl:attribute name="Text">
        <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="//regions/region/country">
    <xsl:element name="TreeNode">
      <xsl:attribute name="Text">
        <xsl:value-of select="."/>
      </xsl:attribute>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>

```

Listing 7.25: Transformation der Datei data/regions.xml in das Format, das TreeView versteht

Nun kann die Definition des Baumes erfolgen. Es sind lediglich die beiden Dateien (XML und XSLT) als Parameter zu übergeben:

```

<iewc:TreeView runat="server" id="mainmenu" >
  <iewc:TreeNode text="Reiseziele"
    ChildType="Folder" Expanded="true"
    TreeNodeSrc="data/regions.xml"
    TreeNodeXsltSrc="data/region2treeview.xslt">
  </iewc:TreeNode>
</iewc:TreeView>

```

Listing 7.26: Definition des Steuerelements (Ausschnitt aus WebControlTreeView.aspx)

Code müssen Sie nicht schreiben. Gestaltung und Inhalt werden durch Styles in der *aspx*-Seite und durch die XML-Daten bestimmt.

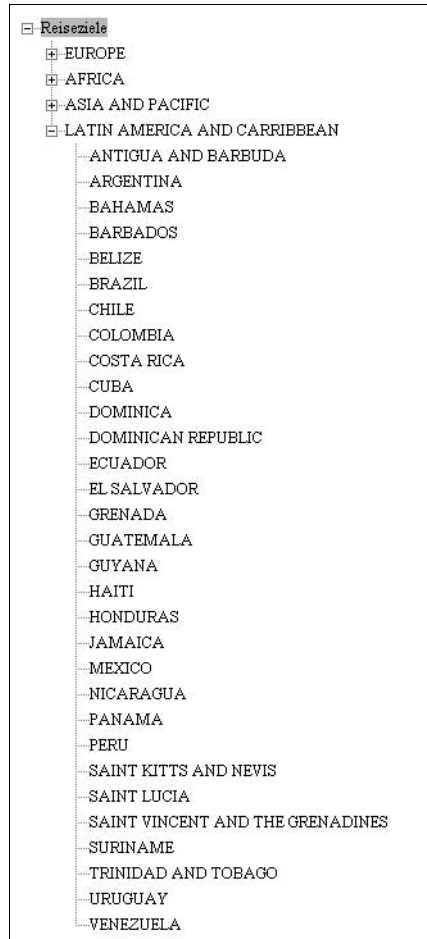


Abbildung 7.22: Der Baum – direkt aus den XML-Daten aus *data/regions.xml* erstellt

Das Beispiel zeigt – zum wiederholten Male – welche Möglichkeiten XML bietet. Immerhin ist die Verarbeitung mit den vielen XML-tauglichen Klassen im Framework sehr leicht möglich.



Bei Versuchen mit XML-Dateien war zu beobachten, dass Namespace-Attribute und XSD-Verweise störend sind. Daten aus strukturierten oder relationalen Datenquellen, die Schema-Beschreibungen mit sich führen, sollten zuvor von diesen Attributen befreit werden.

Eine universelle Transformation für beliebige XML-Dateien

Es ist nahe liegend, sich bei der Gestaltung der Transformation nicht auf bestimmte Tags festzulegen. Das folgende XSLT-Skript transformiert alles, was irgendwie gültiges XML ist, in ein von TreeView akzeptiertes Format. Das Programm *UniversalTreeView.aspx* erledigt dies. Hier der Aufruf:

Universelles XSLT-Skript

```
<iewc:TreeView runat="server" id="mainmenu" ␣
    DefaultStyle="font-family:Verdana; font-size:12pt;" ␣
    HoverStyle="font-family:Verdana; font-size:12pt;">
    <iewc:TreeNode text="XML" Expanded="true" ␣
        TreeNodeSrc="Web.config.xml" ␣
        TreeNodeXsltSrc="data/universal.xslt">
</iewc:TreeNode>
```

Listing 7.27: XSLT erledigte hier die Arbeit: Darstellung beliebiger XML-Dateien (UniversalTreeView.aspx)

Im Beispiel wird eine Datei *Web.config.xml* verwendet. Dies ist eine Kopie der *web.config*, auf die Sie über ASP.NET aus Sicherheitsgründen keinen direkten Zugriff haben. Die XSLT-Datei definiert die nötigen Umwandlungsvorgänge und ein paar Gestaltungselemente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" ␣
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <TREENODES>
            <xsl:apply-templates select="/*" />
        </TREENODES>
    </xsl:template>
    <xsl:template match="/*">
        <xsl:element name="TreeNode">
            <xsl:attribute name="Text">
                <xsl:value-of ␣
                    select="concat('&lt;',name(.),'&gt;')"/>
                (Stammelement)
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="*">
        <xsl:element name="TreeNode">
            <xsl:attribute name="Text">
                <xsl:value-of ␣
```

```

        select="concat('&lt;','&lt;',name(.),'&gt;')"/>
        <xsl:value-of select="text()"/>
    <xsl:value-of
        select="concat('&lt;','&lt;',name(.),'&gt;')"/>
    </xsl:attribute>
    <xsl:for-each select="@*">
        <xsl:call-template name="attributes">
            <xsl:with-param name="attributenode" select="."/>
        </xsl:call-template>
    </xsl:for-each>
    <xsl:apply-templates select="*" />
</xsl:element>
</xsl:template>
<xsl:template name="attributes">
    <xsl:param name="attributenode"/>
    <xsl:element name="TreeNode">
        <xsl:attribute name="Text">
            &lt;span style="font-size:10pt">
                &lt;b><xsl:value-of select="name()"/>&lt;/b>
                ="&lt;span style="color:red">
                    <xsl:value-of select="."/>&lt;/span>"
                &lt;/span>
            </xsl:attribute>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>

```

Listing 7.28: Die Transformationsdatei *data/universal.xslt*

Die Ausgabe zeigt den kompletten XML-Baum – ohne Kommentarseiten – in der Baumstruktur an. Dies ist deutlich besser lesbar als die XML-Dateien direkt zu betrachten (siehe Abbildung 7.23).

Das Programm zeigt auf beeindruckende Weise, dass aktuelle Technologien wie XSLT den Programmieraufwand signifikant verringern, wenn damit verbundene Methoden clever eingesetzt werden. Erst im Gesamtzusammenhang aller XML-Anwendungen zeigt sich, dass der reine Code – egal ob in VB.NET oder C# – deutlich an Bedeutung verliert. Dies führt letztlich zu stabileren, sichereren und leichter zu pflegenden Programmen.



Abbildung 7.23: Ausgabe einer beliebigen XML-Datei, hier web.config, mit dem TreeView-Steuerelement

7.5.5 Toolbar

Im Projekt wurde auch ein `ToolBar`-Steuerelement eingesetzt. Dies ist eine Zusammenstellung von Symbolen, die Aktionen auslösen. Das unterscheidet sich kaum von Bildschaltflächen. Es wird Ihnen allerdings einiges an Gestaltungsarbeit abgenommen. Sie müssen auch nicht zwei Bilder für die Reaktion auf den Klick (gedrückt und normal) anfertigen, denn das Steuerelement simuliert den 3D-Effekt durch Veränderung des Randes.

Das im Projekt verwendete `ToolBar`-Steuerelement wird folgendermaßen definiert:

```

<iwec:ToolBar DefaultStyle="width:20px" 
  runat="server" id="FileOperation">
  <iwec:ToolBarButton tooltip="Neue Datei erzeugen" 
    imageUrl="data/images/newdoc.gif" 
    id="FileNew" />
  <iwec:ToolBarButton tooltip="Ausgewählte Dateien löschen" 
    imageUrl="data/images/delete.gif" 
    id="FileDelete" />

```

```

<iwc:ToolBarButton tooltip="Datei herunterladen" ↵
    imageUrl="data/images/save.gif" ↵
    id="FileDownload" />
<iwc:ToolBarButton tooltip="Dateidaten und Inhalt anzeigen" ↵
    OnButtonClick="FileOperation_Show" ↵
    imageUrl="data/images/redo.gif" ↵
    id="FileShow" />
</iwc:ToolBar>

```

Listing 7.29: Eine Toolbar (Ein weiterer Ausschnitt aus leWebControls.aspx)

Wie es funktioniert

Unbedingt notwendig ist das Attribut `ImageUrl` des Elements `ToolBarButton`, mit dem das Bild zugewiesen wird. Für Symbolleisten ist außerdem der `ToolTip` üblich. Sie müssen dieses Element nicht angeben, es ist aber Teil einer guten Benutzerführung.



Abbildung 7.24: ToolTip einer Toolbar

Ereignisse

Um eine Aktion zu starten, definieren Sie das Attribut `OnButtonClick` mit dem Namen einer Ereignisbehandlungsmethode:

```
OnButtonClick="FileOperation_Show"
```

In der Musterapplikation zeigt diese Methode eine Tabelle mit verschiedenen Dateiinformatoren an. Die Tabelle wird dynamisch, das heißt zur Laufzeit erzeugt. Die entsprechende Seite enthält lediglich einen Platzhalter:

Ausgewählte Datei:

```

<asp:Label Font-Bold="True" Runat="server" ID="FileName" />
<asp:Placeholder Runat="server" ID="FileShowBox" />

```

Die Methode wechselt auf die entsprechende Seite und liest dann die Dateidaten ein. Aus den Angaben wird eine Tabelle erzeugt:

```

Public Function FileOperation_Show(ByVal sender As Object, ↵
                                   ByVal e As EventArgs) As
Boolean
    CentralPage.SelectedIndex = 2
    CentralTab.SelectedIndex = 2
    If FileListView.SelectedItem Is Nothing Then
        Return False
    End If

```



```

End If
FileName.Text = FileListView.SelectedItem.Value
Dim fi As FileInfo
    = New FileInfo(GetCurrentPath
        (DirView.SelectedNodeIndex)
        + "\" + FileListView.SelectedItem.Value)
Dim t As Table = New Table()
t.Height = Unit.Percentage(50.0)
t.BorderWidth = Unit.Pixel(1)
t.BorderColor = Color.Blue
t.Rows.Add(NextRow("Erzeugt am",
    fi.CreationTime.ToLongDateString()
    + ", " + fi.CreationTime.ToLongTimeString()))
t.Rows.Add(NextRow("Letzter Zugriff",
    fi.LastAccessTime.ToLongDateString() + ", "
    + fi.LastAccessTime.ToLongTimeString()))
t.Rows.Add(NextRow("Letzte Änderung",
    fi.LastWriteTime.ToLongDateString() + ", "
    + fi.LastWriteTime.ToLongTimeString()))
t.Rows.Add(NextRow("Größe", fi.Length.ToString()))
Dim Extension As String = fi.Extension
t.Rows.Add(NextRow("Erweiterung", Extension))
Select Case Extension
    Case ".txt"
    Case ".xml"
    Case ".xsd"
    Case ".xslt"
    Case ".xsl"
        Dim sr As StreamReader
            = New StreamReader(fi.FullName)
        Dim cc As HtmlGenericControl
            = New HtmlGenericControl("div")
        cc.Attributes("style") = "overflow:scroll;
            width:400px; height:200px"
        cc.InnerText = sr.ReadToEnd()
        sr.Close()
        t.Rows.Add(NextRow("Inhalt", cc))
        Exit Function
    Case ".gif"
    Case ""
    Case ".png"
        Dim img As System.Web.UI.WebControls.Image
            = New System.Web.UI.WebControls.Image()
        img.ImageUrl = fi.FullName
        t.Rows.Add(NextRow("Bild", img))
        Exit Function
    Case Else
        t.Rows.Add(NextRow("Inhalt", "Unbekannter Dateityp"))
        Exit Function

```

```

        End Select
        FileShowBox.Controls.Add(t)
        Return True
    End Function

```

Listing 7.30: Ereignisbehandlungsmethode FileOperation_Show

Wie es funktioniert

Der Wert des CheckBoxList-Steuerelements FileListView bestimmt den Dateinamen, der zuerst nur angezeigt wird:

```
FileName.Text = FileListView.SelectedItem.Value
```

Der vollständige Pfad (Methode *GetCurrentPath*) wird für das FileInfo-Objekt benötigt:

```

Dim fi As FileInfo ↵
    = New FileInfo (GetCurrentPath(DirView.SelectedNodeIndex) ↵
    + "\" + FileListView.SelectedItem.Value)

```

Nun wird die Tabelle erzeugt:

```
Dim t As Table = New Table ()
```

Es werden nun einige Gestaltungsattribute gesetzt, unter anderem ein blauer, ein Pixel breiter Rand:

```

t.BorderWidth = Unit.Pixel (1)
t.BorderColor = Color.Blue

```

Dann werden Reihen hinzugefügt, wobei die Methode *NextRow* hilfsweise den Aufbau einer komplette Reihe übernimmt:

```

t.Rows.Add (NextRow ↵
    ("Erzeugt am", fi.CreationTime.ToLongDateString () ↵
    + ", " + fi.CreationTime.ToLongTimeString ()))

```

Die Methode *NextRow* zeigt weitere Details bezüglich der Erzeugung von Tabellenelementen:

```

Private Function NextRow(ByVal label As String, ↵
    ByVal text As Object) As TableRow
    Dim tr As TableRow = New TableRow()
    tr.VerticalAlign = VerticalAlign.Top
    Dim tc1 As TableCell = New TableCell()
    tc1.Text = label
    Dim tc2 As TableCell = New TableCell()
    If text.GetType().Name.ToLower() = "string" Then
        tc2.Text = CType(text, String)
    Else
        tc2.Controls.Add(CType(text, Control))
    End If

```

```

tr.Cells.Add(tc1)
tr.Cells.Add(tc2)
Return tr
End Function

```

Listing 7.31: Die Methode *NextRow* erzeugt eine Reihe in Abhängigkeit von den Daten

Durch die Angabe des Datentyps `object` bei der Deklaration kann sowohl einfacher Text als auch ein beliebiges Steuerelement angenommen und dann in der Zelle platziert werden. Die Unterscheidung wird im Code anhand des ermittelten Types ermittelt:

```
If text.GetType().Name.ToLower() = "string" Then
```

Der Rest der Hauptmethode *FileOperation_Show* beschäftigt sich mit der Darstellung des Dateiinhalts bei textbasierten Dateitypen oder der Anzeige, wenn es sich um Bilddaten handelt. In beiden Fällen werden die benötigten Steuerelemente dynamisch erzeugt.

Ausgewählte Datei: mmc.gif	
Erzeugt am	Sonntag, 21. Juli 2002, 18:52:16
Letzter Zugriff	Sonntag, 21. Juli 2002, 18:52:16
Letzte Änderung	Sonntag, 21. Juli 2002, 18:52:16
Größe	356
Erweiterung	.gif
Bild	

Abbildung 7.25: Die dynamisch erzeugte Tabelle mit einem Bild

7.5.6 Weitere Funktionen der Applikation

»Server Explorer«

Die Darstellung musste hier aus Platzgründen auf ein Minimum beschränkt werden. Sie sollten sich intensiv mit der Dokumentation der Internet Explorer WebControls auseinander setzen, denn der Einsatz spart jede Menge Tipparbeit und Überlegung und verleiht Applikationen ein professionelles Aussehen.

An dieser Stelle soll noch kurz die Hochlade-Funktion (Upload) vorgestellt werden, vor allem zur Ergänzung der bereits in Abschnitt 6.3.5, »Dateien per HTTP hochladen (Upload)« ab Seite 530 vorgestellten Basisfunktionen für solche Dateioperationen.

Vorbereitung des Formulars

Formular zum Hochladen vorbereiten

Um Dateien per HTTP hochladen zu können, muss das Formular eine spezielle Kodierung verwenden. Sie können diese fest im Formtag definieren:

```
<form runat="server" enctype="multipart/form-data" ... >
```

Programmier-techniken

Allerdings wird das im vorgestellten Programm nicht gemacht und es funktioniert trotzdem. Wenn Sie sich jedoch den Quelltext im Browser ansehen, finden Sie den Eintrag dennoch. Das entsprechende Attribut wird dynamisch hinzugefügt, wenn ein Upload erstmalig angefordert wird. Dazu wird das Ereignis ausgewertet, das zum Sprung auf die vierte Seite des `MultiPage`-Elementes führt, entweder über die Schaltfläche oder die Registerkarte. In den entsprechenden Methoden wird dann jeweils die folgende Methode aufgerufen:

```
Private Sub ModifyFormForUpload()
    Dim fc As HtmlForm = CType(Page.FindControl("FileForm"),
        HtmlForm)
    fc.Enctype = "multipart/form-data"
    Dim i As Integer
    For i = 0 To 5 - 1
        UploadControls.FindControl("UploadFile" &
            + i.ToString()).Visible =
False
    Next
    If Not DirView.SelectedNodeIndex Is String.Empty Then
        UploadDir.Text &
        =
        DirView.GetNodeFromIndex(DirView.SelectedNodeIndex).Text
        UploadPath.Text &
        = GetCurrentPath(DirView.SelectedNodeIndex)
    End If
End Sub
```

Listing 7.32: Methode zum Vorbereiten eines Formulars zum Datei-Hochladen (Ausschnitt aus `leWebControls.aspx.vb`)

Wie es funktioniert

Mit dieser Methode wird im Formular der Kodierungstyp gesetzt:

```
fc.Enctype = "multipart/form-data"
```

Dann werden die fünf Dateiladefelder unsichtbar gemacht (`Visible = False`). Im HTML-Code sind diese folgendermaßen definiert:

```
<input type="file" id="UploadFile0" &
    runat="server" NAME="UploadFile0" />
```

Damit Sie die Steuerelemente dynamisch, über die Schleife, erreichen können, wird `FindControl` eingesetzt. Leider arbeitet diese Methode nicht rekursiv, sondern durchsucht nur den aktuellen Container. Deshalb stehen die Felder alle innerhalb eines als Server-Steuerelement deklarierten `<div>`-Tags mit dem Namen `UploadControls`. Auf diesem Wege erhalten Sie dynamischen Zugriff auf die Elemente.

FindControl

Die Anzahl der Elemente wird über eine Auswahlliste vom Typ `RadioButtonList` gesteuert. Deren Ereignisbehandlungsmethode schaltet die Sichtbarkeit der gewünschten Anzahl Felder ein.

Als Letztes bleibt noch die Reaktion auf den Sendevorgang selbst, dies wird mit der Ereignisbehandlungsmethode `Upload_SubmitClicked` erledigt, die von der Sende-Schaltfläche des Formulars ausgelöst wird.

Der Hochladevorgang wird in zwei Schritten bedient. Zuerst ist ein Dateiojekt aus den übertragenen Daten zu erzeugen:

```
Dim File0 As HttpPostedFile = UploadFile0.PostedFile
```

Aus der Namensinformation, die immer den kompletten lokalen Pfad umfasst, wird nun der Dateiname ermittelt, denn der lokale Pfad des Benutzers ist völlig uninteressant.

Hochladen ausführen

```
FileName = Path.GetFileName (File0.FileName)
```

Unter diesem Namen erfolgt nun die Ablage mit `SaveAs` im aktuellen Pfad, bestimmt durch das `TreeView`-Element auf Seite eins:

```
File0.SaveAs (GetCurrentPath & " " & DirView.SelectedNodeIndex + "\" + FileName)
```



Abbildung 7.26: Bestätigung eines erfolgreichen Hochladevorgangs



Beachten Sie beim Entwurf von komplexen Formularen mit mehreren Seiten, dass aus Sicht der Übertragung der Daten nur eine Seite existiert. *MultiPage* erzeugt nur einen gestalterischen Mehrseiteneffekt, der die Benutzerführung verbessert. Intern gibt es nur eine Seite und nur ein Formular.

7.6 Kundenspezifische Steuerelemente (Custom Controls)

Kundenspezifische Steuerelemente kapseln nicht HTML-Code, sondern realisieren völlig neue Steuerelemente rein programmtechnisch. Sie sind nicht sehr leicht zu erstellen, dafür kann man sie als Assembly weitergeben und seinen Quellcode so schützen.

Die Nutzung dieser Technik ist interessant, wenn die bereits enthaltenen Steuerelemente nicht ausreichend sind, aber eine vergleichbar komplexe Funktionalität gebraucht wird und eine mehrfache Verwendung möglich oder erforderlich ist.

Bevor Sie sich damit auseinander setzen, sollten Sie die Nutzung von eingebauten Steuerelementen vollständig verstanden haben.



Die Dateien zu dem hier vorgestellten Projekt finden Sie unter *vbnet-controlprojekt* (Testprojekt für Steuerelement) bzw. *vbnetcontrollib* (Definition der Steuerelemente) im Installationszweig der CD.

7.6.1 Grundlagen

Die Technik der kundenspezifischen Steuerelemente basiert auf zwei verschiedenen Verfahren. Zum einen können Sie vorhandene Steuerelemente – ähnlich wie die benutzerdefinierten – zusammenfassen, als Assembly kompilieren und dann eigenen oder fremden Anwendungen bereit stellen. Das hat bereits den Vorteil, dass Sie den Quellcode nicht weiterreichen müssen. Zum anderen können Sie Steuerelemente auch vollständig im Code erstellen. Das ist aufwändiger, aber auch flexibler. Der HTML-Code wird hier vollständig vom Programm erzeugt. Auch hier wird eine Assembly erzeugt und diese kann von anderen Entwicklern genutzt werden, um das Steuerelement in eigene Applikationen einzubauen.

In beiden Fällen müssen Sie auf ein Werkzeug verzichten: Der Visual Studio .NET Designer lässt sich zum Entwurf von kundenspezifischen Steuerelementen nicht einsetzen. Der Code-Editor ist nichtsdestotrotz eine hervorragende Hilfe. Es werden deshalb hohe Ansprüche an Ihr visuelles Vorstellungsvermögen gestellt, wenn Sie Steuerelemente entwerfen sollen, die eine grafisch anspruchsvolle Ausgabe haben.

Die hier gezeigten Techniken zur Entwicklung derartiger Steuerelemente basieren teilweise auf der Nutzung des Visual Studio .NET. Natürlich lassen sich alle Beispiele auch ohne Studio nachvollziehen, allerdings fehlen die Vorlagen und einige Codes müssen von Hand eingegeben werden.

Visual Studio .NET

7.6.2 Zusammengesetzte kundenspezifische Steuerelemente

Zusammengesetzte kundenspezifische Steuerelemente fassen mehrere bereits in ASP.NET verfügbare Elemente und eigenen Code zusammen und können als separate Assembly weitergegeben oder in eigenen Anwendungen genutzt werden.

Erstellung eines Projekts »Web-Steuerelementbibliothek«

Um ein neues zusammengesetztes kundenspezifisches Steuerelement zu entwerfen, nutzen Sie in Visual Studio .NET die Projektvorlage WEB-STEUELEMENTBIBLIOTHEK. Gehen Sie folgendermaßen vor:

1. Im PROJEKTMAPPEN-EXPLORER klicken Sie mit der rechten Maustaste auf die aktuelle Projektmappe.
2. Wählen Sie nun HINZUFÜGEN und dann NEUES PROJEKT.
3. In der Liste der Vorlagen wählen Sie WEB-STEUELEMENTBIBLIOTHEK.
4. Vergeben Sie einen Namen und wählen Sie den Speicherort. Im Beispiel wurde als Name VBNETCONTROLLIB verwendet.
5. Klicken Sie auf OK.
6. Es entsteht eine bereits mit einigem Code gefüllte Klassendefinition. Suchen Sie die folgende Zeile im Code (etwa Zeile 20):

```
output.Write([Text])
```

Ersetzen Sie diese nun durch Folgendes:

```
output.Write("<b>" + [Text] + "</b>")
```

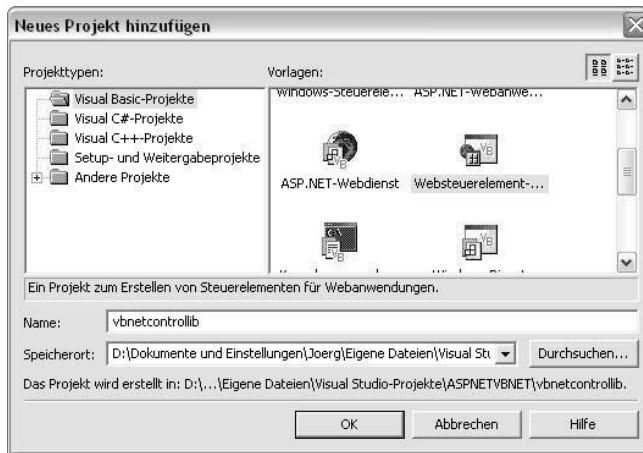


Abbildung 7.27: Ein neues Projekt für eine Steuerelemente-Bibliothek

Lassen Sie sich nicht von dem Begriff »Websteuerelementbibliothek« irritieren. Die Technik wird auch verwendet, wenn Sie nur ein einziges Element realisieren, also von einer Bibliothek weit und breit keine Spur ist.

Damit existiert bereits eine recht umfangreiche Vorlage, automatisch mit *WebCustomControl1.vb* benannt. Lassen Sie den Namen vorerst unverändert.

```
Imports System.ComponentModel
Imports System.Web.UI

<DefaultProperty("Text"),
    ToolboxData("<{0}:WebCustomControl1 runat=server>
        </{0}:WebCustomControl1>")>
Public Class WebCustomControl1
    Inherits System.Web.UI.WebControls.WebControl

    Dim _text As String

    <Bindable(True), Category("Appearance"), DefaultValue("")>
    Property [Text]() As String
        Get
            Return _text
        End Get
```



```

        Set(ByVal Value As String)
            _text = Value
        End Set
    End Property

    Protected Overrides Sub Render(ByVal output ←
                                   As System.Web.UI.HtmlTextWriter)
        output.Write("<b>" + [Text] + "</b>")
    End Sub

End Class

```

Listing 7.33: Die Vorlage für das die Steuerelemente-Bibliothek mit Änderungen (WebCustomControl1.vb)

Vermutlich werden Sie einige Elemente dieses Codes sehen, die Sie noch nicht kennen und die auch an anderer Stelle im Buch nicht behandelt werden. Es handelt sich bei den oberhalb der Definition stehenden Anweisungen in spitzen Klammern um so genannte Attribute. Dies sind hier ganz spezielle Attribute, die nämlich zur Entwurfszeit wirken. Der Designer in Visual Studio hilft Ihnen zwar nicht beim Entwurf der Bibliothek, wohl aber bei der späteren Verwendung der Steuerelemente in einer normalen Applikation. Damit er dann etwas mit dem Code anfangen kann, werden mit Attributen quasi »parallel« zum normalen Programm Anweisungen über die Behandlung zur Entwurfszeit mitgegeben.

Aber auch im »normalen« Code-Teil sind einige Besonderheiten zu finden. Es ist zu beachten, dass die Klasse von `WebControl` erbt. Damit steht eine gewisse Basisfunktionalität zur Verfügung. Vor allem aber werden damit die Methoden festgelegt, die für die Ausgabe der Werte verantwortlich sind. Die erste Methode, die überschrieben werden soll, ist `Render`.

Was im Code steht

Die Klasse `WebCustomControl1` definiert letztlich ein Steuerelement und enthält bereits eine Eigenschaft `Text`, die schreib- und lesbar ist, und eine Überschreibung der Methode `Render`. Diese Methode ist dafür verantwortlich, den vom Steuerelement erzeugten Code auszugeben, also während der Abarbeitung dem Ausgabestrom zu übergeben.

Erstellen Sie nun das Projekt:

Erstellen

1. Wählen Sie dazu **EIGENSCHAFTEN** im Kontextmenü. Ändern Sie den Eintrag `STAMMNAMESPACE` in `Addison.VBNet.Controls`. Dies ist nicht zwingend notwendig, erleichtert aber die An-

wendung und sorgt für einen einheitlichen Namensraum, der für alle Buchprojekte mit *Addison.VBNet* beginnt.

2. Wählen Sie im Kontextmenü den Eintrag ERSTELLEN. Die Anwendung wird kompiliert und die Assembly erstellt.
3. Prüfen Sie die Angaben zum Assemblynamen. Sie werden diesen im nächsten Abschnitt benötigen. Wählen Sie dazu EIGENSCHAFTEN im Kontextmenü.

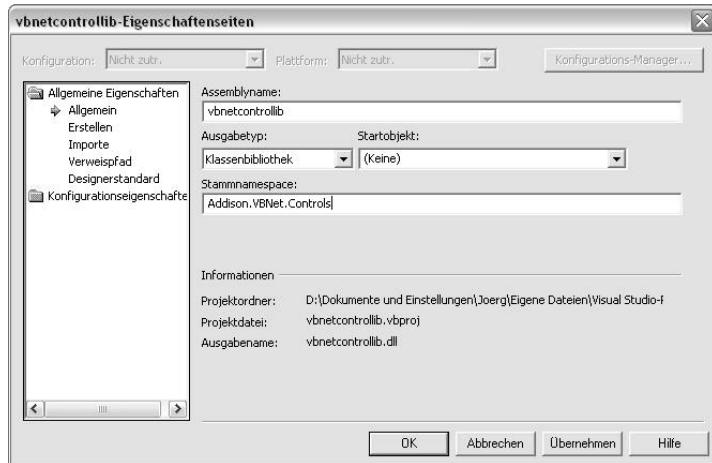


Abbildung 7.28: Prüfen der Projekteigenschaften

Nun sollten Sie ein Testprojekt erstellen, um das Steuerelement testen zu können.

Erstellung eines Testprojekts

Das soeben erstellte Projekt enthält Klassenbibliotheken und erzeugt daraus bei der Übersetzung eine Assembly. Sie können diese Assembly nicht alleine ablaufen lassen. Das Steuerelement wird erst sichtbar, wenn es von einer WebForm aus verwendet wird. Diese Webform sollte in einem weiteren Projekt enthalten sein. Erstellen Sie deshalb zusätzlich noch ein Projekt vom Typ ASP.NET-WEBANWENDUNG mit dem Namen *vbcontrolprojekt*.



Abbildung 7.29: Erstellung eines Projekts ASP.NET-WEBANWENDUNG

Die hier verwendeten Namen stimmen exakt mit denen der fertigen Projekte auf der Buch-CD überein. Sie können sich die Programme natürlich auch von der CD laden. Es ist aber durchaus zu empfehlen, ein paar elementare Schritte selbst zu vollziehen.



Wenn Sie die Projekte von der CD verwenden, beachten Sie bitte, dass in allen Fällen der Namensraum `Addison.VBNet.Controls` heißt. Dies dient der besseren Organisation aller im Buch verwendeten Programme.

Nun sollten Sie einen ersten Versuch unternehmen, das Steuerelement aus der Bibliothek zu verwenden, auch wenn es noch keine Funktion hat, denn es geht hier nur um den prinzipiellen Ablauf des Kompilierens und Verwendens.

Gehen Sie dazu folgendermaßen vor:

1. Öffnen Sie das automatisch generierte Formular `WebForm1.aspx`.
2. Tragen Sie folgende Direktive im Kopf der Seite ein:

```
<%@ Register TagPrefix="cc"
    Namespace="Addison.VBNet.Controls"
    Assembly="VBNetControlLib" %>
```

Die Direktive `@Register` wird in Abschnitt 10.2.4, »Die Direktive `@Register`« ab Seite 959 beschrieben.

3. Erzeugen Sie dann das Steuerelement selbst durch Eingabe des folgenden Codes innerhalb des <form>-Tags der Seite:

```
<cc:WebCustomControl1 id="ccTest" runat="server" Text="Mein
Control"/>
```

4. Jetzt muss die Assembly der Steuerelemente-Bibliothek im Testprojekt bekannt gemacht werden. Dazu wird eine Referenz über den Eintrag VERWEISE hinzugefügt. Im Kontextmenü wählen Sie VERWEIS HINZUFÜGEN. Im folgenden Dialog wählen Sie die Registerkarte PROJEKTE und dort das entsprechende Projekt *vbnetcontrollib*.

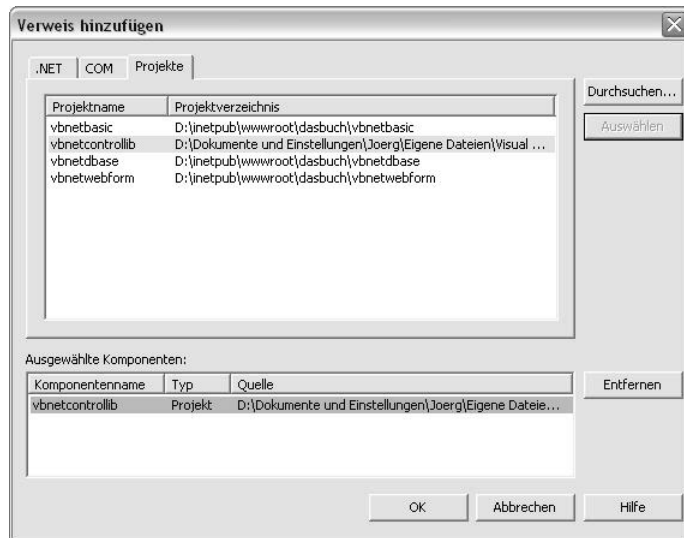


Abbildung 7.30: Verweis auf die Assembly der Bibliothek hinzufügen

5. Wechseln Sie in die Entwurfs-Ansicht. Sie sollten dort das Steuerelement mit dem Text bereits sehen. Markieren Sie es und öffnen Sie dann mit **[F4]** das Eigenschaftfenster (siehe Abbildung 7.31).
6. Erstellen Sie das Projekt jetzt und versuchen Sie es anschließend zu starten. Der Text sollte fett erscheinen, denn die Ausgabe in Listing 7.33 wurde um -Tags erweitert:

Mein Control

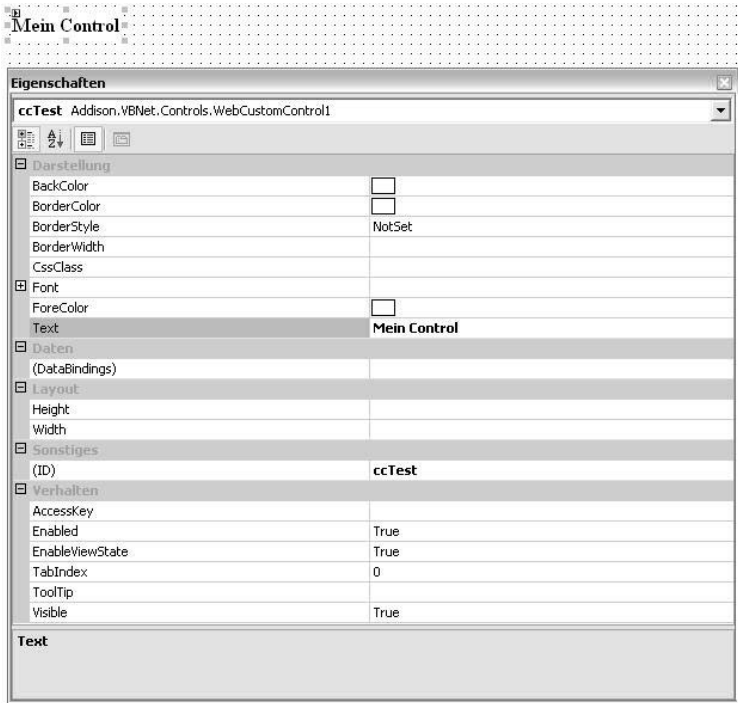


Abbildung 7.31: Ansicht des Steuerelements im Designer

Erstaunlich ist an dem erreichten Effekt weniger die Leistungsfähigkeit des Steuerelements – außer den Text fett zu schreiben, beherrscht es kaum sinnvolle Funktionen. Interessanter ist die Ausstattung mit den Basiseigenschaften, wie sie das Eigenschaftfenster in Abbildung 7.31 zeigt. Die Ausstattung mit Eigenschaften alleine reicht jedoch noch nicht aus. Sie funktionieren noch nicht alle. An diesem Punkt werden Sie mit einer Basiseigenschaft der kundenspezifischen Steuerelemente vertraut gemacht: Sie sind als Steuerelementeentwickler für die gesamte Gestaltung der Ausgabe selbst verantwortlich. Das gilt natürlich auch für die Erhaltung des Anzeigestatus. Wenn Sie dem Steuerelement dynamisch einen Wert zuweisen und dann nach dem erneuten Aufruf der Seite auswerten und verändern, verfällt es immer wieder auf den Anfangszustand. Auch hier müssen sie den Code selbst schreiben.

**Basiseigen-
schaften
verwenden**

Die Gestaltung des Steuerelements kontrollieren

Als Nächstes soll das Problem der Gestaltung gelöst werden. Wenn Sie ein zusammengesetztes kundenspezifisches Steuerelement entwerfen, besteht dieses aus weiteren, eingebauten Steuerelementen. Solange Sie ausreichend fertige Elemente finden, ist dieser Weg optimal, denn er erspart eine Menge Arbeit. Basis dieser Funktionalität bildet die Methode `CreateChildControls`, die die Basisklasse `WebControl` zur Verfügung stellt und die Sie überschreiben müssen, um die Funktionalität zu erweitern.

Erweiterung des Elements

Im Beispiel soll das Element nun erweitert werden. Für den Entwurf wird ein neues Steuerelement erzeugt. Es wird dazu neben der bereits vorhandenen Klasse `WebCustomControl1` des Projekts `vbnetcontrollib` eine weitere Klasse erzeugt und wieder von `WebControl` abgeleitet. Gehen Sie folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste auf das Websteuerelementbibliotheks-Projekt `vbnetcontrollib`.
2. Wählen Sie im Menü HINZUFÜGEN und dann NEUES ELEMENT HINZUFÜGEN. In der folgenden Auswahl klicken Sie auf WEBSTEUERELEMENT (BENUTZERDEFINIERT).
3. Geben Sie dem Element den Namen `multibox.vb`.

Das Steuerelement soll jetzt aus einem Eingabefeld, einer Schaltfläche und einer Ausgabemarke bestehen, wobei der eingegebene Text formatiert in der Ausgabe erscheint. Schreiben Sie die Klasse erstmal gemäß folgendem Listing:

```
Imports System.ComponentModel
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("Text"), <
    ToolboxData("<{0}:multibox runat=server></{0}:multibox>")> <
    Public Class multibox
    Inherits System.Web.UI.WebControls.WebControl

    Dim ccTextBox As TextBox = New TextBox()
    Dim WithEvents ccButton As Button = New Button()
    Dim ccLabel As Label = New Label()
    Dim ccPanel As Panel = New Panel()
    Dim _text As String

    <Bindable(True), Category("Appearance"), DefaultValue("")> <
    Property [Text]() As String
```

```

        Get
            Return _text
        End Get

        Set(ByVal Value As String)
            _text = Value
        End Set
    End Property

    Protected Overrides Sub CreateChildControls()
        ccTextBox.Width = Unit.Pixel(150)
        Controls.Add(ccTextBox)
        ccButton.Text = "Anzeigen"
        ccButton.Width = Unit.Pixel(150)
        Controls.Add(ccButton)
        Controls.Add(New LiteralControl("<br/>"))
        Controls.Add(New LiteralControl("<b>"))
        ccPanel.Controls.Add(ccLabel)
        Controls.Add(ccPanel)
        Controls.Add(New LiteralControl("</b>"))
    End Sub

    Protected Overrides Sub Render(ByVal output As
HtmlTextWriter)
        MyBase.Render(output)
    End Sub

End Class

```

Listing 7.34: Erweiterung um ein zusammengesetztes Steuerelement (erster Schritt)

Wird das Projekt neu erstellt, können Sie das Element schon im **Anwendung** HTML-Code einsetzen:

```
<cc:MultiBox id="ccBox" runat="server" />
```

Beachten Sie, dass der Name der Klasse (*MultiBox*) den Namen des Elements bestimmt. Die Registrierung der Assembly, die bereits im vorherigen Abschnitt mit `@Register` erfolgte, ist dagegen nicht erneut erforderlich. Freilich bleibt das Präfix damit unverändert: »cc«.

Wenn Sie das Projekt starten, sehen Sie die Elemente bereits, allerdings ohne jede Funktion. Auch die Ansicht im Designer ist unbefriedigend, es ist nämlich überhaupt nichts zu sehen:

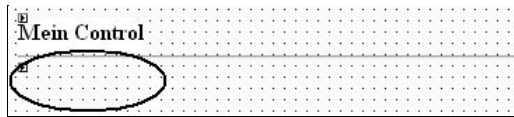


Abbildung 7.32: Die Designer-Ansicht bleibt vorerst leer.

Bei der Ausführung werden die Elemente, die in `CreateChildControls` erzeugt werden, ausgegeben, sind aber noch ohne jede Funktion:



Abbildung 7.33: Die Anzeige erfolgt, aber ohne Funktion.

Das Steuerelement muss nun mit den Eigenschaften und Methoden erweitert werden, die die nötige Funktionalität bieten. Im Beispiel ist das recht einfach. Zuerst wird eine Eigenschaft benötigt, die den in die Textbox eingegebenen Text erfasst. Eine Methode schreibt den Text dann – fett formatiert – in die Ausgabe. Wie das erfolgt, zeigt das folgende Listing, das die Klasse *MultiBox* erweitert. Der Einfachheit halber erweitern Sie einfach die bereits vorgegebene Eigenschaft `[Text]` und schreiben den Rest darunter:

```
<Bindable(True), Category("Appearance"), DefaultValue("")> ➔
Property [Text]() As String
    Get
        Me.EnsureChildControls()
        Return ccTextBox.Text
    End Get
    Set(ByVal Value As String)
        Me.EnsureChildControls()
        ccTextBox.Text = Value
    End Set
End Property

Property Output() As String
    Get
        Me.EnsureChildControls()
        Return ccLabel.Text
    End Get
    Set(ByVal Value As String)
        Me.EnsureChildControls()
        ccLabel.Text = Value
    End Set
End Property
```



```

        End Set
    End Property

    Public Sub SetLabel()
        If ccTextBox.Text Is String.Empty Then
            ccLabel.Text = ccTextBox.Text
        Else
            ccLabel.Text = "Leeres Feld"
        End If
    End Sub

```

Auch hiernach erscheint das Element weder im Designer noch führt es die Aktion aus. Das war auch nicht zu erwarten, denn die Methode *SetLabel*, die die Zuweisung des Textes aus der *TextBox* ins *Label*-Steuerelement übernimmt, wird nirgendwo aufgerufen. Sie müssen dazu die Schaltfläche mit einem Ereignis und dieses mit einer Ereignisbehandlungsmethode verbinden.

Eine wichtige Voraussetzung für neue Steuerelemente ist die Integration in die Steuerelementehierarchie der Seite. Dies erfolgt über eine eindeutige ID. Auch wenn Sie die Namen sorgfältig vergeben, kann ASP.NET nicht sicher feststellen, ob die Vergabe konfliktfrei ist und das Steuerelement deshalb nicht verarbeiten. Um dies sicherzustellen, gibt es eine spezielle Schnittstelle, *INamingContainer*. Wird diese implementiert, funktioniert auch das übliche PostBack-Verhalten auf Anhieb. Ergänzen Sie dazu im Kopf der Klasse Folgendes:

```

Public Class multibox
    Inherits System.Web.UI.WebControls.WebControl
    Implements INamingContainer

```

Diese Verknüpfung von Ereignis über ein Delegat erfolgt entweder in der üblichen Form, mit dem Schlüsselwort *Handles*, oder durch manuelles Hinzufügen des Eventhandlers. Fügen Sie zuerst die folgende Methode ein. Damit wird nun jedes Mal, wenn die Schaltfläche angeklickt wird, die Methode *ccButton_Click* aufgerufen:

```

Private Sub ccButton_Click(ByVal sender As Object, ←
                           ByVal e As EventArgs)
    SetLabel()
End Sub

```

Hier passiert nichts weiter als der Aufruf der Methode *SetLabel*, die die Zuweisung vornimmt.

Steuerelemente in der Hierarchie

Umgang mit Ereignissen

Am Ende der Methode `CreateChildControls`, die immer zuerst aufgerufen wird, ist dann noch der Handler anzufügen (`Click` ist das Ereignis, auf das hier reagiert wird):

```
AddHandler ccButton.Click, AddressOf ccButton_Click
```

Nun können Sie *MultiBox* schon mal testen. Erstellen Sie das Projekt neu und testen Sie es dann:

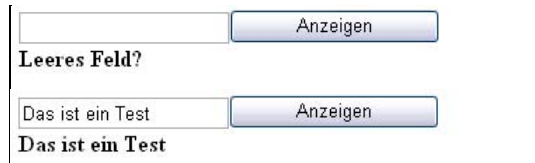


Abbildung 7.34: Reaktion auf leere und gefüllte TextBox
(oben bei leerem Feld, unten mit Text)

Praktische Erweiterung des Steuerelementes

Nun sollten Sie auch etwas an den späteren Anwender des Steuerelementes denken. So wäre es sicher bei einigen Elementen sinnvoll, wenn die Reaktion auf Ereignisse auch außerhalb der Assembly programmiert werden kann. Dazu zuerst ein Blick auf die Anwenderseite. Nachfolgend wurde das Steuerelement um ein `OnClick`-Ereignis erweitert:

```
<cc:MultiBox id="ccMulti" runat="server"
    Text="Ihr Eintrag" OnClick="SetMark_Click"/>
```

In der Code-Datei *WebForm1.aspx.vb* wird nun die Ereignisbehandlungsmethode definiert:

```
Public Sub SetMark_Click(ByVal sender As Object, ByVal e As
EventArgs)
    ccMulti.Text += "+"
End Sub
```

Die Reaktion ist sehr einfach: An den aktuellen Text in der *TextBox* wird ein Pluszeichen angehängt. Leider reicht der Wunsch nicht aus, das Steuerelement muss die Behandlung dieses Ereignisses lernen. Es muss nämlich ein Weg gefunden werden, das bereits an das Steuerelement weitergeleitete Ereignis wieder herauszubekommen und mit dem externen Delegaten zu verbinden. Dazu erweitern Sie die Klasse *MultiBox* folgendermaßen:

```
Public Event Click As EventHandler
```

```
Protected Overridable Sub OnClick(ByVal e As EventArgs)
    RaiseEvent Click(Me, e)
End Sub
```

Listing 7.35: Erweiterung um einen externen Delegaten (Ausschnitt aus multibox.vb)

Außerdem ist die Methode `onButton_Click` folgendermaßen zu ergänzen:

```
Public Sub ccButton_Click(ByVal sender As Object, ByVal e As
EventArgs)
    SetLabel()
    OnClick(e)
End Sub
```

Den Kern bildet die Definition eines Ereignisses, hier handelt es sich um das Abfangen des Mausclicks, das den Namen `Click` hat:

```
Public Event Click As EventHandler
```

Dann wird das Ereignis, das original verwendet wird, überschrieben:

```
Protected Overridable Sub OnClick(ByVal e As EventArgs)
```

Das Weiterreichen an die vom Benutzer des Steuerelementes definierte Methode erfolgt durch einen direkten Aufruf. VB.NET erledigt dies durch die Anweisung `RaiseEvent`; als »Absender« dient die Steuerelementeklasse selbst, wodurch der Anwender Informationen darüber erhält:

```
RaiseEvent Click(Me, e)
```

Zuletzt ist noch sicherzustellen, dass bei Eintritt des Klickereignisses, das primär in `ccButton_Click` landet, neben dem Aufruf von `SetLabel` auch das externe Ereignis bedient wird.

```
OnClick (e)
```

Wenn die Ereignisverarbeitung keine Ereignisargumente verarbeitet, können Sie `e` auch durch `EventArgs.Empty` ersetzen. Im Beispiel wäre das ausreichend, aber möglicherweise planen Sie ja Erweiterungen.

Wenn Sie die Reihenfolge vertauschen, würde das externe Ereignis primär ablaufen. Im Ergebnis der Änderungen taucht nun nach dem Senden des Formulars ein Pluszeichen auf:

**Die Verarbeitung
externer
Ereignisse**



Abbildung 7.35: Behandlung interner und externer Ereignisse

Den Visual Studio .NET-Designer unterstützen

Weitere praktische Änderungen betreffen den Designer. Ergänzen Sie folgendes Attribut für die gesamte Klasse *MultiBox* (dies sollte in einer Zeile stehen oder mit `_`-Trennzeichen getrennt werden):

```
<DefaultProperty("Text"), ␣
ToolboxData("<{0}:multibox runat=server></{0}:multibox>"), ␣
DefaultEvent("Click")> _
Public Class multibox : Inherits WebControl
```

Nun erkennt der Designer, dass ein bevorzugtes Klickereignis definiert wurde. Kompilieren Sie das Steuerelement wieder und wechseln Sie dann in den Designer des Kontrollprojekts. Wenn Sie das Steuerelement nun im Designer doppelklicken, wird eine Ereignisbehandlungsmethode dazu automatisch erstellt und der Delegat dazu verknüpft. Das sieht etwa folgendermaßen aus:

```
Private Sub ccMulti_Click(ByVal sender As System.Object, ␣
                        ByVal e As System.EventArgs) ␣
    Handles ccMulti.Click

End Sub
```



Die beschriebene Technik ist nur für die Designerunterstützung (Visual Studio .NET-Entwurfsmodus) erforderlich. Ohne IDE nutzen Sie das Attribut `AutoEventWireUp` der `Page`-Direktive und öffentliche Methoden bzw. das Schlüsselwort `Handles`, um Ereignisse direkt zu verarbeiten.

Wenn Sie diese Änderungen ausführen, sollten sie den Inhalt von `SetMark_Click` jetzt nach `ccBox_Click` kopieren und `SetMark_Click` löschen.

Teil der Gestaltungsunterstützung des Designers ist nun noch eine Reaktion auf typische Attribute. So erbt die Klasse von `WebControl` unter anderem die Eigenschaften `Width` und `Height`. Wenn Sie im Designer das Steuerelement vergrößern, werden die dazu passenden Attribute automatisch erzeugt:

```
<cc:MultiBox id="ccMulti" runat="server"
    Text="Ihr Eintrag"
    Width="466px" Height="139px" />
```

Natürlich passiert in der Anzeige des laufenden Programms noch nichts. Ihr Steuerelement muss erst lernen, damit umzugehen. Die Steuerung der Ausgabe erfolgt in der Methode `Render`, die bislang keine Aufgaben hatte. Die Definition, was am neuen Steuerelement nun Breite und Höhe darstellt, müssen Sie dagegen selbst treffen. Im Beispiel wird nun das bislang funktionslose `Panel`-Steuerelement aktiviert. Dies erzeugt nämlich ein `<div>`-Tag, das sich gut zur Definition einer Fläche mit fester Ausdehnung eignet. Der gesamte Effekt wird durch folgende Version der `Render`-Methode erreicht:

```
Protected Overrides Sub Render(ByVal output As HtmlTextWriter)
    ccTextBox.Width = Unit.Parse(CInt(Me.Width.Value / 2))
    ccButton.Width = Unit.Parse(CInt(Me.Width.Value / 2))
    Dim i As Integer = (Me.Height.Value -
        Math.Max(ccButton.Height.Value,
        ccTextBox.Height.Value)).ToString()
    ccButton.Style("margin-bottom") = i
    ccTextBox.Style("margin-bottom") = i
    MyBase.Render(output)
End Sub
```

Listing 7.36: Zugriff auf Standardeigenschaften zur Gestaltung

Dieser Teil ist eine nähere Betrachtung wert. Zuerst erfolgt der Zugriff auf `Width` über die Standardeigenschaft:

Wie es funktioniert

```
Me.Width.Value
```

Diese Eigenschaft gibt es, weil sie in der Basisklasse `WebControl` definiert ist. Sie können eigene Eigenschaften ergänzen, indem Sie das mit *Text* gezeigte Verfahren verwenden. Im Beispiel wird nun Folgendes definiert: Die Breite führt zu einer gleichmäßigen Ausdehnung der Steuerelemente `TextBox` und `Button`, wobei die Aufteilung 50:50 ist. Die Höhe der Elemente bleibt dagegen unverändert. Der Text dagegen soll am unteren Rand der durch die Angabe `Height` bestimmten Höhe stehen. Dazu wird eine Stil-Eigenschaft benutzt: `margin-bottom`. Durch Zuweisung an die Stil-Kollektion führt der errechnete Wert zu einem Abstand der oberen Steuerelemente (Schaltfläche und Textbox) zum Text, sodass der Text immer exakt am unteren Rand des Panels erscheint.

Sie können nach der erneuten Übersetzung des Steuerelementes dies bereits im Designer ausprobieren. Das funktioniert, weil Visual Studio .NET per Reflektion auf die Klasse zugreift und das Steuerelement schon im Entwurfsmodus zur korrekten Darstellung nutzt. Wenn Sie das Panel mit der Maus aufziehen, werden mit jedem neuen Darstellungsschritt die Positionen der Elemente neu berechnet und die Darstellung angepasst.



Abbildung 7.36: Verschiedene Größen des Steuerelements im Designer und im Browser

Vorteile bei der Weitergabe

Die gezeigte Technik zum Erstellen eigener Steuerelemente hat, neben der großen Flexibilität beim Eingriff in die Erstellung auch den Vorteil, dass Sie nur die im `\bin\release`-Verzeichnis abgelegte Assembly weitergeben müssen, um das Steuerelement verwenden zu können.



Solange Sie im Debug-Modus arbeiten, liegt die Assembly unter `\bin\debug`. Ändern Sie die Eigenschaften des Projekts auf RELEASE, um es endgültig freizugeben.

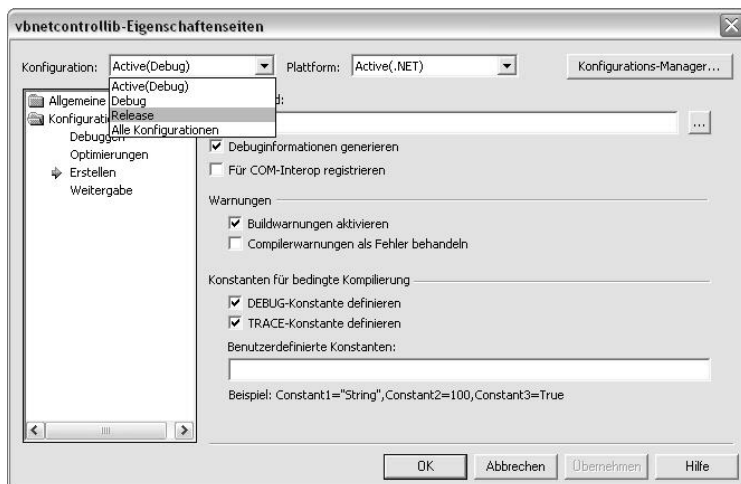


Abbildung 7.37: Umschalten des Projekts auf RELEASE

Wie gezeigt wurde, kann der Anwender die standardmäßig vorhandenen und von Ihnen hinzugefügten Eigenschaften nutzen, eigene Ereignisse anhängen und das Element so verwenden, wie er es von den eingebauten gewohnt ist.

Wenn Sie nun ernsthaft darüber nachdenken, Ihre Steuerelemente weiterzugeben, dann ist ein Blick auf die Programmierung mit Design-Time-Attributen angebracht. Einige Tipps, wie Sie das fertige Werk ins Visual Studio .NET integrieren, finden Sie in Abschnitt 7.6.5, »Die Integration ins Visual Studio« ab Seite 738.

7.6.3 Erweiterte Entwicklung eigener Steuerelemente

In diesem Abschnitt geht es um die Entwicklung eigener Steuerelemente von Grund auf, also einschließlich der Ausgabe des HTML-Codes, der Erhaltung des Anzeigestatus und der Verarbeitung der *PostBack*-Ereignisse. Im vorhergehenden Abschnitt hat das bereits funktioniert, weil fertige Steuerelement nur zusammengefasst wurden und diese die nötige Funktionalität bereits mitbrachten.

Kontrolle des Ausgabeverhaltens eines Steuerelements

Bislang wurde die Methode `Render` zwar gezeigt, aber praktisch wurde keine Manipulation des Ausgabestromes vorgenommen,

weil die enthaltenen Codes auf fertigen Steuerelementen basierten. Tatsächlich können Sie jedoch mit `Render` eigene HTML-Codes erzeugen und ausgeben. Die Methode erwartet die Übergabe eines Parameters vom Typ `HtmlTextWriter`. Über diesen besteht nun eine Zugriffsmöglichkeit auf den Ausgabedatenstrom. Dazu dienen verschiedene Methoden, die in der folgenden Tabelle zusammengefasst wurden:

Methoden	Beschreibung
<code>AddAttribute</code>	Fügt dem nächsten erzeugten HTML-Element ein Attribut hinzu
<code>RenderBeginTag</code>	Beginnt ein neues HTML-Element und stellt das öffnende Tag dar, die Ausgabe erfolgt jedoch noch nicht sofort, sodass weitere Manipulationen möglich sind.
<code>RenderEndTag</code>	Beendet die Erstellung eines HTML-Elementes und stellt das schließende Tag dar, das Tag wird jedoch noch nicht geschrieben.
<code>Write</code>	Schreibt einfachen Text in die Ausgabe
<code>WriteAttribute</code>	Erzeugt ein Attribut für ein HTML-Tag
<code>WriteBeginTagAddAttribute</code>	Erzeugt ein öffnendes Tag und schreibt es sofort
<code>WriteEndTag</code>	Erzeugt ein schließendes Tag und schreibt es sofort
<code>WriteFullBeginTag</code>	Schreibt ein öffnendes Tag, fügt jedoch die schließende spitze Klammer sofort ein
<code>WriteLine</code>	Erzeugt einfachen Text, fügt aber einen Zeilenumbruch an

Tabelle 7.1: Methoden der Klasse `HtmlTextWriter`

Mit diesen Methoden ausgestattet, steht der Erzeugung eigener HTML-Codes nichts mehr im Wege.

Anwendungsbeispiel

Wenn Sie viel mit Schaltflächen arbeiten, wird Ihnen eventuell aufgefallen sein, dass die Nutzung des Ereignisses `OnClick` in JavaScript problematisch ist, denn dieses Attribut wird auch für die Auslösung von serverseitigen Ereignissen benötigt. Deshalb scheidet der Einsatz von `Button` aus, wenn gleichzeitig eigener JavaScript-Code eingesetzt werden soll. Das folgende Beispiel zeigt, wie Sie ein Steuerelement mit ganz spezifischen Eigenschaften entwerfen. Praktisch realisiert werden soll Folgendes:

- ▶ Ein Eingabefeld und eine Schaltfläche.
- ▶ Nach Klick auf die Schaltfläche wird eine Sicherheitsabfrage per JavaScript ausgelöst.
- ▶ Bestätigt der Benutzer, wird ein serverseitiges Klickereignis ausgelöst und der im Textfeld eingegebene Text gespeichert.
- ▶ Bestätigt der Benutzer nicht, wird das Textfeld gelöscht.
- ▶ Das Textfeld soll seinen Status in jedem Fall beim *PostBack* erhalten.

Das Element kann folgendermaßen formatiert werden:

```
<cc:JsBox runat="server" id="JsControl" ␣
    ButtonText ␣
    onClick="JsControl_Click">
    Hinweistext
</cc:JsBox>
```

Es soll keine Beschränkungen hinsichtlich des Inhalts (anstatt »Hinweistext«) geben, auch andere Steuerelemente müssen an dieser Stelle korrekt verarbeitet werden. Alternativ dazu soll ein Attribut `Caption` Verwendung finden, wenn nur einfacher Text benötigt wird:

```
<cc:JsBox runat="server" id="JsControl" ␣
    onClick="JsControl_Click" ␣
    Caption="Eingabe:"/>
```

Starten Sie wie bereits im letzten Abschnitt gezeigt mit einer neuen Klassendatei vom Typ `WEBSTEUERELEMENT (BENUTZERDEFINIERT)`.

Realisiert wird die benötigte Klasse *JsBox* im Projekt *vbnetcontrol-lib*. Zuerst wird die Klassendefinition und die `Render-Methode` vorgestellt:

```
Imports System.ComponentModel
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Collections.Specialized

<ParseChildren(False), _
    DefaultProperty("ButtonText"), _
    ToolboxData("<{0}:JsBox runat=server></{0}:JsBox>")> _
Public Class JsBox
    Inherits System.Web.UI.WebControls.WebControl
    Implements INamingContainer
```

```

Dim _buttontext As String
Dim _fieldtext As String
Dim _caption As String = String.Empty
Dim c As Control

<Bindable(True), Category("Appearance"), DefaultValue("")> _
Property [ButtonText]() As String
    Get
        Return _buttontext
    End Get

    Set(ByVal Value As String)
        _buttontext = Value
    End Set
End Property

<Bindable(True), Category("Appearance"), DefaultValue("")> _
Public Property Caption() As String
    Get
        Return _caption
    End Get
    Set(ByVal Value As String)
        _caption = Value
    End Set
End Property

<Bindable(True), Category("Appearance"), DefaultValue("")> _
Public Property FieldText() As String
    Get
        Return _fieldtext
    End Get
    Set(ByVal Value As String)
        _fieldtext = Value
    End Set
End Property

Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)
    If Caption = String.Empty Then
        For Each c In Controls
            c.RenderControl(output)
        Next
    End If
    output.Write(Caption)
    output.AddAttribute(HtmlTextWriterAttribute.Value, _
        FieldText)
    output.AddAttribute(HtmlTextWriterAttribute.Type, _
        "Text")
    output.AddAttribute(HtmlTextWriterAttribute.Id, _

```

```

        Me.UniqueID)
    output.AddAttribute(HtmlTextWriterAttribute.Name, _
        Me.UniqueID)
    output.RenderBeginTag("input")
    output.RenderEndTag()
    output.AddAttribute(HtmlTextWriterAttribute.Value, _
        ButtonText)
    output.AddAttribute(HtmlTextWriterAttribute.Type, _
        "Button")
    output.AddAttribute("onclick", _
        "javascript: if (confirm('Wirklich eintragen?')){
        + Page.GetPostBackEventReference(Me)
        + "}")
    output.RenderBeginTag("input")
    output.RenderEndTag()
End Sub

```

End Class

Die Attribute am Anfang unterstützen wieder das Erscheinungsbild im Designer. Ansonsten hat die Definition keinen Einfluss auf das Steuerelement selbst. Die Attribute fügen praktische zusätzliche Funktionen hinzu, auf die Visual Studio .NET über die Reflektion-Technik Zugriff hat. Es gibt Attribute zum Steuern des Designers (Design-Time-Attribute) und solche, die das Verhalten der gesamten Klasse modifizieren, ohne dass Sie den Code selbst ändern müssen. Von besonderem Interesse ist hier `ParseChildren(False)`. Dieses Attribut sorgt dafür, dass das neue Steuerelement als Container benutzt werden kann, der Inhalt selbst jedoch nicht verarbeitet wird. Stattdessen werden Inhalte ohne weitere Verarbeitung an die Standardmethode `RenderControl` weitergeleitet. Entfernen Sie das Attribut (der Standardwert ist `True`), erwartet das Steuerelement für enthaltene Tags entsprechende Eigenschaftsdefinitionen; auch dann, wenn es sich lediglich um einfaches HTML handelt.

Die Attribute

Dann folgt die Klassendefinition, abgeleitet von der Basisklasse `WebControl`. Zusätzlich werden drei Schnittstellen implementiert (davon schreiben Sie im Moment nur die erste hinein):

Klassendefinition

► `INamingContainer`

Diese Schnittstelle erlaubt den Zugriff auf den »Inhalt« des Elements, denn das Steuerelement soll optional auch als Container verwendet werden (`<cc:JsBox>...</cc:JsBox>`).

► `IPostBackEventHandler`

Damit das *PostBack*-Ereignis verarbeitet werden kann, wird diese Schnittstelle eingesetzt. Diese Schnittstelle wird erst im zweiten Schritt implementiert.

► `IPostBackDataHandler`

Damit die Formulardaten verarbeitet werden können, ist diese Schnittstelle nötig. Auch diese Schnittstelle wird erst im zweiten Schritt implementiert.

Die Ausgabe definieren

Nun folgt die Definition der Methode `Render`. Die Steuerelemente werden hier vollständig neu generiert. Es ist eine dankbare Aufgabe zur Übung, hier Attribute und Stildefinitionen anzufügen. Einige Hinweise dazu folgen im nächsten Abschnitt 7.6.4, »Stile und Attribute für Steuerelemente« ab Seite 736.

Die Methode beginnt mit der Untersuchung, ob das Attribut `Caption` leer war:

```
If Caption = String.Empty Then
```

Ist das der Fall, werden die Steuerelemente im Inneren durchlaufen:

```
For Each c In Controls
```

Jedes gefundene Element wird über seine eingebauten Methoden verarbeitet. Dazu wird die Methode `RenderControl` des jeweiligen Steuerelementes aufgerufen. Damit die Ausgabe korrekt verarbeitet werden kann, wird `RenderControl` die aktuelle Instanz von `HtmlTextWriter` übergeben:

```
c.RenderControl(output)
```

Dann wird die Überschrift ausgegeben. Das funktioniert immer, weil die Definition gleich die Zuweisung zu `String.Empty` enthält, das Objekt kann also nie leer (`Nothing`) sein:

```
output.Write(Caption)
```

Nun werden Attribute erzeugt. Die Sammlung der letzten Attribute wird beim Erzeugen des Tags automatisch verwendet. Für das Textfeld werden vier Attribute erzeugt.

Der Inhalt des Feldes wird durch `value` bestimmt:

```
output.AddAttribute(HtmlTextWriterAttribute.Value, FieldText)
```

`HtmlTextWriterAttribute` ist eine Aufzählung, die alle typischen Attribute für HTML-Tags enthält. Die Verwendung ist sicherer, als direkt die Zeichenkette anzugeben (was beim zweiten Tag dennoch gemacht wird, aber das dient nur der Demonstration), weil Sie Tippfehler vermeiden.

Da es sich um ein Textfeld handelt, wird der Typ »text« verwendet:

```
output.AddAttribute(HtmlTextWriterAttribute.Type, "Text")
```

Nun wird noch eine ID vergeben, wobei einfach die ID des eigenen Steuerelements verwendet wird:

```
output.AddAttribute(HtmlTextWriterAttribute.Id, this.UniqueID)
```

Gleiches gilt für den Namen, der unbedingt gebraucht wird, weil sonst die `Form`-Kollektion keine Daten des Textfeldes aus dem Formular enthält:

```
output.AddAttribute(HtmlTextWriterAttribute.Name, this.UniqueID)
```

Beachten Sie, dass Sie hier zusätzlich zur ID des Steuerelementes eine weitere Unterscheidung einführen müssen, wenn Sie mehr als ein Tag erzeugen, das Formulardaten enthalten soll. Das ist sehr leicht zu bewerkstelligen, wenn Sie einfach eine fortlaufende Nummer anhängen.



Nun wird das Tag für das Texteingabefeld erzeugt:

```
output.RenderBeginTag("input")
```

Da Eingabefelder keine Container sind, wird es sofort geschlossen:

```
output.RenderEndTag()
```

Der zweite Teil erzeugt nun eine ganz spezielle Schaltfläche. Denn zusätzlich zur Verarbeitung des Klickereignisses soll noch eine Bestätigung per JavaScript erfolgen. Drei Attribute werden benötigt. Während `value` und `type` unproblematisch sind, ist eine nähere Beschäftigung mit `onclick` angebracht. Hier wird zuerst eine Bestätigungsmeldung mit `confirm` erzeugt, eingebaut in eine `if`-Anweisung. `confirm` erzeugt eine Box mit den Schaltflächen OK und ABBRECHEN. Klickt er OK an, ist der Rückgabewert `true`, sonst `false`. Es kommt nun darauf an, bei der Bestätigung das korrekte `PostBack`-Ereignis auszulösen. Das ist normalerweise die `__doPostBack`-Funktion. Die Methode `GetPostBackEventReference` gibt genau den passenden Aufruf zurück:

**GetPostBack-
EventReference**

```

output.AddAttribute ("onclick", "
    javascript: if (confirm('Wirklich eintragen?'))
    {
        + Page.GetPostBackEventReference(this)
        + "; }")

```



Beim Test mit dem Framework 1.0 war Intellisense unter VB.NET nicht in der Lage, die Events des Page-Objekts zu erkennen und bot deshalb GetPostBackEventReference nicht an. Nach dem Kompilieren funktionierte das Programm jedoch problemlos. Mit C# trat derselbe Fehler nicht auf, vermutlich handelt es sich um einen Bug des VB.NET-Editors. Falls der Fehler bei Ihnen auftritt, versuchen Sie sich das neueste Service Pack zu beschaffen.

Am Ende wird auch das Tag für die Schaltfläche geschlossen und beendet:

```

w.RenderBeginTag ("input")

w.RenderEndTag ()

```

An dieser Stelle kann schon ein erster Versuch gestartet werden. Die Anwendung erfolgt wieder im Testprojekt, wo eine weitere Web Form *WebForm2.aspx* erzeugt wurde:

```

<%@ Page language="vb" Codebehind="WebForm2.aspx.vb"
    AutoEventWireup="false"
    Inherits="vbnetcontrolproject.WebForm2" %>
<%@ Register TagPrefix="cc"
    Namespace="Addison.VBNet.Controls"
    Assembly="VBNetControlLib" %>
<HTML>
  <HEAD>
    <title>WebForm2</title>
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <h1>Eigene Steuerelemente</h1>
    <form id="WebForm2" method="post" runat="server">
      <cc:JsBox runat="server" id="JsControl"
        FieldText="" ButtonText="Eintragen">
        <h3>Hier anmelden:</h3>
      </cc:JsBox>
    </form>
    <asp:Label Runat="server" ID="Eingabe" />
  </body>
</HTML>

```

Listing 7.37: Das Testprogramm (*WebForm2.aspx*)



Abbildung 7.38: Das Steuerelement mit der JavaScript-Box (nach Klick auf die Schaltfläche)

Natürlich hat die Klasse noch zu wenig Funktionalität. Der Status des Textfeldes bleibt nicht erhalten und das `Label`-Steuerelement wird auch nicht gefüllt. Sie müssen sich – das ist die wichtigste Erkenntnis bei der Steuerelementprogrammierung – tatsächlich um alle Einzelheiten selbst kümmern.

Fehlende Funktionen

Als Nächstes werden die Eigenschaften betrachtet. Es gibt hier zwei Attribute, die Verwendung finden:

```
FieldText="" ButtonText="Eintragen"
```

Entsprechend wurden mit demselben Namen zwei Eigenschaften in der Klasse `JsBox` definiert (das letzte Listing zeigte dies bereits):

```
Property [ButtonText]() As String
    Get
        Return _buttontext
    End Get

    Set(ByVal Value As String)
        _buttontext = Value
    End Set
End Property
```

Bei der Definition von `FieldText` wird keine Speicherung in der Klasse erfolgen, denn der Status soll nicht nur während der Erstellung, sondern über das Absenden des Formulars hinweg erhalten bleiben. Deshalb erfolgt die Speicherung des Wertes im Anzeigestatus der Seite. Die Definition wird deshalb gegenüber der ersten Variante entsprechend nachfolgendem Muster erweitert:

Status erhalten

```
Public Property FieldText() As String
    Get
        If Not ViewState("Text") Is Nothing Then
            Return ViewState("Text").ToString()
```

```

        Else
            Return Nothing
        End If
    End Get
    Set(ByVal Value As String)
        ViewState("Text") = Value
    End Set
End Property

```

Listing 7.38: Die Definition der Eigenschaft *FieldText*

Entfernen Sie außerdem die Klassenvariable *_fieldtext*, die im ersten Listing nur hilfsweise verwendet wurde, um den Compiler zu beruhigen.



Beachten Sie, dass Sie statt Text einen eindeutigen Namen wählen müssen, wenn Sie mehrere Eingabefelder speichern wollen.

Zum Abschluss der Eigenschaftendefinitionen noch ein Blick auf Caption:

```

Public Property Caption() As String
    Get
        Return _caption
    End Get
    Set(ByVal Value As String)
        _caption = Value
    End Set
End Property

```

Listing 7.39: Die Definition der Eigenschaft *Caption*

PostBack verarbeiten

Nun gilt es als Nächstes, die Formulardaten auch tatsächlich auszuwerten. Zuerst werden die beiden noch fehlenden Schnittstellen bekannt gemacht, um auf die Formulardaten zugreifen zu können. Ergänzen Sie dazu den Kopf der Klassendefinition folgendermaßen:

```

Public Class JsBox
    Inherits System.Web.UI.WebControls.WebControl
    Implements INamingContainer IPostBackDataHandler,
        IPostBackEventHandler

```


Damit ist natürlich die praktische Implementierung der Methode LoadPostData verbunden:

```
Public Function LoadPostData(ByVal Key As String, 
                             ByVal postData As
NameValueCollection) As Boolean _
    Implements IPostBackDataHandler.LoadPostData
    If Not ViewState("Text").ToString() = postData(Key) Then
        ViewState("Text") = postData(Key)
        Return True
    Else
        Return False
    End If
End Function
```

An diese Methode wird automatisch die zu diesem Steuerelement passende Kollektion von Formulardaten übergeben. Sie müssen nun nur noch prüfen, ob sich die Daten geändert haben – hier durch Vergleich mit dem Anzeigestatus:

**Verarbeitung der
Formulardaten**

```
If Not ViewState("Text").ToString() = postData(Key) Then
```

Ist das der Fall, wird der Anzeigestatus erneuert:

```
ViewState("Text") = postData(Key)
```

Die Methode hat Boolean als Rückgabewert. Wenn True zurückgegeben wird, löst ASP.NET anschließend das Ereignis RaisePostDataChangedEvent aus. Sie müssen die Ereignisbehandlungsroutine auch dann definieren, wenn Sie überhaupt nichts damit anfangen möchten – dies verlangt die Schnittstellendefinition:

```
Public Sub RaisePostDataChangedEvent() _
    Implements IPostBackDataHandler.RaisePostDataChangedEvent

End Sub
```

Bleibt als letzter Schritt die Verarbeitung des Klickereignisses. Diese Technik wurde schon im letzten Abschnitt gezeigt. Hier der nötige Code:

```
Public Sub RaisePostDataChangedEvent() _
    Implements IPostBackDataHandler.RaisePostDataChangedEvent
    ' Bleibt leer, wird hier nicht gebraucht
End Sub

Public Event Click As EventHandler

Public Overridable Sub OnClick(ByVal e As EventArgs)
```

```

        RaiseEvent Click(Me, e)
    End Sub

```

Fertig! Damit ist das Steuerelement fast fertig. Ergänzen Sie nun noch den Kopf der Klasse um ein weiteres Attribut:

```
DefaultEvent("Click")
```

Kompilieren Sie das Steuerelement nun. Ein Blick auf den Designer zeigt, dass auch das Klickereignis bekannt ist. Ein Doppelklick auf das Steuerelement erzeugt automatisch eine Ereignisbehandlungsmethode in *WebForm2.aspx.vb*. Ergänzen Sie diese um die hervorgehobene Zeile in der Mitte:

```

Private Sub JsControl_Click(ByVal sender As System.Object,   

                                ByVal e As System.EventArgs)   

    Handles JsControl.Click   

        Ausgabe.Text = JsControl.FieldText
    End Sub

```

Hier wird dann der Inhalt des Textfeldes dem Label-Steuerelement zugewiesen. Das hat den Vorteil, dass der Vorgang im eigenen Code manipuliert werden kann. Ebenso gut hätte man das im Steuerelement selbst kodieren können.

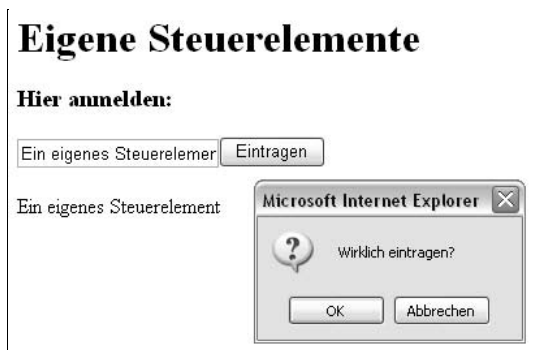


Abbildung 7.39: Das Steuerelement in Aktion

Ausblick Nun ist das Steuerelement fertig. Es realisiert eine flexible Kombination aus einer Überschrift, einem Eingabefeld und einer Sicherheitsabfrage mit JavaScript. Der Zugriff auf den Inhalt ist direkt und über Klickereignisse möglich. Der Anzeigestatus bleibt erhalten und harmonisiert mit beliebigem Inhalt. Was fehlt, sind gestalterische Elemente. Dies wurde im vorhergehenden Abschnitt zum Steuerelement *multibox* bereits besprochen. Kombinieren Sie

die dort gezeigten Techniken mit den hier vorgestellten, um zu wirklich leistungsfähigen und professionellen Steuerelementen zu gelangen. Im nächsten Abschnitt erfahren Sie zusätzlich zu den bereits gemachten Ausführungen weitere Tricks und Kniffe zu Stilen und Attributen von Steuerelementen.

Ein Blick auf den erzeugten HTML-Code zeigt, wie das neue Steuerelement mit ASP.NET harmonisiert:

Wie es intern funktioniert

```
<HTML>
  <HEAD>
    <title>WebForm2</title>
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <h1>Eigene Steuerelemente</h1>
    <form name="WebForm2" method="post" action="WebForm2.aspx"
id="WebForm2">
<input type="hidden" name="__VIEWSTATE" value="dDwtMTQ1MzYwNTQ2O
Tt0PDtsPGk8MT47aTwzPjs+02w8dDw7bDxpPDE+0z47bDx0PHA8cDxsPF
RleHQ7PjtsPEVpbIBlaWdlbmVzIFN0ZXVlcmVsZW11bnQ7Pj47Pjts7Pjs+Pjt
0PHA8cDxsPFRleHQ7PjtsPEVpbIBlaWdlbmVzIFN0ZXVlcmVsZW11bnQ
7Pj47Pjts7Pjs+Pjs+pg1MILApfeXltx0o1v0KFV1zbHY=" />

    <h3>Hier anmelden:</h3>
    <input value="Ein eigenes Steuerelement" type="Text"
      id="JsControl" name="JsControl" />
    <input value="Eintragen" type="Button"
      onclick="javascript: if
        (confirm('Wirklich eintragen?'))
        {
          __doPostBack('JsControl','');
        }" />
    <input type="hidden" name="__EVENTTARGET" value="" />
    <input type="hidden" name="__EVENTARGUMENT" value="" />
    <script language="javascript">
    <!--
function __doPostBack(eventTarget, eventArgument)
{
    var theform = document.WebForm2;
    theform.__EVENTTARGET.value = eventTarget;
    theform.__EVENTARGUMENT.value = eventArgument;
    theform.submit();
}
    // -->
  </script>
</form>
  <span id="Eingabe">Ein eigenes Steuerelement</span>
</body>
</HTML>
```

Vor allem die Kontrolle der clientseitigen Verarbeitung von Klickereignissen ist hier auf stringente Weise umgesetzt worden. Letztlich müssen Sie nicht einmal genau wissen, was JavaScript hier tut. Dank ASP.NET funktioniert es einfach. Freilich liegen die Tücken im Detail, ein genaues Verständnis – vor allem der Schnittstellen – ist unbedingt erforderlich.

7.6.4 Stile und Attribute für Steuerelemente

Weitere Optionen bei der Erzeugung eigener Steuerelemente betreffen die direkte Gestaltung mit HTML-Attributen und Cascading Style Sheets.

Attribute für eigene Steuerelemente erzeugen

Auf die Klasse `HtmlTextWriterAttribute` wurde bereits hingewiesen. Dies ist eine gute Methode, beim Erzeugen der Attribute die korrekte Schreibweise zu sichern. Die Zuweisung findet im Zusammenhang mit `AddAttribute` Verwendung, wie es bereits gezeigt wurde:

```
output.AddAttribute (HtmlTextWriterAttribute.Value, ➡  
                    FieldText)
```

Wenn Sie nun komplexere Gestaltungen benötigen, bieten sich Cascading Style Sheets an. Dafür können Sie eine weitere Klasse verwenden:

```
output.AddStyleAttribute ➡  
(HtmlTextWriterStyle.BackgroundColor, "Red")
```

Die Methode `AddStyleAttribute` weist alle so erfassten Elemente dem Attribut `style` zu und sorgt für die korrekten Trennzeichen gemäß den CSS-Konventionen. Die Elemente der Aufzählung `HtmlTextWriterStyle` umfassen alle in CSS 1.0 definierten Attribute.

Attribute für eingebaute Steuerelemente

Etwas anders sieht es aus, wenn Sie eingebaute Steuerelemente verwenden. Hier erkennt ASP.NET den möglichen Inhalt und verwendet dann passende Strukturen, Aufzählungen oder Klassen, um mit den korrekten Daten umzugehen. Der große Vorteil dabei: Sie können praktisch unmöglich falsche oder sinnlose Zuweisun-

gen erzeugen, was sich gut auf die Qualität des erzeugten HTML auswirkt. Freilich hilft das nicht gegen schlechten Geschmack bei der Gestaltung. Die wichtigsten Strukturen, Aufzählungen oder Klassen zeigt dieser Abschnitt.

Einheiten werden aus der Struktur `Unit` entnommen. `Unit` verfügt über vier statischen Methoden, mit denen Sie entscheiden können, welches Quellformat Ihre Zahlen haben:

Einheiten

► `Parse`

Durchsucht eine Zeichenkette und versucht daraus einen Zahlenwert zu extrahieren.

► `Percentage`

Erwartet eine `Double`-Wert und interpretiert ihn als Prozentwert. Dies ist dann interessant, wenn das Attribut Prozentwerte verarbeiten kann.

► `Pixel`

Erwartet einen `Integer`-Wert und definiert die Anzahl der Pixel.

► `Point`

Erwartet einen `Integer`-Wert und definiert die Größe in Punkt.

Die `Unit`-Struktur überlädt auch Operatoren, sodass ein Vergleich von zwei Instanzen möglich ist. Im ersten Beispiel dieses Abschnittes wurde `Unit` verwendet:

```
ccTextBox.Width = Unit.Pixel (150)
```

`Unit` ist im Namensraum `System.Web.UI.WebControls` definiert.

Farben werden ebenfalls sehr oft benötigt. Sie stammen aus der Struktur `Color`, die in `System.Drawing` definiert ist:

Farben

```
ccTextBox.BackColor = Color.Red
```

Ebenso einfach ist der Umgang mit Fonts. Nutzen Sie für Schriftgrößen die Struktur `FontUnit` aus dem Namensraum `System.Web.UI.WebControls`. `FontUnit.Unit` ergibt die Größe vom Typ `Unit` (siehe oben). `FontUnit.Type` gibt den Schriftgrad vom Typ `FontSize` an. `FontSize` ist eine Aufzählung, die Elemente mit `FontSize.Large` oder `FontSize.Smaller` enthält. Alle Schriftarteigenschaften kapselt die Klasse `FontInfo`. Sie enthält folgende typischen Eigenschaften:

Fonts

- Bold
- Italic
- Name

Dies ist der Name des primären Fonts (Schriftartname).

- Names

Dies ist eine Kollektion aller Fonts, die verwendet werden dürfen, wobei der primäre am Anfang steht.

- Overline
- Size

Die Größe vom Typ `FontSize`

- Strikeout
- Underline

Als letztes Gestaltungselement soll noch die Klasse `Style` vorgestellt werden. Damit lassen sich sehr leicht alle Teile eines `style`-Attributes verwalten. Die folgende Tabelle zeigt die Eigenschaften:

Eigenschaft	Bedeutung und Typ
<code>BackColor</code>	Hintergrundfarbe, <code>Color</code>
<code>BorderColor</code>	Randfarbe, <code>Color</code>
<code>BorderStyle</code>	Stil des Randes, <code>BorderStyle</code> . Der Stil ist selbst wiederum eine Aufzählung, beispielsweise lässt sich <code>BorderStyle.Dotted</code> einsetzen.
<code>BorderWidth</code>	Randbreite, <code>Unit</code>
<code>CssClass</code>	CSS-Klasse, <code>String</code>
<code>Font</code>	Font, <code>FontInfo</code>
<code>ForeColor</code>	Vordergrundfarbe, <code>Color</code>
<code>Height</code>	Höhe, <code>Unit</code>
<code>Width</code>	Breite, <code>Unit</code>

Tabelle 7.2: Eigenschaften der Klasse `Style`

7.6.5 Die Integration ins Visual Studio

Als Tipp soll noch eine Möglichkeit gezeigt werden, wie Sie die Toolbox im Visual Studio .NET um die eigenen Steuerelemente erweitern können. Dazu klicken Sie im Entwurfsmodus in der öff-

neten Toolbox (**Strg** + **Alt** + **X**) mit der rechten Maustaste auf den Block, dem Sie die Steuerelemente hinzufügen möchten, beispielsweise WEB FORMS. Nun wählen Sie im Kontextmenü **TOOLBOX ANPASSEN**. Im folgenden Dialog wechseln Sie zur Registerkarte **.NET Framework-Komponenten**. Klicken Sie auf **DURCHSUCHEN** und wählen Sie die Assembly aus dem Ordner `\bin` oder `\bin\debug` des Projekts `vbnetcontrollib` aus. Aktivieren Sie die selbst gebauten Steuerelemente und klicken Sie dann **OK**.

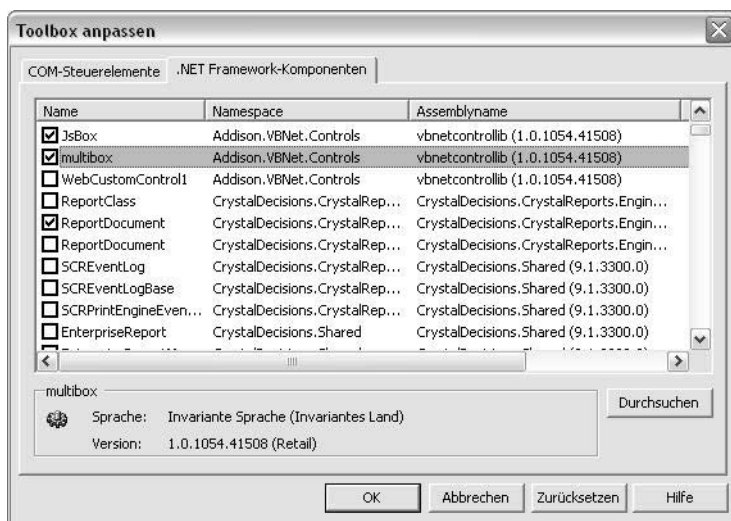


Abbildung 7.40: Hinzufügen eigener Steuerelemente zur Toolbox

Die Elemente erscheinen nun am Ende der Liste und lassen sich per Drag&Drop auf die Entwurfsfläche ziehen. Als Standardsymbol findet das Zahnrad Verwendung. Das ist sicher nicht optimal.

Erscheinungsbild in der Toolbox anpassen

Die Ablage des Elements in der Toolbox ist zwar praktisch, kann aber noch perfektioniert werden. So können Sie das Symbol gegen ein eigenes austauschen. Benötigt wird ein Bitmap-Bild mit 16 Farben und 16 x 16 Pixeln. Zeichnen können Sie es mit dem Grafikeditor in Visual Studio .NET.

Ein Bitmap anfertigen

Gehen Sie dazu folgendermaßen vor:

1. Fügen Sie dem Projekt Ihres Steuerelementes, im vorhergehenden Beispiel hieß es *VBNetControlLib*, ein Bitmap-Element hinzu.



Abbildung 7.41: Ein neues Bitmap-Element wird dem Projekt hinzugefügt.

2. Vergeben Sie ihm einen passenden Namen. Sie benötigen pro Steuerelement – im Projekt also pro von *WebControls* abgeleiteter Klasse – ein Symbol, wenn Sie alle Elemente unterscheiden möchten.
3. Stellen Sie die Größe auf 16 x 16 Pixel ein:

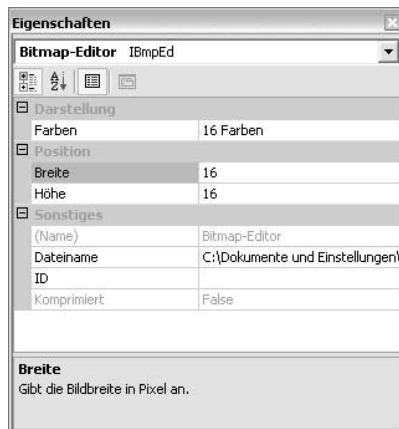


Abbildung 7.42: Im Eigenschaften-Fenster wird die Größe eingestellt.

4. Speichern Sie das Element im Verzeichnis `\bin`, bzw. während der Entwurfsphase, solange noch Debug-Code erzeugt wird, in `\bin\debug`.
5. Zeichnen Sie das Element mit den Zeichenwerkzeugen²:

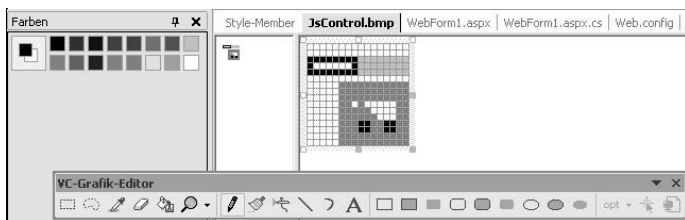


Abbildung 7.43: Der Grafikeditor in Visual Studio .NET

6. Schließen Sie den Grafikeditor.

Nun muss der Assembly eine Information darüber mitgegeben werden, dass ein passendes Symbol existiert. Dazu fügen Sie das folgende Attribut oberhalb der Klassendefinition des betreffenden Steuerelements hinzu (beachten Sie, dass die Attributeklammern »<<« und »>>« nur einmal für alle Attribute auftreten und die Attribute selbst durch Kommata getrennt werden):

```
<ToolboxBitmap("JsControl.bmp")>
```

Die Klasse für dieses Attribut ist im Namensraum `System.Drawing` enthalten, weshalb zuvor noch folgende Zeile an den Anfang der Datei gestellt werden muss:

```
Imports System.Drawing
```

Der Namensraum!

Wenn Sie das Element nun erneut der Toolbox hinzufügen – wie oben bereits beschrieben –, erscheint es nun mit diesem Symbol.

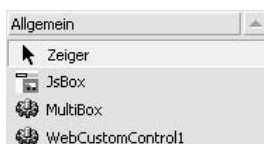


Abbildung 7.44: Eigene Steuerelemente, mit und ohne eigenem Symbol

- 2 Das ist übrigens die eigentliche Herausforderung. Um die Qualität der übrigen Symbole in Windows zu erreichen ist einige Erfahrung nötig. Verzweifeln Sie nicht, dem Autor des Buches sind auch keine brauchbaren Entwürfe gelungen ;-).



Wenn Sie beim Import der Assembly eine Fehlermeldung erhalten, die nur den Namen des Symbols nennt, stimmt der Pfad nicht. Verschieben Sie dann die Bitmap-Datei oder passen Sie den Pfad im Attribut `ToolboxBitmap` an.

Drag&Drop-Verhalten modifizieren

Im nächsten Schritt geht es darum, das Verhalten der Steuerelemente bei der Verwendung der Toolbox zu verbessern. So wurde im Beispielcode vorgeschlagen, dass als Präfix »cc« verwendet wird. Außerdem soll ein Steuerelemente-Entwickler in der Lage sein, die Gestaltung mit dem Designer vorzunehmen. Auch hier helfen Design-Time-Attribute.

Einstellen einer Präfixvorgabe

Um das Präfix fest einzustellen, wird dieses mit dem verwendeten Namensraum verknüpft. Die Information muss Bestandteil des Manifestes werden – das ist eine Informationsmenge, die Bestandteil einer Assembly ist. Wie diese entsteht, muss hier nicht weiter interessieren. Das erledigt der Compiler zuverlässig. Sie können aber ein wenig eingreifen. Im Projekt in Visual Studio finden Sie eine Datei *AssemblyInfo.vb*. Darin sind die Informationen enthalten, die später das Manifest bilden.

Öffnen Sie jetzt diese Datei und fügen Sie am Anfang folgende Deklaration hinzu:

```
Imports System.Web.UI
```

Gehen Sie nun etwas tiefer, etwa bis Zeile 11. Dort finden Sie die `<Assembly>`-Attribute. Fügen Sie dort ein weiteres hinzu:

```
<Assembly: TagPrefix("Addison.VBNet.Controls", "cc")>
```

Damit definieren Sie, dass bei Verwendung des Namensraumes `Addison.VBNet.Controls` für neue Steuerelemente aus dieser Bibliothek das Präfix »cc« verwendet werden soll.

Dateiinformationen modifizieren

Sie können hier auch andere Angaben ergänzen, beispielsweise den Namen Ihres Unternehmens. Betrachten Sie folgende Angaben, die im Beispielprojekt eingetragen wurden:

```
<Assembly: AssemblyTitle("ControlLibrary")>
<Assembly: AssemblyDescription("Verschiedene experimentelle Steuerelemente")>
<Assembly: AssemblyConfiguration("")>
<Assembly: AssemblyCompany("Buennung & Krause GbR")>
```

```
<Assembly: AssemblyProduct("ASP.NET mit VB.Net Version 1.1")>
<Assembly: AssemblyCopyright("(c) 2003 by Joerg Krause")>
<Assembly: AssemblyTrademark("")>
```

Wenn Sie im Windows-Explorer die Dateieigenschaften betrachten, finden Sie die Angaben auf der Registerkarte VERSION wieder.

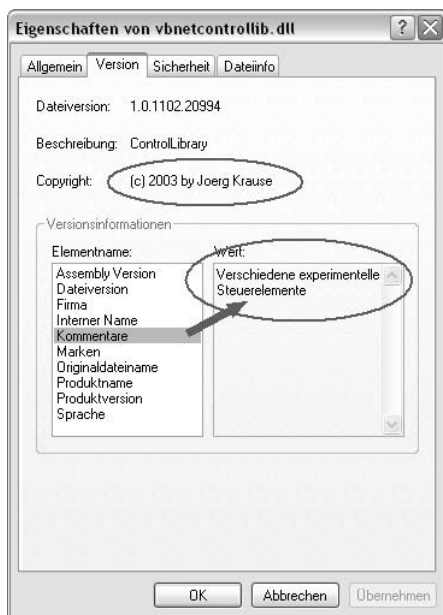


Abbildung 7.45: So erscheinen die Assembly-Informationen im Windows-Explorer.

Speichern Sie die Datei und schließen Sie sie wieder.

Jetzt soll noch festgelegt werden, wie das Element bei Drag&Drop benannt wird. Diese Festlegung wird auch mit einem Attribut getroffen, diesmal jedoch wieder in der Klassenbibliothek selbst, oberhalb der Definition der Klasse des betreffenden Steuerelements. Tragen Sie dort Folgendes ein:

```
<ToolboxData ("<{0}:JsBox runat=server></{0}:JsBox>")>
```

Anstatt der Formatanweisung {0} wird das Präfix eingetragen. Wenn Sie den vorhergehenden Schritt ausgeführt haben, wäre dies »cc«.

Alle übrigen Einstellungen können nun über das Eigenschaften-Fenster erreicht werden. Um es zu öffnen, markieren Sie das Element und drücken dann F4.

Benennung des Elements bei Drag&Drop

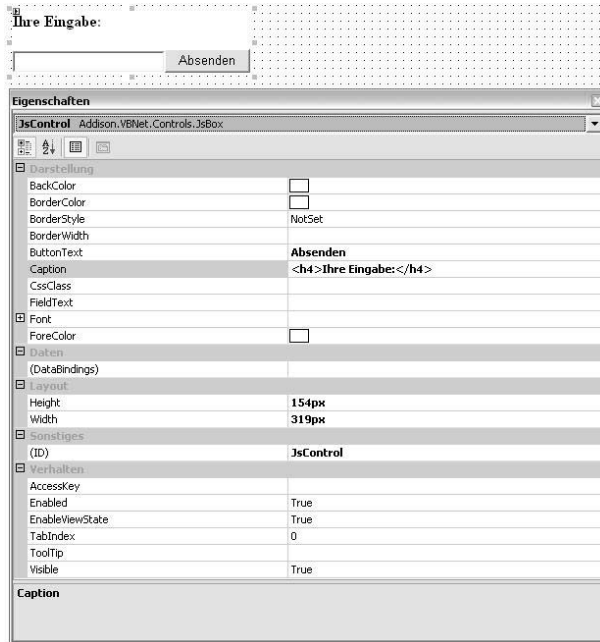


Abbildung 7.46: Eigenschaften des eigenen Steuerelements

Tatsächlich erscheinen alle eigenen Eigenschaften des Elementes bereits in der Liste SONSTIGES. Damit sich der Anwender besser orientieren kann, können Sie eine Vorauswahl eines Attributes in der Liste treffen. Mit dem folgenden Attribut wird festgelegt, dass die Eigenschaft `ButtonText` hervorgehoben erscheint:

```
<DefaultProperty("ButtonText")>
```

Wenn der Anwender ein Doppelklick ausführt, wird er normalerweise in den Code-Editor springen und dort in eine neu erzeugte Methode, die ein Standardereignis verarbeitet. Das folgende Attribut legt fest, welches das Standardereignis ist:

```
<DefaultEvent("Click")>
```

Damit ist schon eine sehr praktische Nutzung möglich. Gemessen am geringen Aufwand lohnt der Einsatz der Attribute für Steuerelemente auch für eigene Projekte.



Dieser Abschnitt war lediglich als Anregung für eigene Versuche gedacht, denn es fehlt der Platz um die endlosen Optionen auch nur ansatzweise darzustellen. Informieren Sie sich in der MSDN-Referenz unter den hier gezeigten Stichworten über weitere Möglichkeiten.

8 Protokollnahe und ablauforientierte Programmierung

Alle Datenströme zwischen Server und Browser werden über ein Protokoll übertragen. Normalerweise kommt dazu HTTP zum Einsatz. Bedingt durch die Art des Protokolls und den Bedingungen und der Arbeitsweise von Browsern und Servern ergeben sich ganz bestimmte Techniken, durch die sich die Webserverprogrammierung grundlegend von der Windows-Programmierung unterscheidet. Dieses Kapitel zeigt die Aspekte dieser Programmierung und stellt die mit ASP.NET benutzten Techniken zusammen.

8.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

8.1.1 Über dieses Kapitel

ASP.NET bietet dem Webprogrammierer ein durchdachtes Programmiermodell und eine weitgehende Abstraktion von den verwendeten Protokollen – speziell HTTP. Durch die Darstellung einer Webseite in Form einer Hierarchie von Steuerelementen, denen ein klares Objektmodell zugrunde liegt, wird der Zugriff auf HTML vom Programmcode stark vereinfacht. Dies funktioniert in den meisten Fällen, ohne dass der direkte Weg zwischen Browser und Server – das Anforderungs/Antwort-Verhalten – programmiert werden muss.

Dennoch gibt es in der Praxis Situationen, in denen die vorgegebenen Möglichkeiten unzureichend sind. In solchen Fällen ist es wichtig, den »wahren« Ablauf zwischen Browser und Server zu kennen. ASP.NET bietet auch hier eine umfassende Unterstützung, indem der gesamte Prozess in ein Objektmodell gepackt wird. Die sich daraus ergebenden Standardobjekte bilden die Basis für die Beherrschung des Datenstromes vom Browser (Request) oder vom Server (Response). Lesen Sie mehr darüber im Abschnitt 8.2, »Die Welt der Standardobjekte« ab Seite 747.

Um generell benutzerabhängig arbeiten zu können, ist ein umfangreiches Sitzungsmanagement integriert. Neben der klassischen Nutzerverfolgung mit Sitzungs-Cookies kennt ASP.NET nun auch Fallback-Mechanismen. Schlagen Sie im Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778, mehr darüber nach.

Ein wichtiger Aspekt des Zugriffs auf das Protokoll HTTP ist außerdem die Programmierung von Cookies. Sie können damit Informationen permanent mit einem Benutzer verbinden. Abschnitt 8.5, »Cookies« ab Seite 791, enthält die nötigen Informationen.

Werden globale, applikationsweite Daten benötigt, ist ein Zugriff auf die Applikationsinstanz sinnvoll. Das Applikationsmanagement widmet sich dieser Aufgabe, beschrieben im Abschnitt 8.6, »Applikationsmanagement« ab Seite 802.

Für die gesamte Applikation ist auch der effiziente Austausch von Daten und der verantwortungsvolle Umgang mit Ressourcen wichtig. Mit ASP.NET können Sie die Zwischenspeicherung von Daten (Caching) detailliert steuern und so jede Anwendung optimieren. Abschnitt 8.7, »Optimierung des Datenverkehrs« ab Seite 809 führt in diese Technik ein.



In diesem Kapitel wird nur selten Code Behind eingesetzt, weil es die zum Teil sehr kurzen Code-Stückchen unnütz aufblähen würde. In der Praxis sollten Sie dennoch, wenn es geht, mit hinterlegtem Code arbeiten. Dieses Buch stellt insofern einen Sonderfall dar, als dass es statt eines großen viele kleine Programme präsentiert.

8.1.2 Wegweiser in die Referenz

Wenn Sie sich in der Online-Referenz über Eigenschaften und Methoden informieren müssen, ist es sinnvoll, die Abhängigkeiten der Klassen im Framework zu kennen. Die folgende Abbildung zeigt dies für die wichtigsten Klassen.

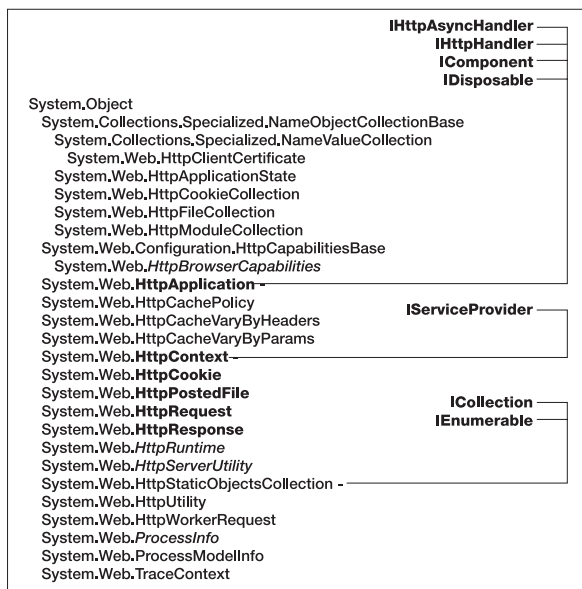


Abbildung 8.1: Der Namensraum `System.Web` (vereinfacht)

8.2 Die Welt der Standardobjekte

Gerade am Anfang bereitet es Schwierigkeiten zu verstehen, wo die Unterschiede zwischen den verschiedenen Arten serverseitigen Zugriffs auf HTML liegen. Dabei machen es die Bezeichnungen der Klassen, die den Elementen zugrunde liegen, nicht unbedingt einfacher.

8.2.1 Die Standardobjekte

Im Namensraum `System.Web` sind eine Vielzahl von Klassen zu finden, die der Programmierung von ASP.NET-Applikationen dienen. Wegen der vielfältigen Beziehungen untereinander und einiger

**Der Namensraum
`System.Web`**

Techniken, wie beispielsweise der automatischen Instanziierung oder statischer Methoden, ist das System dahinter nicht immer sofort erkennbar. Sie sollten sich die an der Abarbeitung der Applikation beteiligten Komponenten und Vorgänge vor Augen halten. Die folgende Abbildung nennt die wichtigsten Objekte und wie diese an der Erstellung der Seiten beteiligt sind.

Application In der Hierarchie ganz oben steht das Objekt `Application`. Es wird automatisch aus der Klasse `HttpApplication` instanziiert. Dieses Objekt enthält verschiedene Eigenschaften, die den Zugriff auf die anderen Objekte erlauben. Allerdings werden auch diese implizit instanziiert. Die wichtigsten, die auch in diesem Buch Verwendung finden, sind Folgende:

► `Application`

Die Eigenschaft `Application` gibt selbst ein Objekt `Application` zurück, das allerdings von der Klasse `HttpApplicationState` abgeleitet wird. Die Namensgleichheit mag irritierend sein, ASP.NET erkennt das korrekte Objekt aber am Kontext der Verwendung.

► `Session`

Mit diesem Objekt wird auf das Sitzungsmanagement zugegriffen.

► `Request`

Hiermit erhalten Sie Zugriff auf alle Informationen, die im Zusammenhang mit der Anforderung vom Browser stehen, beispielsweise den Querystring (URL-Parameter) oder Formularinhalte.

► `Response`

Dieses Objekt dient der Kontrolle des Ausgabedatenstromes. Vermutlich haben Sie schon `Response.Write` verwendet, eine Methode zum Ausgeben von Daten an den Browser.

► `Cache`

Mit diesem Objekt wird die Zwischenspeicherung von Daten kontrolliert.

► `Server`

Dieses Objekt stellt den Zugriff auf Serverfunktionen sicher.

► `Page`

Das Objekt, das die aktuell angeforderte (aktiven) Seite repräsentiert.

► Context

Dieses Objekt fasst die Daten des aktuellen Vorgangs, also alle in einer Sitzung verwendeten Seiten, zusammen. Damit können seitenübergreifende Lösungen programmiert werden, ohne dass große Datenmengen in Sitzungsvariablen gespeichert werden müssen und ohne dass auf das fortschrittliche Formular-Management verzichtet werden muss. Eine Behandlung dieses Themas wird in Abschnitt 8.3.8, »Kontext-Handler und Seitenreferenzierung« ab Seite 773 vorgenommen.

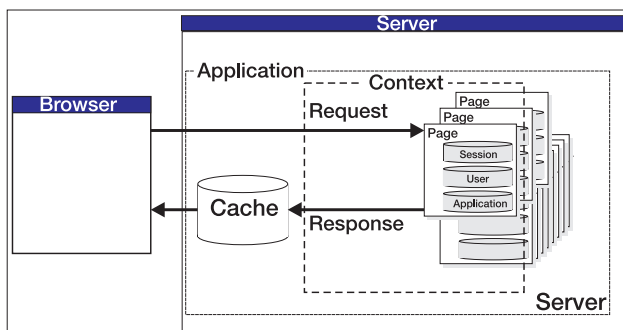


Abbildung 8.2: Zuständigkeit der Objekte in ASP.NET

Das Objekt `Application` wird einmal beim Start der Applikation instanziiert. Es steht dann die gesamte Laufzeit über zur Verfügung. Bei der Gestaltung von Web-Applikationen sollten Sie daran denken, dass diese zwar prinzipiell aus einzelnen Seiten bestehen. Jeder Benutzer löst im ersten Schritt eine Anforderung aus (Request). Es entsteht das `Request`-Objekt. Normalerweise wird eine `aspx`-Seite aufgerufen. Damit entsteht ein `Page`-Objekt. Wenn auf einer Seite weitere Elemente wie Bilder untergebracht sind, löst der Browser nach dem Empfang der Antwort (Response) weitere Anforderungen aus. Diese führen aber nicht zwingend zu weiteren oder erneuten Seitenaufrufen. Andererseits kann ein HTML-Frameset zum Abruf mehrerer Seiten führen, ebenso wie der Nutzer beim Surfen weitere Anforderungen auslöst. Der Benutzer selbst wird – unabhängig von der Anzahl der Anforderungen – innerhalb einer Sitzung als Einheit betrachtet. Es existiert für ihn also nur ein `Session`-Objekt. Alle Seiten mit externem Code zusammen bilden wiederum eine Applikation, die ihrerseits spezifische Eigenschaften hat, die die Art der Verarbeitung kontrollieren. Dazu gehört auch die Arbeitsweise des Zwischenspeichers des Aus-

Instanziierung

gabedatenstromes, repräsentiert im `Cache`-Objekt. Noch globaler ist das `Server`-Objekt zu betrachten, das sehr allgemeine Zustände erreichbar werden lässt. Dazu gehört beispielsweise die Fehlerverwaltung und die Bereitstellung von Hilfsfunktionen zur Kodierung oder Dekodierung.

Weitere Klassen Aus den `Response`- und `Request`-Klassen können weitere abgeleitet werden, die wiederum durch eigene Typen repräsentiert werden. Dazu gehört die Behandlung von Cookies, die eine Klasse zur Erstellung verwenden, die die Darstellung jedes Cookies als Objekt erlaubt und die entsprechende Kollektion im Objekt `Response`, die den Zugriff auf zu sendende Cookies erlaubt bzw. `Request`, wo die empfangenen Cookies als Kollektion bereitgestellt werden.

8.3 Daten senden und empfangen

Von zentraler Bedeutung bei der ASP.NET-Programmierung ist die Kontrolle des Sendens und des Empfangens von Informationen.

8.3.1 Den Datenfluss steuern

Der Datenaustausch zwischen Browser und Webserver besteht aus den Prozessen Anforderung (`Request`) und Antwort (`Response`). Wann immer Daten übertragen werden, erlauben die gleichnamigen Objekte den Zugriff darauf über Methoden und Eigenschaften. Beide Objekte sind bereits instanziiert und bilden immer die aktuelle Anforderung bzw. Antwort ab, sodass nur eine einzige Instanz nötig und möglich ist. Ergänzend ist auch das Objekt `Server` von Interesse, das unter anderem mehrere Hilfsmethoden bietet.

Antwortkontrolle und Datenausgabe in der Seite

Response Die Klasse `HttpApplication` stellt eine Eigenschaft `Response` bereit, die ein Objekt der Klasse `HttpResponse` instanziiert. Dieses steht unter dem Namen `Response` zur Verfügung. Der Vorgang weicht vom üblichen Vorgehen der Ableitung von Objekten ab, damit zum einen die Programmierung vereinfacht, zum anderen aber auch eine wenigstens teilweise Kompatibilität zum alten ASP 3.0 erreicht wird.

Die erste Methode des Objekts, `Write`, haben Sie bereits verwendet. Damit wird ein Text an den Browser ausgegeben. Statt Text kann auch der Inhalt einer Datei ausgegeben werden. Dazu wird `WriteFile` eingesetzt. Diese Methode besitzt zwei optionale Parameter, die Startpunkt und Länge des Textes angeben, gezählt in Byte aus der Datei.

Response.Write
Response.
WriteFile

Beachten Sie, dass die Ausgabe nur sinnvoll ist, wenn der Ausgabedatenstrom tatsächlich kontrolliert werden kann. Dazu müsste der Code in HTML eingebettet sein. Im Zusammenhang mit Steuerelementen ist die Nutzung also ziemlich sinnlos. Aber natürlich gibt es Anwendungsfälle, beispielsweise bei der Auslieferung von Dateien an den Browser.



```
<script language="vb" runat="server">
Dim footer As String = "footer.html"
</script>
<html>
  <head>
    <title>Response.WriteFile</title>
  </head>
  <body>
    <hr/>
    <%
      Response.WriteFile(footer)
    %>
  </body>
</html>
```

Listing 8.1: Ausgabe einer Datei am Ende einer Seite (ResponseWritefile.aspx)

Die Ausgabe an den Browser kann prinzipiell mit zwei Methoden erfolgen. Zum einen wird jeder Text, der erzeugt wird, sofort gesendet. Der Browser stellt diesen meist fortlaufend dar. Zum anderen ist die Verwendung eines Puffers möglich, der die Ausgaben sammelt und dann mit einem Mal zum Browser schickt.

Response.
BufferOutput

Der Effekt kann verdeckt ablaufen, wenn HTML-Tabellen verwendet werden oder der Aufbau der Seite nicht klar ist, bis ein bestimmtes Element auftritt. Hier spielt also auch die Gestaltung eine Rolle. Für einfachen Text gilt aber, dass gesendete Daten auch sofort erscheinen.



Das Verhalten kann modifiziert werden. Sie können auch einen Pufferspeicher verwenden, der alle Ausgaben sammelt. Erst wenn die Seite vollständig abgearbeitet wurde, werden alle Daten gesendet.

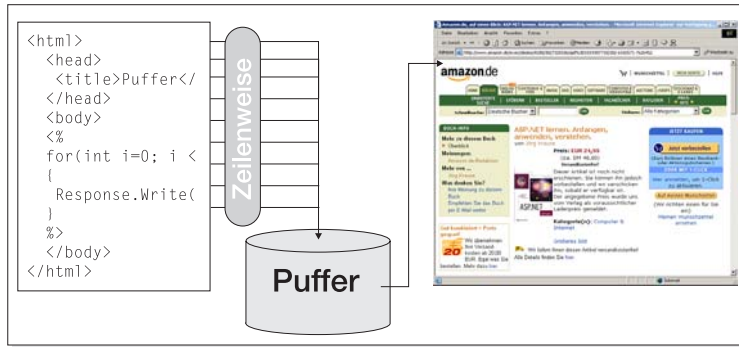


Abbildung 8.3: Arbeitsweise des Puffers: Zeilenweise Ausgaben werden gesammelt und am Ende als Block gesendet

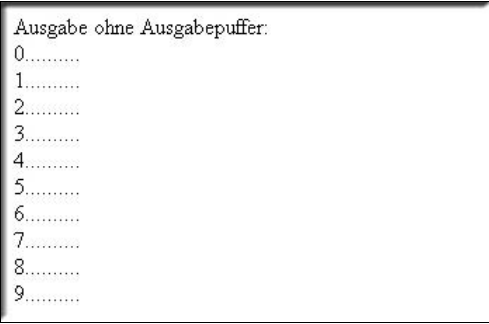
Dies ist die Standardeinstellung – ASP.NET verwendet die Ausgabepufferung. Das folgende Beispiel zeigt, wie die Pufferung abgeschaltet werden kann:

```
<%@ Import Namespace="System.Threading" %>
<script language="vb" runat="server">
Sub Page_Load(sender As Object, e As EventArgs)
    Response.BufferOutput = False
    Thread.Sleep(500)
End Sub
</script>
<html>
    <head>
        <title>Response.Write</title>
    </head>
    <body>
        Ausgabe ohne Ausgabepuffer:
        <br/>
        <%
            Dim i, j As Integer
            For i = 0 To 10 Step 1
                Response.Write (i.ToString())
                For j = 0 To 10 Step 1
                    Response.Write (".")
                    Thread.Sleep(50)
                    Response.Flush
                Next
                Response.Write("<br />")
                Thread.Sleep(100)
            Next
        %>
    </body>
</html>
```

Listing 8.2: Ausgabe ohne Puffer (ResponseWrite.aspx)

Die Einstellung erfolgt über die Eigenschaft `BufferOutput`. Übergeben Sie `True`, um die Pufferung einzuschalten oder `False`, um sie nicht zu verwenden. Die beiden Schleifen geben viele Punkte aus, wobei das Senden der Daten mit `Response.Flush` explizit forciert wird. Damit der Effekt gut beobachtet werden kann, wurden Pausen von jeweils 50 Millisekunden hinter jedem Punkt ausgegeben. Möglicherweise erscheinen die ersten Zeilen nicht punktweise, sondern als Block. Das liegt nicht an Fehlern im Puffer, sondern daran, dass der Browser am Anfang mit der Darstellung nicht nachkommt. Auf schnellen Rechnern fällt der Effekt nicht mehr auf.

Ausgabepuffer



```
Ausgabe ohne Ausgabepuffer:  
0.....  
1.....  
2.....  
3.....  
4.....  
5.....  
6.....  
7.....  
8.....  
9.....
```

Abbildung 8.4: Ohne Ausgabepuffer erscheinen die Punkte nacheinander

Im Beispiel wurde eine kurze Pause (0,5 Sekunden) erzeugt, indem der aktuelle Thread (Arbeitspfad des Prozesses) angehalten wird. Dazu musste zum einen der Namensraum `System.Threading` eingebunden und zum anderen die statische Methode `Sleep` aufgerufen werden.

Pause mit der Threading-Klasse

Der Verzicht auf den Ausgabepuffer ist außerordentlich ineffektiv, auch wenn zur Demonstration künstliche Pausen eingebaut werden mussten. Es gibt jedoch sinnvolle Anwendungen. Wenn beispielsweise eine Datenbankabfrage längere Zeit dauert, kann eine gelegentliche Ausgabe von Zeichen an den Browser für den Benutzer hilfreich sein und vom Anklicken der -Schaltfläche abhalten.

Effizienz des Puffers

Derart lange Aktionen können öfter auftreten, wenn komplexe Abfragen ausgeführt werden. Die ASP.NET-Komponente übernimmt vom IIS den Timeout-Wert von 90 Sekunden für die Abarbeitung einer Seite. Wird innerhalb dieser Zeit das Programm nicht fertig gestellt, wird der Prozess recycelt und der Browser

Zeitüberschreitung

zeigt eine Fehlermeldung an. Der Wert kann natürlich programmtechnisch verändert werden. Da diese Einstellung für den gesamten Serverprozess gilt, wird das Objekt `Server` eingesetzt. `Server` steht ebenfalls automatisch und als einzige Instanz der Klasse `HttpServerUtility` bereit. Das folgende Programm setzt eine kurze Timeout-Zeit und provoziert die Zeitüberschreitung durch eine Endlosschleife.

```
<script language="vb" runat="server">
Sub Page_Init(sender As Object, e As EventArgs)
    Server.ScriptTimeout = 4
End Sub
</script>
<html>
    <head>
        <title>Response.Write</title>
    </head>
    <body>
        Ausgabe ohne Ausgabepuffer:
        <br/>
        <%
            Do While True
                Loop
            %>
    </body>
</html>
```

Listing 8.3: Abbruch einer Endlosschleife durch Timeout (*ServerScriptTimeout.aspx*)

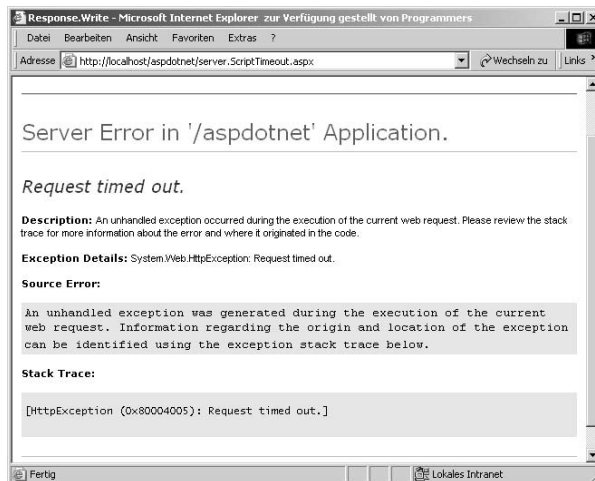


Abbildung 8.5: Fehler durch eine Zeitüberschreitung beim Programmablauf

Wenn Sie das Programm ausprobieren, wird nach Ablauf der Zeit ein Fehlerereignis erscheinen.

Es ist sicher sinnvoll, auf derartige Fehler reagieren zu können. Verwenden Sie *Try-Catch*-Anweisungen dafür. Informationen darüber finden Sie in Abschnitt 3.4.10, »Ausnahmebehandlung« ab Seite 208.

8.3.2 Daten von Seiten zu Seite übertragen

Besteht Ihre Applikation aus mehreren Seiten, müssen oft Daten zwischen diesen ausgetauscht werden. Dafür gibt es mehrere Wege:

- ▶ Cookies
- ▶ Sitzungsvariablen (im Rahmen des Session-Managements)
- ▶ HTTP-Methode POST
- ▶ HTTP-Methode GET

Cookies werden in Abschnitt 8.5, »Cookies« ab Seite 791 behandelt. Die zu speichernden Daten werden dabei an den Browser übertragen und von diesem wieder zurückgesendet. *Session*-Objekte speichern Variablen intern und nutzen den Sitzungsstatus, um dem Benutzer seine Werte zuzuordnen. Ausführlich wird dies in Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778 diskutiert. Die Standardmethoden in der Webserverprogrammierung sind jedoch POST und GET. POST ist das Kommando, das der Browser nutzt, um Formulardaten zu senden. Wenn nur mit Hyperlinks gearbeitet wird, verwendet der Browser zur Anforderung von Ressourcen das Kommando GET.

Die Standardmethoden

Daten per URL übertragen

Die ganze Technik beruht darauf, die zu übertragenden Daten in einer ganz bestimmten Art und Weise an den URL anzuhängen:

HTTP-GET

```
http://www.seite.de/target.aspx?vari1=Wert1&vari2=Wert2
```

Dieser Link überträgt zwei Variablen: *vari1* und *vari2*, die die Zeichenketten »Wert1« bzw. »Wert2« enthalten. Auf der Seite *target.aspx* werden diese im *Request*-Objekt bereit gestellt.

Drei spezielle Zeichen finden dabei Verwendung: Das Trennzeichen zwischen dem URL und den angehängten Parametern ist

das Fragezeichen. Jedes einzelne Wertepaar wird mit einem &-Zeichen getrennt. Zwischen Variable und Wert steht ein Gleichheitszeichen. Sie können theoretisch beliebig viele Werte übertragen. Beachten Sie aber, dass die Browser ganz unterschiedliche Längen für die komplette URL akzeptieren. Der Internet Explorer überträgt ca. 2.000 Zeichen. Die gesamte Parameterschlange nach dem Namen der Seite wird als »QueryString« bezeichnet.

Request. QueryString

Die Auswertung wird entsprechend mit der Methode `Request.QueryString` vorgenommen. Diese Methode extrahiert die einzelnen Werte aus dem URL. Schauen Sie sich das folgende Beispiel an, das zur Bestellung von Büchern dient:

```
<body>
  <h2>Willkommen in unserem Buchladen</h2>
  <b>Ihre Bestellung bitte:</b><br/>
  <a id="link1" runat="server"/><br/>
  <a id="link2" runat="server"/><br/>
  <a id="link3" runat="server"/><br/>
  <a id="link4" runat="server"/><br/>
  <br/>
  <asp:label id="result" runat="server"/>
</body>
```

Listing 8.4: Übertragung von Daten per GET (RequestQueryString.aspx)

```
Public Class RequestQueryString
  Inherits System.Web.UI.Page
  Protected WithEvents result As Label
  Protected WithEvents link1 As HtmlAnchor
  Protected WithEvents link2 As HtmlAnchor
  Protected WithEvents link3 As HtmlAnchor
  Protected WithEvents link4 As HtmlAnchor

  Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Dim self As String =
      = Request.ServerVariables("SCRIPT_NAME")
    Dim booknumber As Integer
    link1.InnerHtml = "Der Unbesiegbare"
    link1.HRef = self + "?book=1"
    link2.InnerHtml = "Der Schnupfen"
    link2.HRef = self + "?book=2"
    link3.InnerHtml = "Sternstagebücher"
    link3.HRef = self + "?book=3"
    link4.InnerHtml = "Eden"
    link4.HRef = self + "?book=4"
```



```

    If Not Request.QueryString("book") Is Nothing Then
        booknumber ←
        = Integer.Parse(Request.QueryString("book"))
    End If
    result.Text = "Ihre Bestellung: "
    Select Case booknumber
        Case 1
            result.Text += link1.InnerText
        Case 2
            result.Text += link2.InnerText
        Case 3
            result.Text += link3.InnerText
        Case 4
            result.Text += link4.InnerText
        Case Else
            result.Text = "Sie haben noch keine ←
                          Wahl getroffen"
    End Select
End Sub
End Class

```

Listing 8.5: Die Code-Datei zu RequestQuerystring.aspx (RequestQuerystring.aspx.vb)

Das Programm nutzt serverseitige Steuerelemente, die im Detail in Kapitel 6, »Programmierung von Web Forms« ab Seite 501 diskutiert werden. Basis der Seite bilden die Hyperlinks, die mit <a>-Tags erzeugt werden:

Wie es funktioniert

```
<a id="link1" runat="server"/>
```

Dies unterscheidet sich insofern von HTML, als dass hier der notwendige Parameter href und der Inhalt des Tags dynamisch in der Methode Page_Load erzeugt werden. An den Browser wird folgende Zeile gesendet:

```
<a href="/aspdotnet/RequestQuerystring.aspx?book=1" id="link1">
    Der Unbesiegbare
</a>
```

Erzeugt wird diese am Anfang der Methode Page_Load. Dort wird zunächst der Name der Seite ermittelt:

```
Dim self As String = Request.ServerVariables("SCRIPT_NAME")
```

Dann wird der Text zugewiesen, der als Link erscheinen soll:

```
link1.InnerHtml = "Der Unbesiegbare"
```

Der entscheidende Punkt ist das Anhängen der GET-Parameter, die später ausgewertet werden sollen:

```
link1.HRef = self + "?book=1"
```

Nachdem sich die Seite selbst aufruft, müssen die übertragenen Parameter noch ausgewertet werden. Dazu wird die Buchnummer aus der Kollektion `QueryString` ermittelt. Die Umwandlung in eine Zahl erleichtert die Weiterverarbeitung. Welche Konvertierung Sie vornehmen, hängt von Ihren Daten ab. Standardmäßig werden die Informationen als Zeichenkette bereitgestellt und folgendermaßen in Ganzzahlen gewandelt:

```
booknumber = Integer.Parse(Request.QueryString("book"))
```

Der *Select*-Zweig sorgt dann dafür, dass das Ergebnis an das Element `<asp:label/>` übertragen wird. Die folgende Abbildung zeigt das Ergebnis nach einem Klick auf einen Link. Beachten Sie die Anzeige der Parameter in der Statuszeile des Browsers:



Abbildung 8.6: Arbeitsweise des Beispiels

Solange Ziffern übertragen werden, bereitet der Umgang mit der Kodierung des Querystring, wenig Probleme. Wenn Sie beliebige Werte übertragen möchten, muss jeder einzelne in die URL-Form gebracht werden. ASP.NET übernimmt das für Sie automatisch. Werden die Buchnamen direkt übermittelt, ist dies gut zu beobachten, wie der folgende Ausschnitt des URL zeigt:

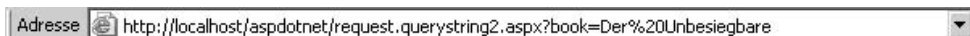


Abbildung 8.7: Kodierung von Sonderzeichen im URL

Neben den Leerzeichen werden vor allem die Zeichen & und = kodiert, da diese als Parametertrennzeichen benötigt werden. Die Darstellung erfolgt in der Form %HH, wobei HH der hexadezimale Code des Zeichens ist.

8.3.3 HTTP-Kopfzeilen erzeugen und analysieren

Normalerweise sorgt ASP.NET für die Erzeugung der richtigen HTTP-Kopfzeilen(Header). Es gibt jedoch Fälle, wo Sie eigene Header erzeugen müssen. Voraussetzung ist immer, dass die Header noch nicht gesendet wurden. Deshalb ist die Nutzung in der Methode `Page_Init` angebracht. Header werden mit der Methode `Response.AppendHeader` der Liste der Header hinzugefügt. Die Methode `Response.ClearHeaders` löscht alle bereits erfassten Header.

Response.AppendHeader

Einsatzmöglichkeiten

Zwei wichtige Header werden häufiger eingesetzt: *Location* und *Content-type*. Beide müssen nicht von Hand erzeugt werden; dafür stehen spezielle Eigenschaften des Objekts `Response` zur Verfügung. Mit *Location* steuern Sie Weiterleitungen, dies wird im Abschnitt 8.3.6, »Anforderungen weiterleiten« ab Seite 767 erläutert.

**Location
Content-Type**

Weitere Header lassen sich einsetzen, um den Dateityp beim programmgesteuerten Herunterladen von Dateien zu bestimmen. Folgende Sequenz hat sich hierfür bewährt:

```
Content-Type: application/pdf
Content-Disposition: attachment; filename=Datei.pdf
Content-Description: ASP.NET-Generated Data
Pragma: no-cache
Expires: 0
```

8.3.4 Servervariablen

Der Browser übermittelt mit jeder Anforderung einige Informationen an den Server, die in Programmen sinnvoll eingesetzt werden können. Zusammen mit Statusinformationen des Servers ergibt sich eine Sammlung von Daten über die aktuelle Verbindung. Das Objekt `Request` stellt diese Information über die Eigenschaft `ServerVariables` zur Verfügung.

**Request.
ServerVariables**

Wichtige Server-variablen

Nicht alle Werte werden für die praktische Programmierarbeit wirklich benötigt. Die wichtigsten sind die Folgenden:

► HTTP_REFERER

Wenn Ihre Seite durch Anklicken eines Hyperlinks auf einer anderen Seite erreicht wurde, enthält diese Variable den URL der Seite, von welcher der Nutzer kam.

► HTTP_USER_AGENT

Der Typ des Browsers wird angezeigt. Sie können auswerten, welche Browser Ihre Nutzer bevorzugen, und die Gestaltung der Seiten daran ausrichten.

► REMOTE_ADDR

Dieses Feld enthält die IP-Adresse, mit der der Browser die Verbindung hergestellt hat.

► QUERY_STRING

Diese Variable enthält die Zeichenkette nach dem Fragezeichen in einem URL; dem Trennzeichen für die Übertragung von Parametern zum Server.

► SCRIPT_NAME

Der virtuelle (relative) Pfad der aktuellen ASP.NET-Seite. Damit können Sie Seiten automatisch mit sich selbst referenzieren, ohne den Speicherort zu kennen.

► SERVER_NAME

Der Name des Webserver oder die IP-Adresse.

► PATH_TRANSLATED

Der physische Pfad der ASP.NET-Seite auf der Festplatte des Webserver.

Wie der Zugriff auf die gesamte Kollektion erfolgt, zeigt das folgende Beispiel:

```
<%  
Dim sv As NameValueCollection  
sv = Request.ServerVariables  
Dim element As String  
For Each element In sv  
    Response.Write ("<b>" + element + "</b> : ")  
    Response.Write (sv(element) + "<br />")  
Next%>
```

Listing 8.6: Auslesen aller Servervariablen (Ausschnitt aus RequestServerVariables.aspx)

Die Klasse `NameValueCollection` bietet die Verwaltung von Schlüssel/Werte-Paaren, wie sie speziell bei der Abbildung von Attribut-Kollektionen der HTML-Tags auftreten. Sie verhält sich damit ähnlich wie `Hashtable`. Mehr Informationen zum Umgang mit Aufzählungen vermittelt der Abschnitt 3.4.9, »Strukturen und Aufzählungen« ab Seite 204.

**Hinweis zu
NameValue-
Collection**

Die Antwort bietet wertvolle Informationen über die aktuelle Verbindung:

```
LOCAL_ADDR : 127.0.0.1
PATH_INFO : /aspdotnet/request.servervariables.aspx
PATH_TRANSLATED : C:\inetpub\wwwroot\dotnet\request.servervariables.aspx
QUERY_STRING :
REMOTE_ADDR : 127.0.0.1
REMOTE_HOST : 127.0.0.1
REQUEST_METHOD : GET
SCRIPT_NAME : /aspdotnet/request.servervariables.aspx
SERVER_NAME : localhost
SERVER_PORT : 80
SERVER_PORT_SECURE : 0
SERVER_PROTOCOL : HTTP/1.1
SERVER_SOFTWARE : Microsoft-IIS/5.0
URL : /aspdotnet/request.servervariables.aspx
HTTP_CONNECTION : Keep-Alive
HTTP_ACCEPT : */*
HTTP_ACCEPT_ENCODING : gzip, deflate
HTTP_ACCEPT_LANGUAGE : de,en-us;q=0.8,en-au;q=0.5,en-gb;q=0.3
HTTP_AUTHORIZATION : Basic QWRtaW5pc3RyYXRvcjpbGVhZW5z
HTTP_COOKIE : ASP.NET_SessionId=33dwkwumswsoid2wxabq1j45
HTTP_HOST : localhost
HTTP_REFERER : http://localhost/aspdotnet/
HTTP_USER_AGENT : Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)
```

Abbildung 8.8: Ausschnitt aus der Liste der Servervariablen

In der Praxis wird die vollständige Liste nur selten benötigt. Das folgende Programm zeigt, wie eine ganz bestimmte Information beschafft werden kann. Es zeigt eine Version des Sperrens einer Seite vor dem direkten Aufruf der Adresse:

**Servervariablen
anwenden**

```
<script language="vb" runat="Server">
void Page_Load()
Public Sub Page_Load (sender As Object, e As EventArgs)
    Dim referer As String ←
        = Request.ServerVariables("HTTP_REFERER")
    Dim scriptname As String ←
        = Request.ServerVariables("SCRIPT_NAME")
    Dim servername As String ←
        = Request.ServerVariables("SERVER_NAME")
    Dim checkrefer As String ←
        = String.Format("http://{0}{1}", servername, scriptname)
    If referer = checkrefer Then
        message.Text = "OK, du bist hier richtig"
    Else
```

```

        message.Text = "Leider falsch. Du kamst von: " + referer
    End If
End Sub</script>
<html>
    <head>
        <title>Request.ServerVariables</title>
    </head>
    <body>
        <asp:label id="message" runat="Server"/>
        <hr/>
        <a href="RequestServerVariable.aspx">Restart</a>
    </body>
</html>

```

Listing 8.7: Schutz einer Seite durch Feststellung der Herkunft des hinführenden Links (RequestServerVariable.aspx)

Wie es funktioniert

Hier wird zuerst die Herkunft abgefragt, indem die Servervariable `HTTP_REFERER` ermittelt wird:

```
Dim referer As String = Request.ServerVariables("HTTP_REFERER")
```

Dann wird der erwartete Wert mit dem tatsächlichen verglichen und eine Nachricht auf der Seite erzeugt. Wenn Sie das Programm von irgendwo starten, ist die Herkunft natürlich nicht die Seite selbst. Es wird die Fehlermeldung angezeigt. Klicken Sie dann auf den Link, ruft sich die Seite selbst auf und die Bedingung stimmt. Im Beispiel wird das laufende Programm, das sich hier selbst identifizieren soll, aus den Werten der Servervariablen `SCRIPT_NAME` und `SERVER_NAME` ermittelt. Diese Werte kann auch ein böswilliger Benutzer nicht ändern, daher ist der Vergleich mit `HTTP_REFERER` äußerst zuverlässig.

Entsprechend dem Ergebnis ändert sich im Beispiel die Meldung. In der Praxis könnte hier eine Umleitung auf die Startseite oder eine Anmeldeprozedur erfolgen.

8.3.5 Den Inhaltstyp bestimmen

Neben der Datenübertragung stellt sich auch die Frage, was eigentlich übertragen wird. Browser stellen nicht nur HTML, sondern auch die im Quelltext angegebenen Bilder, Plug-Ins und Applets dar oder dienen dem Herunterladen von Word- oder Exceldateien bis hin zu ZIP-Archiven.

Standard für den Inhalt der Seite: MIME

Der `Content-Header` enthält Informationen darüber, welche Art Datei zu erwarten ist. Die Kodierung erfolgt nach dem MIME-Standard. Mögliche Angaben sind »text/html«, »image/gif«, »application/msword« usw. Mit der Eigenschaft `ContentType` können Sie die Werte setzen. Der häufigste Wert ist vermutlich »text/HTML«, der für jede Übertragung einer ASP.NET- oder HTML-Datei eingesetzt wird. Dieser Header muss nicht explizit erzeugt werden, dies erledigt ASP.NET für Sie.

**Response.
ContentType
MIME**

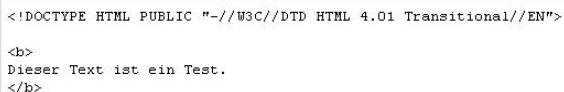
Sie können aber auch einfachen Text übertragen. Wenn Sie beispielsweise ein Lernprogramm über HTML schreiben, dann kann es unter Umständen sinnvoll sein, einen Quelltext im Browser des Studenten anzuzeigen und ihn anschließend zur Ausführung zu bringen. Aber wie wird dem Browser HTML abgewöhnt? Setzen Sie dazu einfach den Header *Content-type* auf Text, wie es im folgenden Beispiel gezeigt wird.

```
<script language="vb" runat="Server">
Public Sub Page_Init (sender As Object, e As EventArgs)
    Response.ContentType = "text/plain"
End Sub</script>
<b>
<%
Response.Write("Dieser Text ist ein Test.")
%>
</b>
```

Listing 8.8: Ausgabe von Text im Browser forcieren (ResponseContenttype.aspx)

Die Ausgabe zeigt, dass der Internet Explorer sich von der Angabe überzeugen ließ und die Seite nicht als HTML interpretiert. Allerdings gibt es einen zweiten Erkennungsmechanismus, der auf die umschließenden `<html>`-Tags reagiert (die hier bewusst weggelassen wurden). Insofern ist der Internet Explorer nicht völlig von diesem Header abhängig. Testen Sie Ihre Applikation mit mehreren Browsern, um die Reaktion kennenzulernen.

Die folgende Abbildung zeigt den Effekt:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<b>
Dieser Text ist ein Test.
</b>
```

Abbildung 8.9: Ausgabe einer HTML-Seite als einfachen Text

Was ist eigentlich MIME?

MIME ist das Akronym für *Multipurpose Internet Mail Extensions*. Trotz der leicht irreführenden Bezeichnung definiert MIME die Struktur und den Inhalt vieler verschiedener Internetnachrichten, nicht nur E-Mail. MIME ist ein Protokoll, das den Versand von Nicht-Text-Dokumenten, wie Audio, Video, Graphik etc. in textbasierten Protokollen, wie E-Mail oder Usenet, unterstützt. Nicht-Text-Elemente werden kodiert und nach dem Empfang wieder dekodiert.

MIME bietet drei Mechanismen, die es ermöglichen, Nicht-Text-Daten anzugeben und in Textdokumente einzubinden:

1. MIME-Version
2. Content-type
3. Content-transfer-encoding

Zusätzlich können zwei weitere optionale Header-Felder definiert werden, die dem Körper (Body) der Nachricht eine eindeutige ID und eine Beschreibung zuweisen:

1. Content-ID
2. Content-Description

Nachrichten, die MIME-kompatible Daten enthalten, müssen das Header-Feld MIME-Version (RFC 2045) setzen. Das Setzen dieses Feldes ist ein Versprechen, dass die Spezifikation eingehalten wurde. E-Mail-Programme können somit Nachrichten, die mit älterer oder inkompatibler Software erzeugt wurden, von kompatiblen Nachrichten unterscheiden. Beispiele für solche Textzeilen sind:

```
MIME-Version: 1.0
MIME-Version: 1.0 (produced by MetaSend Vx.x)
MIME-Version: (produced by MetaSend Vx.x) 1.0
MIME-Version: 1.(produced by MetaSend Vx.x)0
```

Die Spezifikation der MIME-Version darf Kommentare enthalten, deren Formatierung den Angaben in RFC 822 entsprechen muss. Die vier angeführten Beispiele sind daher als identisch zu betrachten.

Das Inhaltstyp-Kopf-Feld spezifiziert den Typ (major) und Untertyp (minor) der Daten im Körper der Nachricht. Dieses Feld wird verwendet, um anzuzeigen, welche Art von Daten versandt wurde. Derzeit sind sieben Haupttypen definiert:

1. text

Umfasst Textdaten, die in der Regel vom Menschen gelesen werden. Typische Beispiele sind:

- ▶ text/html
HTML-Dateien
- ▶ text/plain
Einfacher Text ohne jede Formatierung
- ▶ text/richtext
RTF-Formatierter Text (RTF = Rich Text Format)
- ▶ text/x-vCard
vCard-Datensätze, beispielsweise Kontaktadressen aus Outlook.

2. application

Alle Datentypen, die nicht in die übrigen Kategorien passen:

- ▶ application/excel
- ▶ application/msword
- ▶ application/pdf
- ▶ application/zip

3. image

Einfache Bilder:

- ▶ image/gif
- ▶ image/jpeg
- ▶ image/png
- ▶ image/tiff

4. audio

Audio-Daten:

- ▶ audio/basic
- ▶ audio/midi
- ▶ audio/mpeg

5. video

Video-Daten und bewegte Bilder:

- ▶ video/quicktime
- ▶ video/x-msvideo
- ▶ video/x-sgi-movie

6. message

Weitere verschachtelte Mail- oder MIME-Nachrichten:

- ▶ message/rfc822
Eine eingefügte, weitere E-Mail-Nachricht, beispielsweise bei einer Weiterleitung.
- ▶ message/news
Eine Nachricht aus dem Usenet
- ▶ message/partial
Ein Teil einer Nachricht

7. multipart

Verschiedene Datentypen sind in einer Nachricht vereint:

- ▶ multipart/alternative
Gleicher oder ähnlicher Inhaltstyp
- ▶ multipart/mixed
Mehrfache Inhaltstypen

Das Kopf-Feld `Content-Transfer-Encoding` wird verwendet, um die zusätzliche Kodierung anzugeben, die auf die Daten angewendet wurde. Die zusätzliche Kodierung wird benötigt, um 8-Bit-Daten in 7-Bit-Text umzuwandeln. Derzeit werden hauptsächlich folgende fünf Typen verwendet:

1. 7bit

7-bit Text. Es wird überhaupt nichts kodiert. Dieses Kopffeld besagt, dass die Nachricht nur aus Zeichen im 7-bit-ASCII-Code besteht und nicht mehr als 1.000 Zeichen (inklusive Zeilenumbrüche) enthält.

2. 8bit

8-bit Text. Es wird überhaupt nichts kodiert. Dieses Kopffeld besagt, dass die Nachricht nur aus Zeichen im 8-bit-ASCII-Code besteht und nicht mehr als 1.000 Zeichen enthält (inklusive Zeilenumbrüche). Sie können nicht davon ausgehen, dass alle Mail-Gateways diese Kodierung verstehen.

3. binary

8-bit Text. Es wird überhaupt nichts kodiert. Alle Zeilen zusammen können mehr als 1.000 Zeichen enthalten (inklusive Zeilenumbrüche). Sie können nicht davon ausgehen, dass alle Mail-Gateways diese Kodierung verstehen.

4. quoted-printable

8-bit Text oder mehr als 1.000 Zeichen. Eine Standard-Methode, die alle Zeichen in 7-bit-Code umwandelt. Diese Methode ist besonders geeignet, um Text-Daten in 8-Bit-ASCII-Code (beispielsweise bei Verwendung eines 8-Bit-Zeichensatzes) zu übertragen. Quoted-printable ist wesentlich verlässlicher als »8Bit« oder »binary«.

5. base64

Eine Standardmethode, die binäre Daten in 7-Bit-ASCII umwandelt. Base64 ist die geeignete Methode zum Versenden von binären Daten, beispielsweise GIF-, ZIP- oder Word-Dateien.

8.3.6 Anforderungen weiterleiten

Es gibt Situationen, in denen eine Seite beendet wird und die nächste oder eine bestimmte Seite automatisch erreicht werden soll. Die Methode `Redirect` des Objekts `Response` löst dieses Problem. Intern wird ein Header mit dem Namen *Location* erzeugt.

**Weiterleitungen
intern**

Die Methode `Redirect` benutzt einen speziellen Statuscode des Webserver, den Code »302 Object Moved«. Jede Anforderung eines Browsers wird mit einem bestimmten Statuscode beantwortet. Der Browser entscheidet dann, wie damit zu verfahren ist. Wenn die `Response.Redirect`-Methode aufgerufen wird, sendet der Webserver zuerst die Antwort »302 Object Moved« an den Browser. Der neue URL wird mitgeliefert. Normalerweise sollten alle Browser diesen dann ansteuern. Um zu verstehen, wie `Redirect` funktioniert, lesen Sie die folgenden Zeilen Code, die exakt die Funktionsweise wiedergeben:

```
Response.Status = "302 Object Moved"
Response.AppendHeader = "Location", ↵
                        "http://www.ziel.de/ziel.aspx"
```

Response.Redirect

Einfacher ist natürlich die folgende Schreibweise:

```
Response.Redirect
```

Das folgende Beispiel besteht aus zwei Listings. Das erste zeigt die Seite, die nur mit einem speziellen Link erreicht werden kann. Dieser muss zuvor in einem Formular eingegeben werden. Ist das nicht erfolgt, wird eine Weiterleitung eingesetzt, um zu diesem Formular zu gelangen:

```
<script language="vb" runat="Server">
void Page_Init()
Public Sub Page_Init (sender As Object, e As EventArgs)
    If Request.Form("name") <> "Demo" Then
        Response.Redirect("ResponseRegister.aspx")
    End If
End Sub</script>
<html>
<head>
    <title>Response.Redirect</title>
</head>
<body>
    <h3>Die Anmeldung war korrekt!</h3>
</body>
</html>
```

Listing 8.9: Einsatz einer Weiterleitung (ResponseRedirect.aspx)

Das Ziel besteht nur aus HTML – ein einfaches Formular:

```
<html>
<head>
    <title>Response.Redirect</title>
```

```
</head>
<body>
  Ihre Anmeldung bitte;<br/>
  <form method="post" action="ResponseRedirect.aspx">
    Ihr Name: <input type="text" name="name" />
    <input type="Submit" value="Anmelden" />
    (Tipp: Geben Sie das Wort "Demo" ein)
  </form>
</body>
</html>
```

Listing 8.10: Formular zur Erfassung des Anmeldenamens (ResponseRegister.aspx)

Obwohl als Ziel hier mit `action="ResponseRedirect.aspx"` eindeutig auf die zuvor gezeigte Seite verwiesen wird, bleibt das Formular sichtbar, bis der Name korrekt eingegeben wurde (im Beispiel »Demo«). Tatsächlich springt der Browser mittels `Redirect` zwischen den beiden Seiten hin und her. Dabei wird auf das Ziel verzweigt, dort wird dann der Wert überprüft und entschieden, ob der Vorgang auf der Seite fortgesetzt wird oder ein Rücksprung an das Formular erfolgen muss. Ist Letzteres der Fall, wird mit `Response.Redirect` der Browser zur Anforderungen des ursprünglichen URL aufgefordert.

Wie es funktioniert

8.3.7 Übergabe der Programmsteuerung

Die im letzten Abschnitt gezeigte Technik mit `Response.Redirect` ist gut geeignet, um den Benutzer dauerhaft auf eine andere Site zu leiten. Umfangreiche Anwendungen benötigen jedoch andere Verfahren. Im Objekt `Server` sind zwei Methoden zu finden, die einmal die Ausführung direkt an eine andere Seite übertragen (Transfer) oder ein anderes Programm ausführen und dann im ursprünglichen fortsetzen (Execute).

Die Steuerung weitergeben: `Server.Transfer`

`Response.Redirect` leitet die Übertragung der Ausführung auf ein anderes Programm über eine Anforderung des Browsers ein. Ein solcher Umweg hat den Nachteil, dass Zeit und Bandbreite verloren gehen. Außerdem hat der Client Einfluss auf den Ablauf. Dafür kann die Übertragung auch auf eine andere Site im Internet außerhalb der eigenen Domäne erfolgen. Letzteres ist mit `Server.Transfer` nicht möglich. Nach der Ausführung der Methode

Server.Transfer

wird die Codeausführung mit der aufgerufenen Seite fortgesetzt, eine direkte Rückkehr ist nicht möglich.

Browsertyp Das folgende Beispiel zeigt, wie die Methode eingesetzt werden kann. Die Ausführung wird hier anhand des Browsertyps gesteuert:

```
<script language="vb" runat="server">
Public Sub Page_Init (sender As Object, e As EventArgs)
    Dim bc As HttpBrowserCapabilities = Request.Browser
    If bc.Type = "IE6" or bc.Type = "IE5" Then
        Server.Transfer("ServerTransferIE.aspx?" + bc.Type)
    Else
        Server.Transfer("ServerTransferOther.aspx?" + bc.Type)
    End If
End Sub
</script>
```

Listing 8.11: Übertragung der Programmausführung in Abhängigkeit vom Browsertyp (ServerTransfer.aspx)



Ausführlichere Informationen zur browserabhängigen Programmierung finden Sie im Abschnitt 7.3, »Browserorientiert programmieren« ab Seite 662.

Beispielhaft wird hier eine der Antwortseiten gezeigt, die den *QueryString* auswertet, um an der Information über den erkannten Browsertyp zu partizipieren.

```
<script language="vb" runat="server">
Public Sub Page_Load (sender As Object, e As EventArgs)
    bctype.Text = Request.QueryString.ToString()
End Sub
</script>
<html>
    <head>
        <title>ServerTransferIE</title>
    </head>
    <body MS_POSITIONING="GridLayout">
        <h1>ServerTransfer</h1>
        Es wurde der Internet Explorer mit folgender Kennung erkannt:
        <asp:Label Runat="server" ID="bctype"/>
    </body>
</html>
```

Listing 8.12: Auswertung des Browsertyps auf der anderen Seite (ServerTransferIE.aspx)

Die Ausgabe ist wenig spektakulär. Bei `Response.Redirect` konnte der Benutzer den Umschaltprozess beobachten. Der Browser zeigt

te den Namen der aufgerufenen Seite in der Adresszeile an, bis die Weiterleitung ausgeführt wurde. Dann erscheint der neue Name. Bei der Übertragung der Ausführung mit `Server.Transfer` ist der Prozess nicht zu bemerken – sogar der alte Seitenname bleibt bestehen.



Abbildung 8.10: Alter URL und neuer Inhalt. So arbeitet `Server.Transfer`

Eine Anwendung mit `Server.Transfer` finden Sie in Abschnitt 8.3.8, »Kontext-Handler und Seitenreferenzierung« ab Seite 773.



Im Gegensatz dazu arbeitet `Server.Execute` wie der Aufruf eines Unterprogramms.

Die Steuerung zeitweilig delegieren: `Server.Execute`

Diese Methode ruft innerhalb einer ASP.NET-Seite eine andere auf, führt diese aus und setzt nach dem Aufruf an der ursprünglichen Stelle fort. Ausgaben, die während der neuen Aufforderung entstehen, werden an der Stelle des Aufrufes ausgegeben. Standardmäßig wird der Dateiname einer auszuführenden Seite angegeben:

Server.Execute

```
Server.Execute("name.aspx")
```

Die Methode ist allerdings mit einem zweiten Parametersatz überladen. Dabei ist die Angabe eines `StringWriter`-Objekts erlaubt, das den erzeugten Datenstrom aufnehmen kann. Dadurch wird der Inhalt manipulierbar. Das folgende Beispiel nutzt die Methode, um HTML-Code aus einer anderen Datei in Abhängigkeit von bestimmten Bedingungen zu holen.

```
<%@ Page Language="vb" AutoEventWireup="true" %>
<%@ Import Namespace="System.IO" %>
```

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<script language="vb" runat="server">
Public Sub Page_Load (sender As Object, e As EventArgs)
    Dim result As StringWriter = New StringWriter()
    Dim self As String = Request.ServerVariables("SCRIPT_NAME")
    link1.HRef = self + "?start=first"
    link2.HRef = self + "?start=next"
    If Request.QueryString("start") = "next" Then
        Server.Execute("ServerExecuteOther.aspx", result)
    Else
        Server.Execute("ServerExecuteFirst.aspx", result)
    End If
    welcome.Text = result.ToString()
End Sub
</script>
<html>
    <head>
        <title>ServerExecute</title>
    </head>
    <body MS_POSITIONING="GridLayout">
    <h1>Willkommen auf unserer Website</h1>
    <p>
    <asp:Label ID="welcome" Runat="server"/>
    </p>
    Wir möchten Sie auf unseren Seiten über
    Aktuelles aus unserem Unternehmen informieren.
    Wählen Sie eine Information:
    <ul>
        <li><a id="link1" runat="server" >Erste Site</a>
        <li><a id="link2" runat="server" >Kontaktseite</a>
        <!-- weitere Links -->
    </ul>
    </body>
</html>

```

Listing 8.13: Einbinden externer Daten mit `Server.Execute` (`ServerExecute.aspx`)

Wie es funktioniert

In den beiden Dateien *otherse.aspx* und *firstse.aspx* steht nur HTML-Code, kein Programmcode. Beim ersten Aufruf der Seite ist kein Parameter im Querystring. Die folgende Prüfung misslingt deshalb:

```
If Request.QueryString("start") = "next" Then
```

Nun wird der Else-Zweig ausgeführt und die Daten aus *firstse.aspx* werden an das `StringWriter`-Objekt übergeben.

Der Zugriff auf das Dateisystem soll hier nicht weiter beschrieben werden. Dateioperationen dieser Art werden in Abschnitt 4.11, »Zugriff auf das Dateisystem« ab Seite 329 genauer betrachtet.



Ein `StringWriter`-Objekt kann Text enthalten, in diesem Fall die Ausgabe der Ausführung von `ServerExecuteFirst.aspx`. Das ist in diesem Fall der dort untergebrachte Quelltext. Dieser Text wird dann an das `Label`-Steuerelement `welcome` übergeben:

```
welcome.Text = result.ToString()
```

Der Text erscheint an der Stelle, wo das `Label`-Steuerelement definiert wurde:

```
<asp:label id="welcome" runat="server"/>
```

Damit sich die Seite selbst aufruft, wurde zuvor der Seitenname ermittelt:

```
Dim self As String = Request.ServerVariables("SCRIPT_NAME")
```

Dieser wird dem `<a>`-Tag im HTML-Teil zugewiesen:

```
link2.HRef = self + "?start=next"
```

Entscheidend ist hier der Parameter, der angehängt wird und beim nächsten Aufruf zur Ausführung der anderen Datei führt.

8.3.8 Kontext-Handler und Seitenreferenzierung

Bislang war die Rede immer von nur einem Formular, das sich selbst aufruft und dabei Status und Inhalt erhält. Bei der Weitergabe der Daten an eine andere Seite geht der Status verloren. ASP.NET bietet hier eine neue Technik des Zusammenhalts zwischen den Bausteinen einer Site: den Kontext.

Die bisherigen Beispiele waren in einer Beziehung praxisfern: Sie liefen alle in einer einzigen Seite ab. All die schönen Verfahren wie die Auswertung des *PostBack* oder die Verwendung des Anzeigestatus funktionieren nämlich nur dann so einfach, wenn sich Seiten immer wieder selbst aufrufen. Ungeachtet der auch damit erreichbaren Komplexität ist es in der Praxis wünschenswert, nicht an eine derartige Einschränkung gebunden zu sein. Abgesehen vom Rückfall in alte Zeiten und der direkten Nutzung der Formulardaten, wie das mit dem alten ASP nötig war, ist also eine neue Technologie erforderlich.

Das Prinzip der Kontextverwaltung

Im Objektmodell von ASP.NET befindet sich unter anderem eine Klasse `HttpContext`. Diese fasst den gesamten aktuellen Vorgang, also die Sitzung, Seiten und Daten pro Benutzer zusammen. Einen Überblick über das gesamte Objektmodell wurde in Abschnitt 8.2, »Die Welt der Standardobjekte« ab Seite 747 und dort insbesondere in Abbildung 8.2 vorgestellt.

Nur für interne Seitenübergänge

Der Kontext existiert durch ein der Anfrage (dem Request vom Browser) zugeordnetes Objekt. Es wird von `Page` und von `HttpApplication` implementiert, sodass Sie darüber auf die Seiten und Applikationsinformationen Zugriff haben. An dieser Stelle interessiert nur die Seite. Wenn Sie mit Formularen arbeiten, die sich über zwei Seiten erstrecken, und die Übergabe zwischen den Seiten mit `Server.Execute` oder `Server.Transfer` erfolgt, hält das `Context`-Objekt diese quasi zusammen.

Nun wurde am Anfang schon festgestellt, dass der Ablauf des Lebenszyklus einer Seite ein in sich geschlossener Vorgang ist. Am Ende, wenn die Seite verarbeitet und die Daten gesendet wurden, werden alle Objekte entsorgt und der Inhalt geht verloren. Als Speicher für nutzerabhängige Zustände kam nur der Anzeigestatus in Betracht. Später lernen Sie noch Sitzungsvariablen kennen, die in einem eigenen Speicherbereich liegen. Außerdem kämen als Speicher noch Cookies in Betracht, aber das ist vermutlich die schlechteste Lösung.

Funktionsweise

Das `Context`-Objekt ist lediglich in der Lage, den Zusammenhang zwischen zwei Seiten und damit den in den Klassen enthaltenen Daten zu erhalten, wenn ein innerer Seitenübergang erfolgt. Dazu wird in der zweiten Datei eine Referenz definiert und eine Instanz der Klasse der ersten Datei übernommen. So erhalten Sie jedoch keine neue Instanz, sondern Zugriff auf die bereits bestehende. Das klingt komplizierter als es ist. Ein einfaches Beispiel soll zeigen, wie es funktioniert.

Auswertung von Formulardaten auf einer zweiten Seite

Das folgende Beispiel zeigt ein einfaches Formular. Die Daten werden dann, wenn das Formular fertig ausgefüllt wurde, an eine zweite Seite übertragen und dort ausgewertet.

Die Übertragung wird mit `Server.Transfer` vorgenommen. Mehr Informationen dazu finden Sie im Abschnitt »Die Steuerung weitergeben: `Server.Transfer`« ab Seite 769. Diese Methode übergibt die Steuerung ohne den Umweg über den Client zu nehmen. Entsprechend schnell und einfach ist die Verwendung. Da die nächste Seite eine eigene Klasse verwendet, muss es natürlich einen Zugriff auf die Vorhergehende geben. Das ganze Beispiel besteht aus vier Dateien.

**Server.Transfer
verwenden**

```
<h1>Ein mehrseitiges Formular</h1>
Geben Sie bitte Namen und Ort an:
<form id="ContextForm1" method="post" runat="server">
Name: <asp:TextBox Runat="server" ID="FieldName" />
<asp:RequiredFieldValidator Runat="server"
    ControlToValidate="FieldName"
    ErrorMessage="Angabe erforderlich"
    ID="Requiredfieldvalidator1" />
<br/>
Ort: <asp:TextBox runat="server" ID="FieldCity" />
<asp:RequiredFieldValidator Runat="server"
    ControlToValidate="FieldCity"
    ErrorMessage="Angabe erforderlich"
    ID="Requiredfieldvalidator2" />
<br/>
<asp:Button Text="Weiter >" Runat="server"
    ID="Submit" OnCommand="Submit_Click"/>
</form>
```

Listing 8.14: HTML-Teil des Formulars (ContextForm1.aspx)

```
Public Class ContextForm1
    Inherits System.Web.UI.Page
    Protected WithEvents RequiredFieldValidator1 As RequiredFieldValidator
    Protected WithEvents FieldName As TextBox
    Protected WithEvents FieldCity As TextBox
    Protected WithEvents RequiredFieldValidator2 As RequiredFieldValidator
    Protected WithEvents Button1 As Button

    Public FieldNameData As String
    Public FieldCityData As String

    Private Sub Button1_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles Button1.Click
        If Page.IsValid Then
            FieldNameData = FieldName.Text
```

```

        FieldCityData = FieldCity.Text
        Server.Transfer("ContextForm2.aspx")
    End If
End Sub
End Class

```

Listing 8.15: Code-Datei des Formulars (ContextForm1.aspx.vb; Page_Load wird nicht benötigt)

Hier werden lediglich die Felddaten in öffentlichen Variablen der Klasse gespeichert. Anschließend verzweigt das Programm serverseitig auf die zweite Seite:

```

<h1>Formularauswertung</h1>
Folgende Daten wurden erfasst:<br/>
Name: <asp:Label Runat="server" ID="LabelName"/><br/>
Ort: <asp:Label Runat="server" ID="LabelCity"/><br/>

```

Listing 8.16: HTML-Teil der Antwortseite (ContextForm2.aspx)

```

Public Class ContextForm2
    Inherits System.Web.UI.Page
    Protected WithEvents LabelName As Label
    Protected WithEvents LabelCity As Label

    Dim cf1 As ContextForm1

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Page.IsPostBack Then
            cf1 = CType(Context.Handler, ContextForm1)
            LabelName.Text = cf1.FieldNameData
            LabelCity.Text = cf1.FieldCityData
        End If
    End Sub
End Class

```

Listing 8.17: Code-Datei der Antwortseite (ContextForm2.aspx.vb)

Wie es funktioniert

Der spannende Teil besteht jetzt im Zugriff auf die Klasse mit den gespeicherten Daten der ersten Seite. Die Seite ist noch Bestandteil des Kontextes, denn durch die Übertragung wurde die aktuelle Anforderung vom Browser nicht beendet und damit besteht der Zusammenhang mit der ersten Seite fort. Programmtechnisch kann darauf über die Eigenschaft `Handler` zugegriffen werden:

Context.Handler `cf1 = CType(Context.Handler, ContextForm1)`

Die im vorhergehenden Listing definierten öffentlichen Variablen stehen – wie alle Teile der Klasse – nun in der zweiten Seite zur Verfügung:

```
LabelName.Text = cf1.FieldNameData
```

Die folgende Abbildung veranschaulicht die Vorgänge nochmals.

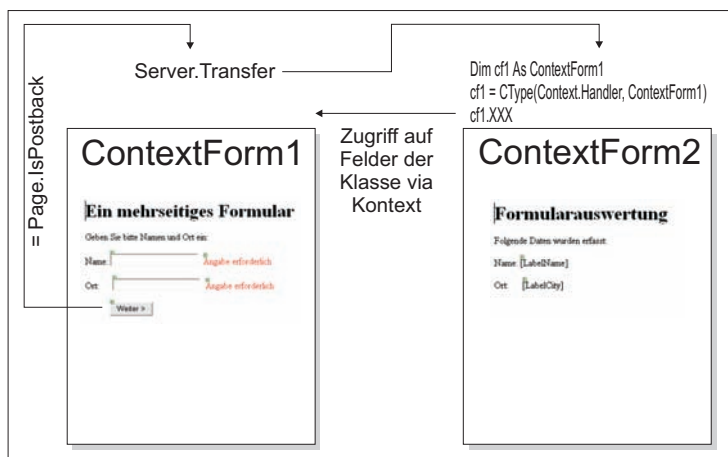


Abbildung 8.11: Seitenübergang mit *Server.Transfer* und *Context.Handler*

Typische Probleme

Der Umgang damit ist nicht ganz unproblematisch. So können Sie nur dann auf die Klasse zugreifen, wenn der Browser noch keine erneute Anforderung gesendet hat. Wenn Sie also auf der zweiten Seite noch eine Bestätigung programmieren und diese vom Nutzer per POST senden lassen, geht der Zusammenhang endgültig verloren, denn es entsteht ein neuer Kontext. Die Instanziierung der Klasse gelingt nicht mehr. Freilich können Sie dennoch das Formular erneut versenden, nur muss es dann andere Wege des Zugriffs geben. Insofern ist die Zeile `If Not Page.IsPostBack` bereits eine Vorbereitung darauf, denn sie sichert ab, dass keine Fehlermeldung auftritt, wenn Sie dennoch ein Formular einbauen.

PostBack beachten

Die bisherigen Programme verwendeten Code Behind. Wenn Sie nur mit *aspx*-Seiten arbeiten, funktioniert es auch, aber es gibt einige Hinweise zu beachten. Sie müssen auf der zweiten Seite, also dem Ziel des `Server.Transfer`, eine Referenzierung einbauen:

Kein Code Behind verwendet?

```
<% @Reference page="ContextForm2.aspx" %>
```

Wenn Sie nun in der zweiten Seite auf die Klasse zugreifen möchten, müssen Sie deren Namen kennen. Mit hinterlegtem Code ist das einfach, da Sie den Klassennamen selbst festlegen. Sie können ein Attribut der @Page-Direktive verwenden, um den internen Namen zu überschreiben:

```
<% @Page ... ClassName="ContextForm1" ... %>
```

Damit funktioniert auch der Zusammenhang.

Benutzer-Steuer- elemente

Nur als Ergänzung sei hier angemerkt, dass in das Verfahren auch Benutzer-Steuerelemente einbezogen werden können. Die Referenzierung erfolgt dann folgendermaßen:

```
<% @Reference control="ContextForm2.aspx" %>
```

Kontext-bezogener Datenaustausch

Wenn Sie mit öffentlichen Variablen oder mühevoll zu programmierenden Eigenschaften nicht glücklich sind, können Sie auch eine für den Datenaustausch verwendbare Kollektion innerhalb des Context-Objekts verwenden. Folgendermaßen werden Werte darin gespeichert:

```
Context.Items.Add("ContextVariable", "234")
```

Und mit der nächsten Zeile holen Sie den Wert wieder heraus:

```
Context.Items("ContextVariable").ToString()
```

Im Übrigen verhält sich die Kollektion wie viele andere in .NET und kann mit den entsprechenden Standardmethoden und -eigenschaften bedient werden. Damit ist auch die in VB.NET übliche Kurzschreibweise möglich: `Context("Variable") = "Wert"`.

Das Speichern der Variablen erfolgt immer vor dem Server-Transfer, die Entnahme dann am Zielort. Auch hier gilt natürlich: Wird die Seite erneut vom Nutzer angefordert, entsteht ein neuer Kontext und alle Variablen und Zusammenhänge sind verloren.

8.4 Sitzungen (Sessions)

Wie bei der HTTP-Beschreibung bereits angedeutet, ist die Wiedererkennung von Benutzern nicht Bestandteil dieses Protokolls.

Die Zeit vom Abruf der ersten Seite durch einen bestimmten Benutzer bis zum Verlassen der Domain wird als Sitzung (engl. Session) bezeichnet.

8.4.1 Grundlagen

ASP.NET verfügt über ein umfangreiches und ausgereiftes Sitzungsmanagement. Vom grundlegenden Prinzip her geht es darum, den Benutzer wiederzuerkennen, wenn er dieselbe oder eine andere Seite der Applikation abrufen. War die Wiedererkennung erfolgreich, können der Sitzung Daten zugeordnet werden, die für den gesamten Verlauf zur Verfügung stehen sollen, ohne dass eine explizite Übertragung von Seite zu Seite mittels URL-Querystring oder Formularen stattfindet.

Sitzungsmanagement

Da der eigentliche Ablauf über HTTP keine direkte Speicherung des Zustandes zulässt, sind im Laufe der Zeit verschiedene Methoden entwickelt worden, solche Zustandsinformationen zu erhalten. Sie werden nachfolgend erläutert.

Prinzip des Sitzungsmanagements

Das Sitzungsmanagement beruht darauf, einem Benutzer bei der ersten Anforderung eine eindeutige, willkürlich gewählte Identifikationszeichenfolge zuzuordnen und diese so bereit zu stellen, dass Sie bei künftigen Anforderungen desselben Benutzers immer wieder übertragen wird. Der Benutzer kann dann später an der Zeichenfolge erkannt werden. Die Identifikationszeichenfolge wird im Folgenden kurz als ID oder im Zusammenhang mit dem Sitzungsmanagement als Session-ID bezeichnet.

ID oder Session-ID

Die Übertragung der ID basiert auf einem von drei Wegen:

► Cookies

Die älteste und verbreitetste Methode der Statuserhaltung basiert auf Cookies¹. Cookies sind kleine Datenmengen, die der Server an den Browser sendet und die dieser meist in Form von Textdateien speichert. Ruft der Benutzer eine neue Seite von demselben Server ab, sendet der Browser die Informationen zurück, die dieser Server platziert hat. Wenn nun der Ser-

¹ Das war sogar der ursprüngliche Zweck, für den Cookies entwickelt wurden. Mehr dazu finden Sie im Abschnitt 8.5, »Cookies« ab Seite 791.

Probleme mit Cookies

ver im Cookie eine Identifikationsnummer sendet, die so genannte Session-ID, kann er den Benutzer anhand dieser Nummer wiedererkennen.

Nun genießen Cookies einen zweifelhaften Ruf, denn es gibt zahlreiche Missbrauchsmöglichkeiten. Auch wenn Sitzungs-Cookies davon nicht betroffen sind, schalten viele Benutzer die Cookie-Option in ihrem Browser generell ab. Es wird also eine Alternative benötigt.

► HTTP-GET

Diese Methode erweitert den URL um einen entsprechenden Parameter, der als Attribut die ID enthält.

Die Erweiterung des URL bei HTTP-GET hat folgendes Erscheinungsbild:

```
http://www.server.der/aspcode.aspx?sid=8iw5fafd7902f
```

Viele Webprogrammiersysteme nutzen Automatismen, um GET-Parameter anzuhängen. Solche Eingriffe können jedoch – auch wenn diese Überlegung eher theoretischer Natur ist – in Konflikt mit eigenen Programmierschritten geraten. Was schwerer wiegt: Solche Systeme sind nie perfekt. Fast immer reicht ein wenig JavaScript, um auch hochwertige Parser bei der Suche nach den URL aus dem Konzept zu bringen. ASP.NET verwendet deshalb eine andere Technik: Eine Erweiterung des URL im Pfadteil.

► Einbettung als virtueller Pfad

Hierbei wird zwar auch GET als Übertragungsweg genutzt, die Sitzungs-ID jedoch im URL so verpackt, dass sie wie ein virtuelles Verzeichnis erscheint.

Diese Technik bietet mehrere Vorteile. Zum einen sind zusätzliche Parameter davon völlig unberührt. Zum anderen können Suchmaschinen – deren Robots auch Sitzungs-IDs zugeordnet werden – solchen Links problemlos folgen. Für die Indizierung dynamischer Webseiten ist das enorm wichtig.

Der URL sieht hier folgendermaßen aus:

```
http://www.server.der/(8iw5fafd7902f)/aspcode.aspx
```

► HTTP-POST

Wenn Formulare verwendet werden, wird ein verstecktes Feld erzeugt und zusammen mit den Formulardatei zurück gesendet. Solche Felder stören nicht das Layout der Seite und lassen sich leicht programmieren. Wenn das Feld als HTML Server-

Steuerelement programmiert wird, könnte sogar der Anzeige-status zum Erhalt des Sitzungsstatus verwendet werden. Dies setzt aber voraus, dass alle Anforderungen über POST laufen, was selten der Fall ist.

Im HTML-Code erscheinen solche Felder folgendermaßen:

```
<input type="hidden" name="sid" value="8iw5fafd7902f"/>
```

Prinzipiell versucht ASP.NET standardmäßig zuerst, Cookies einzusetzen. Hat der Benutzer seinen Browser so eingestellt, dass Cookies überhaupt nicht akzeptiert werden, wird der URL um die Sitzungsparameter ergänzt, sodass die nächste Anforderung per GET die Informationen wieder enthält. Dazu wird immer die Methode »Einbettung als virtueller Pfad« verwendet.

Ablauf

Natürlich können Sie die Verwendung von Cookies generell unterbinden, damit Benutzer mit rigiden Sicherheitseinstellungen nicht durch unnütze Alarmer beunruhigt werden. Wie die genauen Einstellungen aussehen und wo sie erfolgen, wird in Abschnitt 8.4.2, »Konfiguration des Sitzungsmanagements« ab Seite 786 gezeigt.

Ablageart von sitzungsgebundenen Daten

Das gezeigte reicht zwar, um den Benutzer zu erkennen; die mit ihm verbundenen Daten müssen aber auch gespeichert werden. Denn ohne diese Informationen ist das Sitzungsmanagement nicht sinnvoll. Die einfachen Beispiele in den vorhergehenden Kapiteln kamen ganz ohne aus, denn dort wurden verschiedene Benutzer nicht mit verschiedenen Informationen versorgt. Für die Ablage der Daten gibt es mehrere Wege:

Wo Daten gespeichert werden

- ▶ Gemeinsame Speicherbereiche
- ▶ Interne Speicherbereiche
- ▶ Dateien
- ▶ Datenbanken

ASP.NET verwendet drei der vier möglichen Methoden. Die häufigste Verwendung finden gemeinsame Speicherbereiche mit dem IIS (In-Process). Hier werden die Variablen im Speicher abgelegt und bei erneuten Aufrufen zur Verfügung gestellt. Diese Methode ist sehr schnell, und mit Hilfe weiterer Maßnahmen gut skalierbar.

In-Process

Gegenüber Programmfehlern ist das Verfahren recht robust, denn die Speicherung erfolgt im Prozess des IIS, nicht im Workerprozess von ASP.NET. Stürzt ihre Applikation ab oder wird sie vom IIS recycelt, bleiben die Sitzungsdaten dennoch erhalten.

Wenn Sie mit einem Lastverteilungssystem arbeiten möchten, funktioniert die einfache Ablage der Daten in einem computerbezogenen Speicher nicht. Denn die in einem lokalen Speicher abgelegten Variablen stehen auf einem anderen System nicht zur Verfügung. Lastverteilungssysteme basieren aber darauf, eingehende Anfragen relativ frei oder nach einem bestimmten Schema auf die verfügbaren Server zu verteilen. Sie können bei der Programmierung nicht beeinflussen und auch nicht berücksichtigen, wie diese Verteilung stattfindet, denn sie läuft außerhalb des Computers ab. Es gibt verschiedene Lösungen für das Problem. So können intelligente Router, die Lastverteilungsfunktionen enthalten, die Verbindung zwischen Browser und Server erkennen und für eine bestehende Verbindung die Umschaltung der Server unterdrücken. Zuverlässiger sind Softwarelösungen, die auf den betreffenden Servern installiert werden, wie der Application Center Server von Microsoft. ASP.NET nutzt eine eigene, einfachere und zugleich leistungsfähigere Technik: Das *SessionState*-Modul.

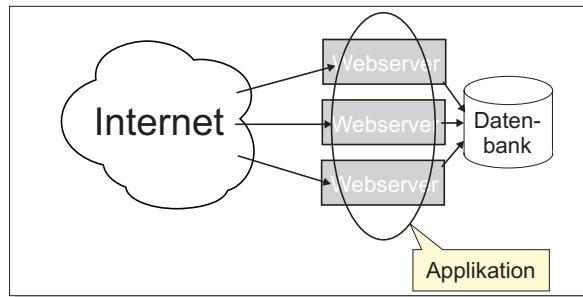


Abbildung 8.12: Bei der Lastverteilung arbeiten mehrere Webserver parallel

SessionState

Diese weitere Möglichkeit ist ein spezieller Dienst, der »Out-Of-Process«, also außerhalb des IIS läuft und die Verteilung über mehrere Server erlaubt. Dieser Dienst – *SessionState* – kommuniziert über TCP/IP mit anderen Webservern eines Serververbunds und tauscht Informationen über laufende Sitzungen aus. Alle Server verfügen deshalb ständig über dieselben aktuellen Sitzungsinformationen. Damit ist der Einsatz schneller externer Lastverteilungssysteme kein Problem. Der Dienst ist über die Konfiguration

der Applikation administrierbar. Wenn Sie nur einen Server verwenden, lohnt der Einsatz unter Umständen auch. Denn der Erhalt des Sitzungsstatus erfolgt nicht nur außerhalb des ASP.NET-Prozesses, sondern auch außerhalb des IIS. Ein Neustart des Webdienstes führt deshalb nicht zum Verlust der Sitzungsinformationen.

Die dritte Möglichkeit ist eine Datenbank, vorzugsweise wird dies der SQL Server 2000 sein. In diesem umfangreichsten, teuersten und leistungsfähigsten Szenario werden mehrere Webserver um einen SQL Server gruppiert. Die Lastverteilung verteilt die Anforderungen auf die Webserver, die ihrerseits Daten über den Datenbankserver austauschen. Der Datenbankserver selbst nutzt natürlich auch den Speicher zum schnellen Zugriff auf die Daten. Die Sitzungsinformationen werden so an einem zentralen Punkt gehalten und müssen über das Netzwerk nicht repliziert werden. ASP.NET unterstützt die Verwendung des SQL Servers direkt, indem in der Konfiguration eine Verbindungszeichenfolge angegeben wird, die auf Server und Datenbank verweist.

Datenbank nutzen

Ablageform der sitzungsgebundenen Daten

Die sitzungsgebundenen Daten werden in Form einer Kollektion gespeichert. Diese wird durch die `HttpSessionState`-Klasse bereitgestellt. Daneben bietet diese Klasse verschiedene Eigenschaften und Methoden, um über den Status der Sitzung zu wachen. Einen Überblick finden Sie nachfolgend. Aus der `HttpSessionState`-Klasse wird im Rahmen des Kontext des aktuellen Prozesses automatisch ein `Session`-Objekt abgeleitet, das in der praktischen Programmierung eine zentrale Rolle spielt.

Wie Daten gespeichert werden

Das `Session`-Objekt besitzt einige Eigenschaften, mit denen es überprüft werden kann:

Eigenschaften

► `IsCookieless`

Wenn die Eigenschaft `True` zurückgibt, werden Cookies nicht verwendet.

► `IsReadOnly`

Wenn die Sitzung als Nur-Lese-Sitzung gekennzeichnet wurde, wird `True` zurückgegeben. Ist Schreiben nicht erlaubt, ist ASP.NET in der Behandlung etwas schneller.

► Mode

Diese Eigenschaft enthält einen Wert einer Aufzählung, der angibt, welche Speichermethode verwendet wird. Mögliche Werte sind `Off`, `InProc`, `StateServer` oder `SQLServer`. `InProc` ist der Standardwert.

► IsNewSession

Wenn die Sitzung mit der aktuellen Anforderung erzeugt wurde, ist diese Eigenschaft `True`.

► SessionID

Diese Eigenschaft enthält die Session-ID.

► Timeout

Tragen Sie hier die Anzahl Minuten ein, die eine Sitzung ohne Aktivität laufen darf. Der Standardwert beträgt 20. Löst der Benutzer innerhalb dieser Zeit keine Anforderung aus, wird die Session-ID verworfen und bei einer dann folgenden Anforderung eine neue erzeugt.

Alle Eigenschaften außer `Timeout` sind nur lesbar. `IsReadOnly` kann über einen Parameter der `@Page`-Direktive gesteuert werden:

```
<% @Page EnableSessionState="ReadOnly" %>
```

Andere Parameter dieses Attributes sind `True` (aktiviert und schreibbar) bzw. `False` (deaktiviert).

Die vollständige Liste ist in der Referenz zu ASP.NET zu finden. Neben den für dieses Objekt spezifischen sind die für Kollektionen typischen Eigenschaften `Count`, `Item` und `Keys` zu finden. Allgemeine Informationen zum Umgang mit Kollektionen finden Sie im Abschnitt 4.3, »Aufzählungen und Kollektionen« ab Seite 222.

Methoden Das `Session`-Objekt kennt auch einige Methoden:

► Abandon

Abbruch der Sitzung

► Add

Fügt ein weiteres Objekt der Kollektion hinzu. `Session("Wert") = 13` entspricht `Session.Add("Wert", 13)`, weil in VB.NET die Kollektion über eine Standardeigenschaft verfügt.

► Clear,

Mit `Clear` werden alle Werte gelöscht, mit `RemoveAll` alle Objekte.

Im Kern geht es hier immer nur darum, ein Objekt aus der aktuellen Seite so zu speichern, dass es dem Benutzer auf der nächsten Seite wieder zugeordnet werden kann. Das `Session`-Objekt bietet dafür eine Kollektion von gespeicherten Objekten. Die Verwendung des Typs `Object` schränkt die Verwendung nicht ein – auch ganze *DataSets* lassen sich speichern.

Serialisierung

Einzige Voraussetzung ist, dass der Typ Serialisierung unterstützt. Die Serialisierung ist ein Vorgang, bei dem komplexe Datentypen als Zeichenkette kodiert werden, sodass später der ursprüngliche Typ mit den Daten wieder entstehen kann. Zeichenketten sind der einzige zuverlässige Weg der Speicherung außerhalb der ASP.NET-Komponente, denn so muss die Komponente den Speicherort selbst nicht kennen. Deshalb funktioniert das Verfahren mit den verschiedenen Speicherprinzipien, ohne dass sich in der Programmierung der Sitzungsvariablen selbst etwas ändert.

Wenn Sie eigene Klassen speichern wollen, müssen diese mit dem Attribut `<Serializable>` gekennzeichnet werden. So weit die Basisklassen dies akzeptieren, ist das Framework in der Lage, die aus der Klasse instanziierten Objekte komplett in eine serialisierte Form zu überführen und später daraus wieder zu restaurieren.

An Sitzungen gebundene Ereignisse

ASP.NET kennt für Sitzungen auch Ereignisse. Diese führen zur Ausführung von Methoden in der Datei *global.asax*, wenn sie definiert sind. Es gibt keinen Zwang, die Ereignisse zu verwenden oder anderweitig zu bedienen. Ein Anwendungsbeispiel und Informationen zum Umgang mit *global.asax* finden Sie im Abschnitt 8.6.2, »Die Datei *global.asax*« ab Seite 808. Verfügbar sind folgende Ereignisse:

Session-Ereignisse

► OnStart

Diese Ereignis wird ausgelöst, wenn eine neue Sitzung startet.

► OnEnd

Endet eine Sitzung, egal aus welchen Gründen, wird dieses Ereignis ausgelöst. In den meisten Fällen dürfte es sich um das Erreichen des Zeitlimits handeln.

Umgang mit Sitzungs-Cookies

Sitzungs-Cookies Die Sitzungs-Cookies sind in ASP.NET leicht zu erkennen, wenn Sie einen Cookie-Manager einsetzen. Im Netscape 6.2 ist dies besonders einfach.

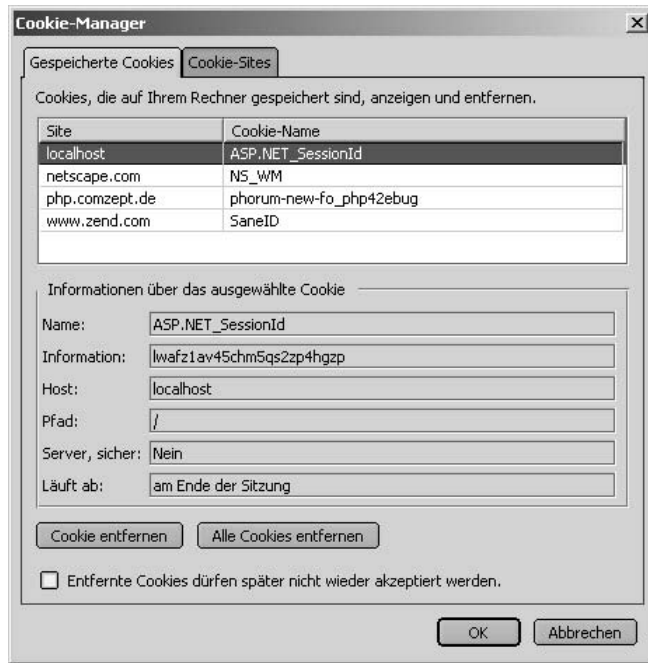


Abbildung 8.13: ASP.NET-Sitzungscookie im Cookiemanager

In der Session-ID selbst werden keine Daten gespeichert, sie dient nur der Wiedererkennung. Ansonsten weisen die von ASP.NET erzeugten Sitzungscookies keinerlei Besonderheiten auf.

8.4.2 Konfiguration des Sitzungsmanagements

Die Konfiguration des Sitzungsmanagements erfolgt entweder fur den gesamten Server in der Datei *machine.config* oder fur eine bestimmte Applikation in *web.config*.

Konfiguration des Abschnitts <sessionState>

Nachfolgend finden Sie die Standardkonfiguration, wie sie von ASP.NET nach der Installation verwendet wird:

```
<sessionState ❷
  mode="InProc" ❸
  stateConnectionString="tcpip=127.0.0.1:42424" ❹
  stateNetworkTimeout="10" ❺
  sqlConnectionString="data source=127.0.0.1;❻
                        user id=sa;password=" ❽
  cookieless="false" ❾
  timeout="20"
/>
```

Das wichtigste Attribut ist *mode*. Hiermit legen Sie fest, welche Methode zur Bestimmung des Ablageortes der Daten verwendet wird:

► Off

Das Sitzungsmanagement wird abgeschaltet.

► InProc

Der Standardwert, die integrierte Verwaltung mit Sitzungs-Cookies und der Speicherung der Werte im IIS-Prozess wird verwendet.

► StateServer

Der *SessionState*-Dienst wird verwendet, um die Daten so abzulagern, dass sie über mehrere Server verteilt werden können. Auch diese Methode verwendet zum Erhalt der Sitzung Cookies.

► SQLServer

Wenn der SQL Server verwendet wird, bleibt zwar auch die Sitzungserkennung mit Cookies erhalten, die Ablage der Daten erfolgt nun jedoch in einer Datenbank.

Um Cookies zu deaktivieren, setzen Sie folgendes Attribut:

```
cookieless="true"
```

**Cookies
deaktivieren**

Die Methode führt zur Erweiterung des virtuellen Pfades, sodass keine Manipulation des HTML-Codes stattfindet. Der Benutzer kann damit auch nicht in böswilliger Absicht den Wert manipulieren. Sehr wohl kann er die Session-ID sehen (siehe Abbildung 8.14).



Abbildung 8.14: So erscheint die Session-ID bei einer cookielosen Sitzung.

TimeOut Der dritte wichtige Parameter ist der Zeitüberschreitungswert. Standardmäßig dauert eine Sitzung 20 Minuten. Es hängt von Ihrer Applikation ab, ob Sie diesen Wert ändern müssen:

```
timeout="30"
```

Denken Sie aber daran, dass es sich hier um den maximal zulässigen Zeitraum ohne Benutzeraktivität handelt. Solange der Benutzer aktiv ist, wird die Sitzung fortgeführt. Einen Abbruch müssten Sie programmtechnisch mit der Methode `Abandon` erzwingen. Die Frage, die Sie sich stellen müssen, lautet: »Wie lange bleibt ein aktiver Benutzer auf meiner Seite, ohne eine neue Ressource anzufordern?«

Die beiden übrigen Attribute konfigurieren den `SessionState`-Dienst bzw. den SQL Server. Sie müssen für den Webserver eine feste IP-Nummer verwenden. Teilen Sie dem Dienst die aktive Nummer folgendermaßen mit:

```
stateConnectionString="tcpip=192.168.100.22:42424"
```

Ebenso ist der Port festzulegen, der Standardwert ist 42424.

Die Konfiguration des SQL Servers für Sitzungsdaten

Sitzungsdaten im SQL Server 2000

Bleibt zuletzt die Nutzung des SQL Servers. Sie müssen natürlich über eine Datenbank verfügen, die so eingerichtet ist, dass die Sitzungsdaten auch gespeichert werden können. Dazu wird ein Installations-Skript mitgeliefert, das Sie in folgendem Pfad finden:

```
%systemroot%\Microsoft.NET\Framework\v1.0.3705\InstallSql-  
State.sql
```

Um das Skript auszuführen, starten Sie den Query Analyzer und wählen es über **DATEI | ÖFFNEN** aus.

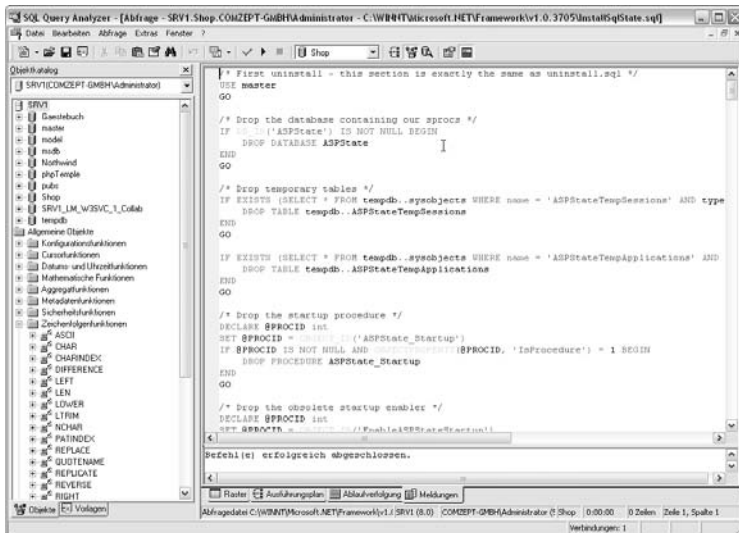


Abbildung 8.15: Das Installations-Skript im Query Analyzer

Drücken Sie nun **F5**. Das Skript wird ausgeführt. Die Installation erfolgt in der neu angelegten *ASPState*-Datenbank, sodass Ihre übrige Datenbankkonfiguration davon völlig unberührt bleibt.

Installation des Skripts

Tragen Sie dazu die in Ihrem System benötigte Verbindungszeichenfolge ein:

```
sqlConnectionString="data source=Name_Des_Servers;
    user id=Datenbank_Nutzer;
    password=Kennwort"
```

Der Datenbankbenutzer *Datenbank_Nutzer* ist meist »sa«. Wenn der SQL Server auf demselben Computer läuft wie der Webserver, tragen Sie für *Name_Des_Servers* »localhost« ein.

Die Ablage der Daten erfolgt in temporären Tabellen in der Datenbank *tempdb*. Dort finden Sie zwei Tabellen: *ASPStateTempSessions* und – für Applikationsvariablen – *ASPStateTempApplications*.

SessionId	Created	Expires	LockDate	LockCookie	Timeout	Locked	SessionItemShort	SessionItemLong
1	220b7cf3gdpdpqg:dhnh4500000001	15.00.2002 02:05:2	15.00.2002 02:33:1	15.00.2002 02:12:1	13	20	0	<linar>

Abbildung 8.16: Eine Sitzungsvariable im SQL Server

8.4.3 Sitzungsvariablen verwenden

Das Speichern der Werte innerhalb einer Sitzung erfolgt in der Kollektion im `Session`-Objekt. Jedes einzelne Element der Kollektion wird allgemein als »Sitzungsvariable« bezeichnet. Tatsächlich handelt es sich aber immer um eine Auflistung. Wird aus der Kollektion ein Element entnommen und daraus eine reguläre Variable erstellt, verliert sie ihren besonderen Status und verhält sich exakt so, wie die ursprünglich als Quelle verwendete. Damit kann der Name bei der Erzeugung geändert werden.



Der Umgang mit dem `Session`-Objekt ist unabhängig von der Wahl des Speicherortes und der Methode der Übertragung der Session-ID. Sie müssen Ihre Programme nicht überarbeiten, wenn Sie die Konfiguration später ändern.

Praktische Anwendung

Sitzungsvariablen erzeugen

Das folgende Programm zeigt, wie eine Sitzungsvariable erzeugt und gespeichert wird:

```
<script language="vb" runat="server">
Public Sub Page_Init(sender As Object, e As EventArgs)
    Session.Add("Farbe", "red")
End Sub
</script>
```

Listing 8.18: Erzeugen und Speichern einer Sitzungsvariablen (Ausschnitt aus `Session1.aspx`)

Beachten Sie hier, dass als Methode zum Aufruf des Codes `Page_Init` eingesetzt wird. Cookies sind Header-Informationen, die vor dem Senden der Seite zusammengestellt werden müssen. Es ist offensichtlicher, diese gleich am Anfang zu setzen – auch wenn die Seite erst am Ende des gesamten Prozesses gesendet wird und Sie die Header bis dahin problemlos verändern können.

Das folgende Programm zeigt, wie auf den gespeicherten Wert wieder zugegriffen werden kann:

```
<%@ Page Language="vb" AutoEventWireup="true" %>
<script language="vb" runat="server">
Public Sub Page_Load(sender As Object, e As EventArgs)
    Dim color As String
    color = Session("Farbe").ToString()
```

```
        currentColor.Text = color
    End Sub
</script>
<html>
  <head><title>VB.NET lernen</title></head>
  <body>
    <h1>Sessions</h1>
    Dieses Skript liest eine Sessionvariable:
    <b><asp:label id="currentcolor" runat="server"/></b>
    <br/>
    Klicken Sie auf den Link, um wieder
    auf die erste Seite zu gelangen.
    <br/>
    <a href="session1.aspx">Zur ersten Seite</a>
  </body>
</html>
```

Listing 8.19: Vollständiges Beispiel zum Abruf der Session-Variablen (Session2.aspx)

Im Gegensatz zum Senden spielt der Zeitpunkt des Abrufs der Informationen beim Empfang keine Rolle. Der Browser sendet immer alle Cookies zu der Domain zurück, von der er sie empfangen hat. Im `Session`-Objekt stehen diese Informationen deshalb immer zur Verfügung. Der eigentliche Speicherort für die Werte steht natürlich auch immer bereit, egal ob es sich um eine Datenbank oder den lokalen Speicher handelt. Beachten Sie, dass Daten immer als Objekt und in serialisierter Form gespeichert werden.

**Sitzungsvariablen
auslesen**

8.5 Cookies

Im Abschnitt über Sitzungen und Applikationen wurden Cookies implizit verwendet – zur Speicherung des Status. Cookies sind aber auch unabhängig davon vielfältig einsetzbar.

8.5.1 Cookies als Informationsspeicher

Cookies dienen der Speicherung von Informationen für einen kurzen Zeitraum. Wie dies funktioniert und warum es Probleme mit Cookies gibt, wird in diesem Abschnitt erläutert. Cookies (dt. Kekse) haben einen völlig irreführenden, verharmlosenden Namen. Aber sie sind bekannt und werden oft als der Angriffspunkt des bösen Hackers aus dem Web verteufelt, der sich an den privaten Dateien der Surfer zu schaffen machen will.

Warum Cookies so unbeliebt sind

Die Ursache für den ganzen im Prinzip unbegründeten Ärger ist ein Artikel von Jon Udell in der Märzausgabe 1997 der Zeitschrift BYTE, einer der größten amerikanischen Computerfachzeitschriften. Dort wurde berichtet, dass Cookies Informationen auf Anforderung des Servers auf der lokalen Festplatte des Nutzers speichern und natürlich auch lesen können. Daraus wurde geschlussfolgert, dass der Server private Daten vom Computer des Surfers lesen kann. Des Weiteren wurde behauptet, dass andere Server wiederum die Daten lesen können, die schon im Cookie gespeichert sind, wodurch das Auslesen privater Kennwörter möglich sein soll. All das hat zur Verdächtigung der Cookies beigetragen und dazu geführt, dass viele Nutzer die Funktion abschalten oder sogar die Cookiedateien regelmäßig löschen. In der Maiausgabe 1997 der BYTE wurde der gesamte Artikel revidiert und richtig gestellt – zu spät, wie sich herausstellte. Die sensationlüsterne Presse hatte wieder etwas Negatives am Web entdeckt und wollte sich das Spielobjekt nicht wegnehmen lassen. Der Pro-Cookie-Artikel wurde totgeschwiegen.

Dennoch gibt es auch berechtigte Einwände gegen Cookies.

Wenn Sie sich die Cookie-Datei aufmerksam ansehen, werden Sie mit hoher Wahrscheinlichkeit ein Cookie entdecken, das einem Server mit dem Namen »doubleclick« oder der deutschen Entsprechung »doppelklick« gehört. Ebenso hoch ist die Wahrscheinlichkeit, dass Sie noch nie auf der Website unter *www.doubleclick.net* waren. Wie kommt nun dieses »fremde« Cookie in den Browser? Ein Blick auf die Seite der Firma DoubleClick offenbart die Motivation. DoubleClick liefert Marktinformatonen über die Nutzer des Internet in Echtzeit. Dazu werden von jedem Nutzer Profile angelegt. Die Erkennung der richtigen Nutzer erfolgt – logischerweise – per Cookie.

Aber wie funktioniert das? Wenn niemand die DoubleClick-Site besucht, können die Cookies nie gesetzt werden – oder doch? DoubleClick-Kunden sind Händler und Anbieter im Internet, die mehr über ihre Kunden wissen möchten. Wenn ein

argloser Surfer die Homepage eines DoubleClick-Kunden besucht, erhält er eine Bannereinblendung, mit der ein Cookie verknüpft ist. Jede andere Seite mit Verbindung zu DoubleClick verfährt ebenso, nur liefert der Browser mit der Banneranforderung brav die alten DoubleClick-Cookies aus – eine perfekte Verfolgung über alle Sites, die im DoubleClick-Netzwerk verbunden sind. Nebenbei liefern die DoubleClick-Kunden auch Daten über die erkannten Nutzer und füllen damit die Profile, im Gegenzug erhalten Sie selbst ständig verbesserte Profile. Das Profil selbst wird bei DoubleClick gespeichert – das Cookie enthält nur die ID-Nummer. Genutzt werden die Profilinformatoren für die Steuerung derselben Werbeanzeigen, mit denen die Cookies geliefert wurden. So werden zwei Surfer, die zur gleichen Zeit die gleiche Seite besuchen, verschiedene Banner sehen – je nach Profil. Das eigentliche Problem ist nicht die Tatsache, dass auf diese Weise Informationen gesammelt werden, sondern dass dies heimlich und ohne Kenntnis der Nutzer erfolgt. Erst beim Anschalten der Cookiewarnung im Browser bemerken viele Nutzer, mit welcher Intensität beim Surfen davon Gebrauch gemacht wird. Diese Nutzung entspricht im Übrigen nicht mehr der ursprünglichen Intention. Die Zwischenschaltung des DoubleClick-Servers lässt die Cookies transparent werden – genau das sollte per Definition aber nie passieren. Natürlich lassen sich Cookies abschalten. Aber man sollte sich durchaus die Mühe machen, im Cookiemanager die Werbecookies zu deaktivieren.

Aus heutiger Sicht muss man sagen, dass tatsächlich eine Reihe von Missbrauchsmöglichkeiten bestehen, die vor allem die Privatsphäre des Nutzers betreffen. Eine Sicherheitslücke, die ein generelles Abschalten rechtfertigen würde, sind Cookies aber definitiv nicht.

Missbrauch

Wie Cookies funktionieren

Cookies wurden von Netscape erfunden und sind seit der ersten Version des Navigators dabei. Später wurde daraus ein Standard, der auch vom World Wide Web Consortium W3.ORG unterstützt wird. Cookies sind eine oder mehrere Dateien, die der Browser anlegt und in denen der sendende Server auf Wunsch Informatio-

Einsatz

nen unterbringen und wieder auslesen kann. Der Sinn von Cookies ist die Wiedererkennung des Nutzers bei einer späteren Session. Cookies lösen also ein gravierendes Problem des HTTP-Protokolls. Cookies können temporär sein, also am Ende einer Sitzung wieder gelöscht werden, andere sind permanent und werden nie oder sehr viel später gelöscht.

Aufbau und Prinzip

Cookies werden zwischen Server und Browser durch HTTP-Header übertragen. Durch Senden eines *Set-Cookie*-Headers wird ein Cookie in der Cookiedatei erzeugt. Soll beispielsweise der Name eines Nutzers gespeichert werden, sieht der zugehörige Header folgendermaßen aus (gesendet wird der Text allerdings in einer Zeile):

```
Set-Cookie: UserName=Roger+Waters; ␣  
           path=/; ␣  
           domain=comzept.de; ␣  
           expires=Tuesday, 01-Jan-2003 00:00:01 GMT
```

Der neue Eintrag in der Cookiedatei wird jetzt erstellt. Die erste Zeile nach dem Header-Namen *Set-Cookie* enthält den Namen des Cookies und dessen Inhalt. Das Cookie wird, wenn nun Domain und Pfadangabe stimmen, in jede Anfrage in die Kopfzeilen eingebaut, die der Browser an den Server stellt. Für jedes Verzeichnis im Webserver können Sie also eigene Cookies erzeugen.

Es folgt eine genaue Erklärung der Parameter eines Cookies:

► NAME=VALUE

Dies ist der Name und Inhalt des Cookies und der einzige Parameter, der nicht optional ist. Erlaubt sind alle Zeichen außer Kommata, Semikola und Leerzeichen. Es ist empfehlenswert, Methoden wie `Server.UrlEncode` zu verwenden, wenn mit dem Inhalt umfangreichere Daten gespeichert werden sollen.

► expires=DATE

Das Verfallsdatum definiert, bis zu welchem Zeitpunkt das Cookie leben darf. Wird das Verfallsdatum überschritten, wird das Cookie gelöscht und nicht mehr an den Server gesendet. Das Datum muss folgendes Format haben:

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

Die Darstellung basiert auf RFC 822, RFC 850, RFC 1036 und RFC 1123. Die einzige legale Zeitzone ist GMT. Beachten Sie die Minuszeichen als Trennzeichen im Datum – dies entspricht

nicht den Möglichkeiten, wie sie in den genannten RFCs dargestellt werden. Der Parameter `expires` ist optional. Wird er nicht angegeben, verfällt das Cookie am Ende der Sitzung.

► `domain=DOMAIN_NAME`

Wenn der Browser in der Liste der gespeicherten Cookies sucht, vergleicht er das Domainattribut *domain* mit dem Internet-Domainnamen des Servers, von dem die URL aufgerufen wurde. Dabei werden auch Teile einer Domain berücksichtigt. Der Domainname »acme.com« wird also auch für vollständige Domains der Art »shipping.crate.acme.com« gültig sein. Nur Server, deren Domainname übereinstimmt, können Cookies setzen und lesen. Außerdem muss der Domainname wenigstens drei Punkte enthalten, ausgenommen davon sind die folgenden generischen Topleveldomains, bei denen zwei Punkte genügen:

- COM
- EDU
- NET
- ORG
- GOV
- MIL
- INT

Wird das Attribut nicht angegeben, wird der Name des Servers (Hostname) verwendet.

► `path=PATH`

Das Pfad-Attribut wird verwendet, um eine Untereinheit der URL innerhalb einer Domain angeben zu können. Wenn der Domainname überprüft und eine Übereinstimmung gefunden wurde, wird nun der Pfad verglichen. Stimmt auch der Pfad überein, wird das Cookie mit der nächsten Anforderung an den Server übertragen. Der Pfad »/foo« wird als gültig angesehen für »/foobar« und »/foo/bar.html«. Der Pfad »/« ist ein genereller Platzhalter. Wird kein Pfad angegeben, wird der Pfad der aufgerufenen Seite angenommen.

► `secure`

Wenn ein Cookie mit dem Attribut `secure` gekennzeichnet ist, wird es nur übertragen, wenn die Verbindung über einen sicheren Kanal zustande kommt (HTTPS, das heißt HTTP über

SSL). Ohne dieses Attribut werden Cookies unverschlüsselt (Klartext) übertragen.

**Syntax der Cookie
HTTP Request
Header**

Wenn eine Ressource auf dem Server via URL angefordert wird, prüft der Browser den URL gegen alle Cookies und fügt der Anforderung an den Server bei Übereinstimmung Variablen-/Wertepaare der Cookies hinzu. An dieser Stelle wird klar: Ein Server fordert keine Cookies und liest keine Cookies, er bekommt sie vom Browser geliefert! Die Wertepaare haben das folgende Format:

```
Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...
```

**Zusätzliche
Hinweise**

Mehrere Cookies werden in einer Anforderung übertragen. Schreiben zwei Server mit demselben Namen und Pfad Cookies, überschreibt der letzte Prozess den vorhergehenden ohne Rückfrage oder Benachrichtigung. Wird ein anderer Name verwendet, wird dieser als weiteres Cookie der Liste hinzugefügt. Wird der Pfad auf eine weitere Verzeichnistiefe erweitert, so wird ein neues Cookie erzeugt. Treffen nun bei der Abfrage mehrere Cookies zu, werden alle zutreffenden Übereinstimmungen gesendet. Das Verfallsdatum teilt dem Client lediglich mit, ab wann es sicher ist, dass das Cookie gelöscht werden kann. Es gibt aber keinen Zwang, dass zu diesem Zeitpunkt tatsächlich die Löschung erfolgt. Ebenso können Cookies vor Ablauf des Verfallsdatums gelöscht werden, wenn der intern dafür bereitgestellte Speicherplatz erschöpft ist. Beim Senden der Cookies werden die weiter übereinstimmenden Pfadangaben vor den weniger übereinstimmenden übertragen.

Cookie-Grenzen

Die maximalen Werte für Cookies, die ein Browser speichert, betragen laut Definition:

- ▶ 300 Cookies insgesamt
- ▶ 4 KByte pro Cookie (Name und Inhalt zusammen)
- ▶ 20 Cookies pro Server und Domain

Wenn diese Grenzen überschritten werden, darf der Client das am längsten ungenutzte Cookie löschen. Wenn das Cookie größer als 4 KByte ist, wird der Inhalt gekürzt, nicht aber der Name.

8.5.2 Cookies praktisch verwenden

Cookies werden in ASP.NET umfassend unterstützt. Da die Aussendung im Header der Serverantwort erfolgt, werden zu sendende Cookies als Kollektion im Objekt `Response` gesammelt. Umgekehrt sind die vom Browser übermittelten Cookies als Kollektion im Objekt `Request` zu finden. Zur Bildung der Cookies selbst wird die Klasse `HttpCookie` verwendet, mehrere Cookies werden in Objekten der Klasse `HttpCookieCollection` gespeichert.

Damit Sie in den nächsten Beispielen den Effekt sofort sehen, ist es empfehlenswert, in den Sicherheitseinstellungen des Internet Explorers, Registerkarte DATENSCHUTZ, für alle Cookies die Option EINGABEAUFORDERUNG zu wählen. Sie werden dann jedes Mal aufgefordert, empfangene Cookies zu bestätigen. In dem dann angezeigten Dialog können Sie auf DETAILS klicken, um zu sehen, was Ihr Programm tatsächlich gesendet hat.



Cookie setzen und überprüfen

Das folgende Programm setzt ein Cookie mit einer Laufzeit von zwei Stunden:

```
<script language="vb" runat="Server">
Public Sub Page_Init(sender As Object, e As EventArgs)
    Dim phonecookie As HttpCookie = New HttpCookie("")
    phonecookie.Name = "Musiker"
    phonecookie.Value = "Roger Waters"
    phonecookie.Expires = DateTime.Now.AddHours(2)
    Response.Cookies.Add(phonecookie)
End Sub</script>
<html>
  <head>
    <title>Response.Cookies</title>
  </head>
  <body>
    Das Cookie wurde gesetzt. Klicken Sie auf den
    <a href="request.cookies.aspx">Link</a>, um es wieder
    auszulesen.
  </body>
</html>
```

Listing 8.20: Erzeugen und Senden eines Cookies (`ResponseCookies.aspx`)

Das Programm nutzt die Methode `Page_Init` zur Ausgabe der Cookies. Cookies werden als HTTP-Header übertragen. Dies würde mit `Page_Load` auch funktionieren, da aber keine Elemente der fertigen Seite benötigt werden, ist die Wahl der Methode etwas direkter. Zuerst wird ein Cookie-Objekt erzeugt:

```
Dim phonecookie As HttpCookie = New HttpCookie( "")
```

Cookie-Daten setzen

Dann werden die Eigenschaften gesetzt, die unbedingt notwendig sind. Der Name wird mit `Name` festgelegt. Alternativ kann die Angabe auch gleich im Konstruktor erfolgen. Die folgende Zeile würde dann die ersten beiden ersetzen:

```
Dim phonecookie As HttpCookie = New HttpCookie("Musiker")
```

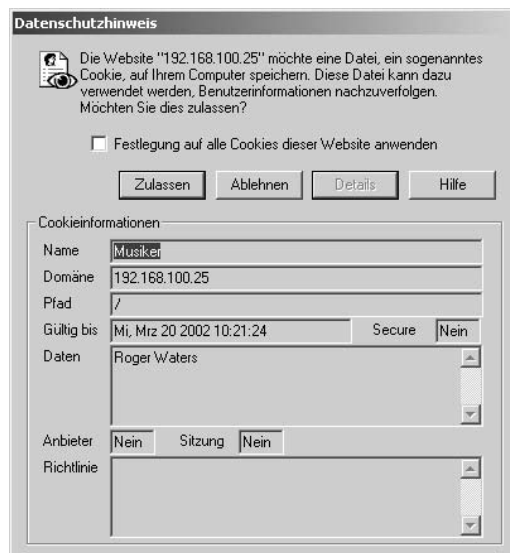


Abbildung 8.17: Anzeige des gesendeten Cookies im Internet Explorer

Nun wird der Wert des Cookies mit `Value` gesetzt. Zuletzt ist noch das Verfallsdatum wichtig, welches mit der Eigenschaft `Expires` festgelegt wird. Die Darstellung der korrekten Datumsform erledigt ASP.NET automatisch. Deshalb können hier einfach Methoden der Klasse `DateTime` verwendet werden. `Now` ermittelt die aktuelle Zeit und `AddHours(2)` rechnet zwei Stunden hinzu. Detaillierte Informationen zu den Datums- und Zeitmethoden finden Sie in Abschnitt 4.9, »Datum und Zeit« ab Seite 284. Wenn Sie im Browser die Anzeige des Cookies forciert haben, sehen Sie folgende Ausgabe in der Abbildung 8.17.

Für die praktische Nutzung ist natürlich das Auslesen auf einer anderen Seite wichtiger. Das folgende Beispiel zeigt dies:

```
<script language="vb" runat="Server">
Public Sub Page_Load(sender As Object, e As EventArgs)
    Dim cv As String = Request.Cookies("Musiker").Value
    cookievalue.Text = cv
End Sub</script>
<html>
    <head>
        <title>Request.Cookies</title>
    </head>
    <body>
        Es wurde folgendes Cookie mit dem 
        Name Musiker erkannt:<br/>
        Inhalt: "<asp:label id="cookievalue" 
        runat="Server"/>"
    </body>
</html>
```

Listing 8.21: Auslesen empfangener Cookies (RequestCookies.aspx)

Der Zugriff auf empfangene Cookies erfolgt über `Request.Cookies`. Auch dies ist wieder eine Kollektion. Es besteht die Möglichkeit, über den Index direkt zuzugreifen, wie dies im Beispiel gemacht wird. Der Wert des Cookies steht in der Eigenschaft `Value` zur Verfügung. Denken Sie daran, dass jedes Cookie wiederum ein Objekt der Klasse `HttpCookie` ist. Die ersten beiden Zeilen der Methode `Page_Load` könnten auch folgendermaßen geschrieben werden:

```
Dim oCookie As HttpCookie = Request.Cookies("Musiker")
cookievalue.Text = oCookie.Value
```

Die Instanziierung eines Objekts lohnt freilich nur, wenn Sie planen, mehr damit zu tun, als nur den Inhalt auszulesen.

**Cookies
empfangen**

Cookie-Kollektionen verwenden

Bislang wurde zwar von der Cookie-Kollektion gesprochen, aber diese nicht tatsächlich verwendet, da nur ein einziges Cookie vorhanden war. Das folgende Beispiel erzeugt eine Kollektion mit Hilfe der Klasse `HttpCookieCollection` und sendet drei Cookies. Das darauf folgende Listing stellt alle Cookies dar, die der Browser gesendet hat.

```
Public Sub Page_Init(sender As Object, e As EventArgs)
    Dim aValues(,) As String = {{"Musiker", "Roger Water"}, _
```

```

        {"Band", "Pink Floyd"}, _
        {"Album", "The Wall"}}

Dim i As Integer
For i = 0 To aValues.GetLength(0) - 1
    Dim sCookName As String = aValues(i,0)
    Dim sCookValue As String = aValues(i,1)
    Dim cTemp As HttpCookie = New HttpCookie(sCookName)
    cTemp.Value = sCookValue
    Response.Cookies.Add(cTemp)
Next
End Sub

```

*Listing 8.22: Senden mehrerer Cookies aus einem Array
(Ausschnitt aus ResponseCookiesColl.aspx)*

Wie es funktioniert

In diesem Programm werden Daten aus einem zweidimensionalen Array entnommen und jeweils einem Cookie-Objekt zugewiesen:

```
cTemp.Value = sCookValue
```

Dieses wird dann an die Cookie-Kollektion des Response-Objekts angehängt:

```
Response.Cookies.Add(cTemp)
```

Haben Sie die Eingabeaufforderung im Browser wieder eingeschaltet, werden die drei Cookies nacheinander angezeigt.

```

<script language="vb" runat="server">
Public Sub Page_Load(sender As Object, e As EventArgs)
    Dim MyCookieColl As HttpCookieCollection = Request.Cookies
    Dim MyCookie As HttpCookie
    Dim CookieNames() As String = MyCookieColl.AllKeys
    Dim i As Integer
    For i = 0 To CookieNames.Length - 1
        MyCookie = MyCookieColl(CookieNames(i))
        WriteCookies.Text = "<b>Cookie:</b> " & 
            + MyCookie.Name + "<br>"
        WriteCookies.Text += "<b>Wert:</b> " & 
            + MyCookie.Value + "<br>"
    Next
End Sub
</script>
<html>
<head>
    <title>RequestCookiesColl</title>
</head>
<body MS_POSITIONING="GridLayout">
    <asp:Label Runat="server" ID="WriteCookies"/>

```

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

Listing 8.23: Auslesen aller empfangenen Cookies (RequestCookiesColl.aspx)

Zum Auslesen wird eine Instanz der Klasse `HttpCookieCollection` direkt benutzt.

Wie es funktioniert

```
Dim MyCookieColl As HttpCookieCollection = Request.Cookies
```

Diese wird dann mit einer Schleife durchlaufen, zuvor werden jedoch die Namen extrahiert. In der Kollektion bilden die Namen die Schlüssel, deshalb kommt die Eigenschaft `AllKeys` zum Einsatz:

```
Dim CookieNames() As String = MyCookieColl.AllKeys
```

In der Schleife kann dann der Zugriff über die so erhaltene Liste der Schlüssel (= Cookienamen) erfolgen:

```
MyCookie = MyCookieColl(CookieNames(i))
```

Die Ausgabe nutzt dann die Eigenschaften `Name` und `Value` des Cookie-Objekts.

Tipps zur Fehlersuche

Wenn die Cookies nicht wie erwartet funktionieren oder die Daten nicht korrekt sind, hilft das interne Tracing weiter. Diese Funktion zeichnet alle Zustände der aktuellen Anforderung auf und zeigt sie an. Um das Tracing für eine einzelne Seite einzuschalten, ergänzen Sie die `Page-Direktive` um das `Trace-Attribut`:

```
<%@ Page Language="vb" AutoEventWireup="true" trace="true"%>
```

Beim Ausführen sollten Sie dann etwa folgende Ausgabe erhalten:

Cookieauflistung		
Name	Wert	Größe
Musiker	Roger Water	19
ASP.NET_SessionId	ifrfrkeuglcfshw45oyrsuiiy	42
Band	Pink Floyd	15
Album	The Wall	14
Musiker	Roger Water	19
Band	Pink Floyd	15
Album	The Wall	14
Headerauflistung		
Name	Wert	
Connection	Keep-Alive	
Accept	*/.*	
Accept-Encoding	gzip, deflate	
Accept-Language	de	
Cookie	Musiker=Roger Water; ASP.NET_SessionId=ifrfrkeuglcfshw45oyrsuiiy; Band=Pink Floyd; Album=The Wall	
Host	localhost	
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)	

Abbildung 8.18: Überwachung von Programmfunktionen: Header im Allgemeinen (unten) und Cookies im Besonderen (oben)

8.6 Applikationsmanagement

Das Applikationsmanagement kümmert sich um Abläufe, Prozesse und Variablen, die sich auf die gesamte Anwendung auswirken, also unabhängig von Benutzern sind.

8.6.1 Einführung in das Applikationsereignismodell

Sitzungen sind immer benutzerorientiert. Dies ist nicht hilfreich, wenn Daten zwischen Benutzern ausgetauscht werden sollen. Es gibt deshalb auch eine Sicht auf die in ASP.NET ablaufenden Prozesse aus dem Blickwinkel der gesamten Applikation. Das Applikationsmanagement in ASP.NET hat zwei Aspekte:

1. Verwaltung des Ablaufes der Ereignisse des gesamten Anforderungs- und Antwortprozesses.
2. Speicherung von Daten außerhalb des Anforderungs- und Antwortprozesses in einem für alle Prozesse gemeinsamen Speicherbereich.

Grundlagen des Ereignismodells

Applikation Eine Applikation umfasst alle Dateien, Seiten, Module und Programme im Sichtbereich eines virtuellen Verzeichnisses auf einem einzelnen Webserver. Der Datenaustausch kann also nur innerhalb dieses Kontextes stattfinden. In einer solchermaßen gebildeten Applikation gibt es eine ganze Reihe spezieller Ereignisse, auf die Sie reagieren können, aber nicht müssen. Zehn Ereignisse werden immer ausgelöst, wenn eine *aspx*-Seite angefordert wird.

Standardereignisse Die Behandlung dieser und der nachfolgend beschriebenen bedingten Ereignisse finden in der Datei *global.asax* statt. Der Start der Ereignisse bei der Verarbeitung der Seite (*Page_Init*, *Page_Load* usw.) geschieht innerhalb des Prozessschrittes »Ausführung der Seite« in Abbildung 8.19. Die einzelnen Ereignisse haben folgende Bedeutung:

► **BeginRequest**

Dieses Ereignis wird bei jeder Anforderung zuerst ausgelöst. Hier können Aktionen ausgeführt werden, die unabhängig von irgendeinem anderen Teil des Programms benötigt werden.

► AuthenticateRequest, AuthorizeRequest

Diese Ereignisse dienen der Behandlung der Authentifizierung (Erkennung) bzw. Autorisierung (Rechtevergabe).

► ResolveRequestCache

Wenn der interne Zwischenspeicher verwendet wird, liefert ASP.NET Seiten aus dem Cache aus, ohne erneut den Code auszuführen. Dieses Ereignis wird unmittelbar vor der Prüfung der Gültigkeit des Zwischenspeichers aufgerufen. Es erlaubt damit die Ausführung von Code unabhängig vom Zustand des Zwischenspeichers oder seiner Verwendung.

► AcquireRequestState

An dieser Stelle werden die Sitzungsdaten rekonstruiert. Wenn eine eigene Sitzungsverwaltung implementiert werden soll, sollte der Code hier ausgeführt werden.

► PreRequestHandlerExecute

Dies ist das letzte Ereignis vor der Ausführung der Anforderung. Diese führt unter normalen Umständen und bei einfachen Seitenabrufen zu einem Page-Request.

► PostRequestHandlerExecute

Dies ist das erste Ereignis nach der Ausführung der Anforderung.

► ReleaseRequestState

In diesem Schritt werden die Sitzungsdaten aktualisiert. Änderungen daran sind nun nicht mehr möglich.

► UpdateRequestCache

Falls der Zwischenspeicher aktualisiert werden soll und dazu spezifische Aktionen notwendig sind, kann dieses Ereignis verwendet werden.

► EndRequest

Dies ist das letzte Ereignis, bevor die Daten an den IIS zur Auslieferung übergeben werden. Danach werden Kopf und Körper der Antwort gesendet.

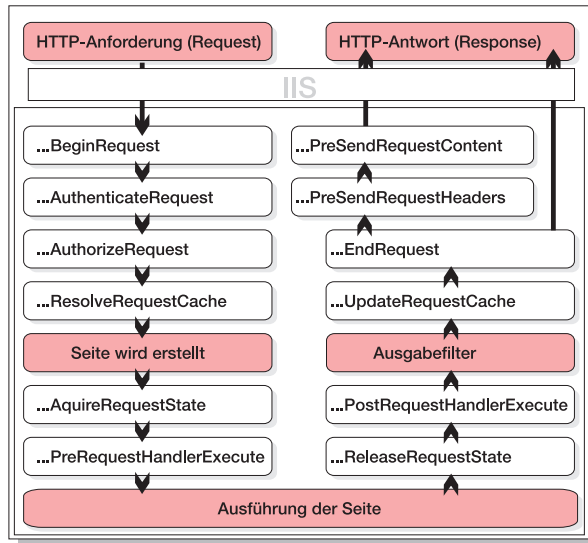


Abbildung 8.19: Ablauf des Aufrufes der zehn Standardereignisse einer Applikation

Ungepufferte Ausgaben behandeln

Bei der Beantwortung einer Anfrage gibt es zwei Strategien, die angeforderten Daten zu übertragen. Entweder wird der gesamte Inhalt gesammelt und anschließend komplett übertragen. Diese Strategie verwendet einen Ausgabepuffer, um die Daten zu sammeln. Dann gilt der Ablauf, den die bereits beschriebenen Ereignisse behandeln. Wird dagegen die Seite ungepuffert aufgebaut, werden immer wieder kleinere Fragmente an den IIS zum Senden übergeben. Während die linke Seite in Abbildung 8.19 unverändert und einmalig zur Ausführung gelangt, funktioniert das mit der Ausgabe nicht mehr. Deshalb gibt es zwei spezielle Ereignisse:

► PreSendRequestHeaders

Dieses Ereignis wird ausgelöst, bevor die Kopfzeilen gesendet werden.

► PreSendRequestContent

Auch wenn mehrere Fragmente einer Antwort übertragen werden, gibt es nur einen Zeitpunkt, an dem die Kopfzeilen vollständig sind. Dieses Ereignis löst aus, wenn die Header vollständig vorliegen und die Übertragung der Inhalte beginnt.

Am Ende der Übertragung wurden die anderen Antwort-Ereignisse dennoch ausgelöst.

Neben den unbedingten Ereignissen gibt es auch bedingte. Diese treten nur auf, wenn die folgenden Bedingungen innerhalb einer Applikation auftreten:

Bedingte Ereignisse

► Start

Dieses Ereignis tritt auf, wenn die Applikation das erste Mal aufgerufen wird. Dies passiert logischerweise nur ein einziges Mal.

► End

Auch dieses Ereignis ist einmalig. Es wird beim Stoppen der Applikation aktiv.

► Error

Wenn Fehler auftreten, können Sie diese mit diesem Ereignis behandeln. Es ist mächtiger als `Page_Error`, da alle Laufzeitfehler einer Applikation abgefangen werden können.

► Disposed

Dieses Ereignis tritt auf, wenn eine Applikation vollständig entfernt wird, beispielsweise über die Managementkonsole des IIS.

Alle Applikationsereignisse werden in der Datei *global.asax* (*Applikationsereignisse*) verarbeitet.

Die Datenspeicherung: Applikationsvariablen

Ein typischer Weg, große Datenmengen allen Benutzern bereit zu stellen, sind Datenbanken. Kleinere Datenmengen, die sich während der Laufzeit häufig ändern und deren Inhalt von den aktuellen Aktionen der Benutzer abhängt, sind darin nicht immer optimal aufgehoben. ASP.NET kennt dafür applikationsweite Variablen. Die Verwaltung erfolgt im Rahmen des Applikationsmanagements. Ebenso wie für die Sitzungen gibt es ein spezielles Objekt dafür: `Application`. Instanzen werden ähnlich dem `Session`-Objekt automatisch durch die ASP.NET-Komponente gebildet.

Ebenso wie bei den Sitzungen können auch Applikationen Variablen speichern. Diese stehen dann allen Benutzern zur Verfügung. Von den Sitzungsvariablen unterscheiden sich Applikationsvariablen durch drei Eigenschaften:

Applikationsvariablen

1. Applikationsvariablen basieren nicht auf Cookies.
2. Der Webserver muss keine Session mitführen, um mit der Applikationsvariablen zu arbeiten.
3. Die Verwendung ist risikolos und unabhängig vom Browser oder seinen Einstellungen.

Das Einsatzspektrum der Variablen ist weit gefächert. Es gibt viele Anwendungen, die davon profitieren. Wann immer Sie wieder auf dieselben veränderlichen Informationen zugreifen möchten, wie beispielsweise einen Nachrichtendienst oder Tipp des Tages, sind Applikationsvariablen einsetzbar. Speichern Sie den Wert in der Applikationsvariablen und er steht allen Nutzern überall zur Verfügung. Andere typische Einsatzfälle sind Bannersteuerungen oder Besuchszähler, so genannte Counter.

Das folgende Programm zeigt, wie eine Applikationsvariable erzeugt und mit einem Wert belegt wird:

```
<%@ Page Language="vb" AutoEventWireup="true" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<script language="vb" runat="server">
Public Sub Page_Load(sender As Object, e As EventArgs)
    Application("Version") = "0.9.3"
End Sub
</script>
<html>
    <head>
        <title>ApplicationSetvar</title>
    </head>
    <body MS_POSITIONING="GridLayout">
        Auf dieser Seite wird eine Applikationsvariable
        erzeugt und gesetzt. <br />
        Auf der <a href="ApplicationGetvar.aspx">
        folgenden Seite</a> wird sie wieder ausgelesen.
    </body>
</html>
```

Listing 8.24: Setzen einer Applikationsvariablen (ApplicationSetvar.aspx)

Auf einer anderen Seite kann dann auf diese Variable zugegriffen werden:

```
<%@ Page Language="vb" AutoEventWireup="true" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<script language="vb" runat="server">
Public Sub Page_Load(sender As Object, e As EventArgs)
    Dim version As String = Application("Version").ToString()
    ver.Text = version
End Sub
</script>
<html>
    <head>
        <title>ApplicationGetvar</title>
    </head>
    <body MS_POSITIONING="GridLayout">
        Auf dieser Seite wird eine Applikationsvariable
        erzeugt und gesetzt. <br />
        Auf der <a href="ApplicationSetvar.aspx">
        folgenden Seite</a> wird sie wieder ausgelesen.
    </body>
</html>
```

```
End Sub
</script>
<HTML>
  <HEAD>
    <title>ApplicationGetvar</title>
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    Auf dieser Seite wird die Applikationsvariable ausgelesen:
    <br>
    <asp:Label Runat="server" ID="ver" />
  </body>
</HTML>
```

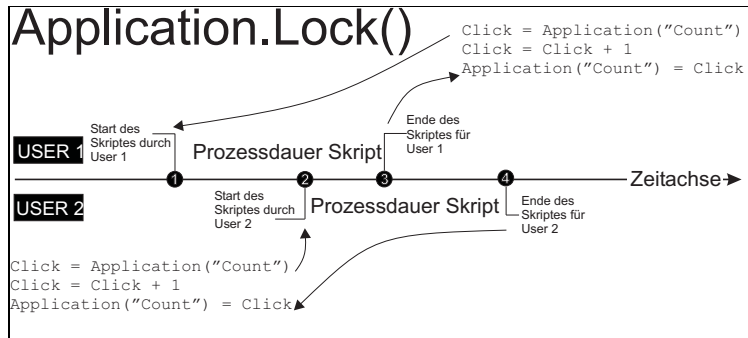
Listing 8.25: Nutzung einer Applikationsvariablen (ApplicationGetvar.aspx)

Intern werden Applikations-Variablen als Objekte gespeichert, deshalb ist die Anwendung von `ToString` angebracht, um wieder den richtigen Datentyp zu erhalten. Wie schon bei den Sitzungen gezeigt, ist das `Application`-Objekt direkt instanziiert und muss nicht abgeleitet werden. Die Variablen bilden eine Aufzählung, auf die direkt über einen Indexer zugegriffen werden kann. Daher ist die Kurzschreibweise `Application("Name")` zulässig.

Wie es funktioniert

Eine gute Anwendung ist ein Hitzähler für Ihre Webseite. Damit der Zähler auch exakt arbeitet, ist es sinnvoll, über die interne Arbeitsweise nachzudenken. Der Webserver liefert Seiten an Nutzer, wann immer diese die Seiten anfordern. So entstehen parallel laufende Prozesse. Da alle Prozesse Zeit brauchen, um ausgeführt zu werden, ergibt sich möglicherweise ein Problem. Wenn zwei Nutzer gleichzeitig eine Seite aufrufen, werden die Werte parallel verarbeitet. Wenn der Ursprungswert der Variablen 4 ist, schreibt Nutzer 1 mit seiner Sitzung den Wert 5 zurück. Bis dahin hat aber auch Nutzer 2 die Seite gestartet, ebenfalls den Wert 4 ermittelt und 5 zurückgeschrieben. Danach steht der Zähler auf 5 und nicht, wie es richtig wäre, auf 6. Das `Application`-Objekt kennt deshalb zwei besondere Methoden, die dazu gedacht sind, andere Prozesse vorübergehend zu stoppen – `Lock` und `UnLock`. Diese statischen Methoden stammen aus der Klasse `HttpApplicationState` stehen unmittelbar zur Verfügung. Die folgende Abbildung zeigt, wie die Methoden eingesetzt werden können, um den Zähler korrekt ablaufen zu lassen.

Daneben sind auch Methoden und Eigenschaften zum Zugriff auf die Variablen-Kollektion vorhanden, die sich nicht von anderen Kollektionen unterscheiden.

Abbildung 8.20: Ablauf von Seitenabrufen und Einsatz von *Application.Lock*

Eine praktische Anwendung finden Sie im folgenden Abschnitt.

8.6.2 Die Datei *global.asax*

Der Umgang mit *global.asax* ist recht einfach. Alle in den letzten beiden Abschnitten angesprochenen Ereignisse werden an Methoden geleitet, die Sie hier definieren können. Die Methoden sind – wie alle Ereignisbehandlungsmethoden – Sub-Methoden, die nichts zurückgeben. Applikations-Ereignisse tragen das Präfix *Application_*, Sitzungsereignisse dagegen *Session_*². Wenn Sie also eine Behandlung der Ereignisse wünschen, dann muss lediglich die passende Methode dazu definiert werden.

Die Datei *global.asax* liegt im Stammverzeichnis der Applikation. Dies ist bei virtuellen Verzeichnissen das Startverzeichnis des Verweises, so wie es im IIS eingerichtet wurde. Allein das Ablegen der Datei an der richtigen Stelle reicht aus, um auf Applikations- und Sitzungsereignisse reagieren zu können. Das folgende Beispiel zeigt eine Datei *global.asax*, die gleich mehrere Aufgaben erfüllt:

```
Sub Application_Start(ByVal sender As Object,   
                    ByVal e As EventArgs)   
    Application("HitCounter") = 0   
    Application("Version") = "0.9.3"   
End Sub
```

```
Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)   
    Application.Lock()
```

² Die Namen sind anders als in C#, was vor allem bei der Übernahme von fremdem Code Schwierigkeiten bereitet. In VB.NET fehlt das Präfix-Teil »On« (Session_Start statt Session_OnStart).

```
Application("HitCounter") = ←  
    Convert.ToInt32(Application("HitCounter")) + 1  
    Application.UnLock()  
End Sub
```

Listing 8.26: Ereignisverarbeitung in global.asax (Ausschnitt)

Verwendet wird hier das Ereignis `Application_Start`. Beim Start der Applikation wird die entsprechende Methode ausgeführt. Die Applikations-Variablen `HitCounter` und `Version` werden erzeugt und gesetzt. Mit dem Start jeder neuen Sitzung (Ereignis `Session_Start`) wird die Applikation verriegelt, der Zähler `HitCounter` erhöht und die Applikation wieder freigegeben. Der eigentliche Zähler wird dann auf der Seite folgendermaßen ausgegeben (siehe dazu Testprogramm `HitCounter.aspx`):

Wie es funktioniert

```
Ausgabe: <% Response.Write(Application("HitCounter")) %>
```

Listing 8.27: Ausschnitt aus `HitCounter.aspx` (funktioniert nur mit der passenden `global.asax/global.asax.vb`)

8.7 Optimierung des Datenverkehrs

Webserver bedienen sehr viele Anforderungen, weitaus mehr, als klassische Dateiserver im lokalen Netzwerk. Schon sehr früh wurden deshalb Optimierungsstrategien entworfen, die den Datenverkehr in Weitverkehrsnetzen verbessern und den Zugriff auf Daten beschleunigen. Zu den wichtigsten Eigenschaften gehört das Zwischenspeichern von Dokumenten für einen schnelleren Abruf, das so genannte Caching.

In ASP.NET gibt es mehrere Optimierungsverfahren. Im alten ASP wurden oft Applikationsvariablen verwendet, um häufig benötigte, zentrale Daten aufzunehmen. Diese Werte werden immer im Speicher gehalten und stehen deshalb schneller zur Verfügung, als wenn ein Dateisystemzugriff erfolgen muss. Dieses Prinzip wurde in ASP.NET bis zu einem eigenen Modul ausgebaut.

8.7.1 Caching von Seiten und Steuerelementen

Caching ist eine Programmier Technik, bei der häufig angeforderte Daten in einem schnellen Zwischenspeicher gehalten werden, um so langsame Zugriffe auf das Dateisystem, auf Datenbanken oder

Ansprüche

andere Quellen zu sparen. Aus Sicht des Browsers erscheinen diese Seiten dann wie der Abruf von statischen Informationen. Das ist meist ausreichend, wenn man an die Startseite einer Site denkt. Dort wird oft lediglich zum Tagesende das Datum gewechselt und eventuell noch der Nachrichtenbereich. Eine Impressum-Seite dürfte sich dagegen nur alle paar Monate – wenn überhaupt – ändern. Eine Nachrichtenagentur, die stündlich Neuigkeiten herausgibt, wird andererseits darauf bedacht sein, immer frische Inhalte zu liefern und eine Zwischenspeicherung unterdrücken. Es gibt also ganz unterschiedliche Ansprüche an die Art und Weise des Caching.

In ASP.NET sind sehr feine Einstellungen des Caches möglich:

- ▶ Speicherung von Objekten. Dies wird in Abschnitt 8.7.2, »Speicherung von Daten während der Laufzeit« ab Seite 816 erläutert.
- ▶ Speicherung ganzer Seiten im Ausgabe-Cache
- ▶ Speicherung von Teilen einer Seite, sodass statische Inhalte zwischengespeichert und dynamische weiterhin neu erzeugt werden
- ▶ Festlegung des Speicherortes auf dem Übertragungsweg:
 - ▶ Nutzung des Zwischenspeichers im Browser
 - ▶ Nutzung eines Proxy-Servers, der HTTP 1.1 versteht
 - ▶ Nutzung des Webserver-Caches, den ASP.NET selbst verwaltet

Fremde Speicher steuern

Bevor Sie sich jetzt wundern, wie ASP.NET Zugriff auf die Zwischenspeicher des Browsers oder eines Proxy-Server erlangt: Cache-Methoden sind im Protokoll HTTP festgelegt und können durch entsprechende Kopfzeilen gesteuert werden. Nicht mehr passiert bei der Wahl eines außerhalb des Webserver liegenden Speichers. Die diesen Ort verwaltende Instanz wird mit den Kopfzeilen aufgefordert, für die Speicherung in der gewünschten Art und Weise zu sorgen.

Speicherung ganzer Seiten im Cache

Die Speicherung von Seiten im Cache ist nicht trivial. Einfache Lösungen verwenden den Dateinamen der zu speichernden Seite. Wenn der Name nun durch Anhängen verschiedener Parameter scheinbar wechselt, wird die Seite dennoch bei jedem Aufruf im

Cache aktualisiert, was im Endeffekt mehr Zeit kostet als ohne Caching. Der Fall, dass nur statische Seiten ohne Parameter Verwendung finden, ist dagegen relativ selten. Das Caching in ASP.NET ist sehr ausgereift und bietet hier einige interessante Lösungen an.

Zuerst soll jedoch ein einfaches Beispiel den Effekt demonstrieren. Das folgende kleine Programm gibt die aktuelle Zeit und eine passende Begrüßung aus. Rufen Sie die Seite mehrfach ab. Sie werden sehen, dass sich der Inhalt nur alle 30 Sekunden verändert. Diese Zeit wurde nämlich als Speicherzeit für den Cache gewählt.

<%@OutputCache>

```
<%@ Page language="vb" Codebehind="CachePage.aspx.vb" ␣
    AutoEventWireup="false" ␣
    Inherits="Addison.VBNet.WebForm.CachePage" %>
<%@ OutputCache Duration="30" VaryByParam="none" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<html>
  <head>
    <title>CachePage</title>
  </head>
  <body MS_POSITIONING="GridLayout">
    <h1>Cache testen</h1>
    <h2>Ganzseiten-Cache</h2>
    <asp:Label Runat="server" ID="DatumZeit"/>
    <br/>
    <asp:HyperLink Runat="server" ID="link" />
  </body>
</html>
```

Listing 8.28: Seite, die 30 Sekunden im Cache verbleibt (CachePage.aspx)

Die Code-Datei enthält nur Befehle, die der Demonstration dienen, aber keinen Einfluss auf das Cache-Verhalten haben:

```
Public Class CachePage
  Inherits Page
  Protected DatumZeit As Label
  Protected link As HyperLink

  Private Sub Page_Load(ByVal sender As Object, ␣
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    link.NavigateUrl = Request.Path
```

```
link.Text = "Erneut abrufen"  
DatumZeit.Text = System.DateTime.Now.ToLongTimeString()  
End Sub  
  
End Class
```

Listing 8.29: Code-Datei zu CachePage.aspx (CachePage.aspx.vb)

Die Steuerung erfolgt hier über die Direktive `@OutputCache`. Die Tabelle 10.2 in Abschnitt 10.2.5, »Die Direktive `@OutputCache`« ab Seite 960 zeigt alle Optionen auf einen Blick. Die nötigen Einstellungen werden hier diskutiert.

- Duration** Die wichtigste Angabe beim Caching ist sicher die Zeit, die die Seite im Cache verbleiben soll. Diese Angabe erfolgt mit dem Attribut `Duration` in Sekunden. Dies führt erstmal zur Wahl aller Speicherwege. Die Seite wird also im Webserver gespeichert und die HTTP-Kopfzeilen werden gesendet, um Proxys und Browser zur Mitarbeit zu bewegen.
- VaryByParam** Einzige Pflichtangabe außer der Zeit ist das Attribut `VaryByParam`. Steht hier der Wert »none«, werden die per GET oder POST gesendeten Daten nicht beachtet und trotz Änderungen daran bleibt die Seite im Cache. Ihre auf `PostBack` oder Steuerelementen mit Anzeigestatus basierenden Applikationen werden dann möglicherweise nicht mehr richtig funktionieren. Das Attribut verhindert aber nicht, dass POST-Daten übertragen werden. Wenn Sie nur Formulardaten annehmen, die Anzeige des Formulars aber nicht ändern möchten, kann der Einsatz sinnvoll sein. Ein anderer Parameter ist »*« – hiermit werden alle GET- und POST-Daten beachtet und nach jeder Änderungen wird die Seite im Cache erneuert. Dies ist die sicherste Methode, allerdings ist der Speicherefolg nicht sehr hoch. Die dritte Methode ist die gezielte Festlegung eines Parameters, dessen Änderungsverhalten überwacht werden soll. Schreiben Sie `VaryByParam="Selection"` und Ihre Seiten nutzen einen GET-Parameter mit diesem Namen, wird bei einer Änderung daran eine neue Instanz in den Cache gelegt. Die beiden folgenden Aufrufe werden dann unterschieden:

`CachePage.aspx?Selection=Home`

`CachePage.aspx?Selection=Impressum`

Das dritte wichtige Attribut ist `Location`. Damit wird der Speicherort gesteuert. Die Angabe ist optional. Ohne wird der Parameter »Any« angenommen, es werden also alle verfügbaren Speicherorte genutzt. Zulässig sind folgende Parameter:

Location

► Any

Jeder verfügbare Speicher wird verwendet, also ein Browser, Proxy und Server.

► Client

Der Speicher des Browsers wird verwendet. Das geschieht unabhängig davon, ob der Browser diese Funktion unterstützt, denn ASP.NET wird nur die dafür notwendigen Kopfzeilen senden.

► DownStream

Der Speicher des Browsers oder ein HTTP 1.1 fähiger Proxy wird verwendet. Auch hier kann nicht geprüft werden, ob auf dem Übertragungsweg solche Proxys existieren.

► None

Die Zwischenspeicherung ist deaktiviert.

► Server

Der Ausgabespeicher befindet sich auf dem Webserver.

Ein weiteres Attribut heißt `VaryByControl` und dient der Kontrolle der Ablage einzelner Benutzer-Steuer-element (*ascx*-Dateien). Es wird im nächsten Abschnitt beschrieben. Darüber hinaus kann mit `VaryByCustom` der Typ des Clients überwacht werden. Möglicherweise erstellen Sie von ein und derselben Seite mehrere Versionen für den Internet Explorer, Netscape und Opera. Dann können Sie diese Seiten getrennt im Cache halten, wenn Sie `VaryByCustom="browser"` benutzen. Andere Werte als dieser bedürfen einer Modifikation der Verwaltung.

VaryByControl

Speicherung von statischen Teilen einer Seite im Cache

Eigentlich lassen sich Teile einer Seite nicht speichern. Stattdessen können Sie die `@OutputCache`-Direktive auch auf Benutzer-Steuer-elemente anwenden. Informationen darüber, wie Benutzer-Steuer-elemente erzeugt und verwendet werden, finden Sie im Abschnitt 7.2, »Modularer Code mit Benutzer-Steuer-elementen« ab Seite 647. Praktisch genügt es, die Speicherbedingungen in der Direk-

tive festzulegen. Lediglich das Attribut `Location` ist nicht verfügbar. Da einzelne Steuerelemente keine HTTP-Kopfzeilen senden können, wäre eine Modifikation der Seitenspeicherstrategie unmöglich.

PartialCaching Wenn Sie benutzerdefinierte Steuerelemente entwickeln, die ausschließlich aus Code bestehen, wie es im Abschnitt 7.6, »Kundenspezifische Steuerelemente (Custom Controls)« ab Seite 706 für die kundenspezifischen Steuerelemente gezeigt wird, können Sie das Systemattribut `<PartialCaching(30)>` verwenden. Die »30« steht für die Anzahl der Sekunden, die das Steuerelement gespeichert wird.

Programmgesteuerte Beeinflussung des Cache

Neben der Methode, Direktiven zu verwenden, kann der Cache auch über Code kontrolliert werden. Das folgende Listing ersetzt den Code aus Listing 8.29, nicht jedoch die zugrunde liegende *aspx*-Datei – abgesehen davon, dass die `@OutputCache`-Direktive nun entfällt. Die Aufgabe übernimmt der Code in der `Page_Init`-Methode:

```
Public Class CachePageCode
    Inherits Page
    Protected DatumZeit As Label
    Protected link As HyperLink

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        link.NavigateUrl = Request.Path
        link.Text = "Erneut abrufen"
        DatumZeit.Text = System.DateTime.Now.ToLongTimeString()
    End Sub

    Private Sub Page_Init(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Init
        Response.Cache.SetExpires(DateTime.Now.AddSeconds(30))
        Response.Cache.SetCacheability(HttpCacheability.Public)
    End Sub
End Class
```

Listing 8.30: Steuerung des Cache im Code (*CachePageCode.aspx.vb*)

Der Zugriff auf den Cache erfolgt über das `Response`-Objekt und dessen Eigenschaft `Cache`. Diese Eigenschaft gibt ein `HttpCachePolicy`-Objekt zurück. Der Zugriff darauf ist beispielsweise folgendermaßen möglich:

```
Dim c As HttpCachePolicy = Response.Cache
```

Zwei Eigenschaften korrespondieren mit den gleichnamigen Attributen der Direktive `@OutputCache`:

Eigenschaften

► `VaryByHeaders`

Eine Kollektion der HTTP-Header (Kopfzeilen), die zur Unterscheidung von Seiten im Cache herangezogen werden.

► `VaryByParams`

Eine Kollektion der Parameter, die zur Unterscheidung von Seiten im Cache geprüft werden.

Des Weiteren sind eine Vielzahl Methoden verfügbar, von denen hier die wichtigsten vorgestellt werden:

Methoden

► `SetExpired`

Setzt das absolute Ablaufdatum der Seite im Cache. Benutzen Sie die Add-Methoden der Datumsstrukturen, um das Zieldatum zu berechnen.

► `SetCacheability`

Legt die Art des Speichers fest. Dies ist ein Wert der folgenden Aufzählungen:

► `HttpCacheability.NoCache`

Keine Speicherung auf Clientseite

► `HttpCacheability.Public`

Alle Speicher werden verwendet

► `HttpCacheability.Private`

Nur Browser, nicht jedoch auf Proxy-Servern

► `HttpCacheability.Server`

Nur auf dem Server

► `SetLastModified`

Legt das Datum für die `Last-Modified`-Kopfzeile fest. Damit wird dem Client mitgeteilt, wann sich das Dokument das letzte Mal geändert hat, falls dieser eine weitere eigene Speicherstrategie verwendet.

Weitere Methoden für exotische Header

Einige weitere Methoden setzen gezielt HTTP-Kopfzeilen, die jedoch nur selten Verwendung finden. Nachteil all dieser Methoden ist die Abhängigkeit von den Möglichkeiten eines Ihnen praktisch unbekannten Clients. Insofern kommt eine umfassende Nutzung nur in geschlossenen Systemen in Betracht.

8.7.2 Speicherung von Daten während der Laufzeit

In vollständig dynamischen Systemen sind die Möglichkeiten des Zwischenspeichers auch dann nicht immer gegeben, wenn eine so ausgereifte Technik wie in ASP.NET zur Verfügung steht. Deshalb können Sie in ASP.NET generell Objekte in einem internen Cache ablegen. Dies erfolgt mit der Klasse `Cache`.

Dabei wird der Speicher innerhalb von ASP.NET – also In-Process – verwendet. Stürzt der Worker Process ab oder wird er recycelt, gehen die Daten in diesem Cache verloren. Der Einsatz ähnelt dem der Applikationsvariablen, dort werden die Daten jedoch im IIS-Prozess gehalten und sind damit »recyclingresistent«. Natürlich hat die Klasse `Cache` auch Vorteile. Sie können für alle Objekte, ähnlich wie beim Seiten-Cache, Ablaufdaten festlegen. Da das Objekt gelegentlich verschwindet³, müssen Sie vor der Verwendung gespeicherter Objekte prüfen, ob diese noch verfügbar sind und verloren gegangene wieder erzeugen. Ein Beispiel zeigt, wie die Nutzung praktisch aussieht:

```
Imports System.Web.Caching

Public Class CacheObjects
    Inherits System.Web.UI.Page

    Protected DataGridView As DataGrid
    Protected DatumZeit As Label
    Protected link As HyperLink

    Private City As ArrayList = Nothing
    Private onRemove As CacheItemRemovedCallback = Nothing

    Private Sub GetData()
        Dim al As ArrayList = New ArrayList()
        al.Add("Berlin")
        al.Add("Paris")
    End Sub
End Class
```

³ Recycling-Vorgänge finden automatisch statt, vergleichen Sie dazu die Einstellungsmöglichkeiten in der *machine.config*, beschrieben in Abschnitt 10.4.2, »Den ASP.NET-Worker Process konfigurieren« ab Seite 969.

```

        al.Add("Madrid")
        al.Add("Londond")
        al.Add("Wien")
        al.Add("Oslo")
        al.Add("Stockholm")
        Dim cd As CacheDependency
        = New CacheDependency(Server.MapPath(Request.Path))
        onRemove
        = New CacheItemRemovedCallback(AddressOf
        RemovedCallback)

        Cache.Add("CityCache", al, cd, _
            DateTime.Now.AddSeconds(5), _
            TimeSpan.Zero, _
            CacheItemPriority.Normal, onRemove)
        Cache("CityCache") = al
    End Sub

    Public Sub RemovedCallback(ByVal k As String,
        ByVal v As Object,
        ByVal r As CacheItemRemovedReason)

        GetData()
        City = CType(Cache("CityCache"), ArrayList)
        City.Add(r.ToString() + " " + k)
        Cache("CityCache") = City
        DatenGrid.DataBind()
    End Sub

    Public Sub Page_Load(ByVal sender As Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        City = CType(Cache("CityCache"), ArrayList)
        If City Is Nothing Then
            GetData()
            City = CType(Cache("CityCache"), ArrayList)
        End If
        DatenGrid.DataSource = City
        DatenGrid.DataBind()
        link.NavigateUrl = Request.Path
        link.Text = "Erneut abrufen"
        DatumZeit.Text = System.DateTime.Now.ToLongTimeString()
    End Sub
End Class

```

Listing 8.31: Caching von Objekten mit automatischen Recycling (CacheObjects.aspx.vb)

Das Programm nutzt intensiv das `Cache`-Objekt. Während die Zuweisung zur Kollektion sicher weniger Probleme bereitet, ist ein Blick auf die Anwendung der Methode `Add` angebracht.

Wie es funktioniert

CacheDependency Grundsätzlich sind derart explizit erzeugte `Cache`-Objekte immer von zwei Dingen abhängig: Änderung an einer oder mehreren Dateien oder Ablauf eines Zeitraumes bzw. – alternativ dazu – Verstreichen eines Zeitraumes, in dem das Objekt nicht benutzt wurde. Die Abhängigkeit von der Datei wird mit der Klasse `CacheDependency` festgelegt. Hier wird die Abhängigkeit vom Programm selbst festgelegt:

```
Dim cd As CacheDependency
    = New CacheDependency
        (Server.MapPath(Request.Path))
```

Add im Detail Die Methode `Add` verlangt zuerst einen Namen für den Speicherplatz:

```
Cache.Add("CityCache",
```

Dann wird das zu speichernde Objekt festgelegt:

```
al,
```

Nun wird die Abhängigkeit übergeben:

```
cd,
```

Es folgt das Ablaufdatum. Der andere Zeitparameter muss dann Null sein:

```
DateTime.Now.AddSeconds(5),
```

Dieser Parameter kann nur dann ungleich Null sein, wenn der vorhergehende Null ist. Die Angabe eines Zeitraumes legt fest, wie lange das Objekt unbenutzt sein muss, damit es entfernt wird:

```
TimeSpan.Zero,
```

Es folgt noch eine Priorität. Wenn der Server mehr Speicher benötigt, werden unter anderem auch `Cache`-Objekte entfernt. Die Reihenfolge, in der das passiert, legt dieser Parameter fest:

```
CacheItemPriority.Normal,
```

Zuletzt wird noch ein Delegat zugewiesen, der auf eine Methode zeigt, die das Ereignis »Objekt wurde aus dem Speicher entfernt« behandelt. Im Beispiel wird das Objekt dann neu aufgebaut:

```
onRemove)
```

Die Programmierung des Delegaten erfolgte bereits im Kopf der Klasse:

CachedItemRemovedCallback

```
Private onRemove As CacheItemRemovedCallback = Nothing
```

Die Zuweisung erfolgte vor der Erstellung des Cache-Objekts:

```
onRemove = New CacheItemRemovedCallback(AddressOf RemovedCallback)
```

Nun passiert Folgendes: Immer wenn das Objekt aus dem Cache verschwindet, wird die Callback-Methode aufgerufen. Im Beispiel passiert dort Folgendes: Zuerst wird der Speicher wieder aufgebaut. Dann wird die Ursache für das Entfernen an die Kollektion angehängt, sodass sie sichtbar wird. Das dient hier nur der Demonstration.

Caching von Objekten

Daten im Cache halten

Item
Berlin
Paris
Madrid
Londond
Wien
Oslo
Stockholm
Removed CityCache
00:15:36
Erneut abrufen

Abbildung 8.21: Reaktion eines im Cache gehaltenen Objekts

Die Aufzählung `CacheItemRemovedReason` enthält die Hinweise `Removed` (regulär entfernt), `DependencyChanged` (Datei verändert), `Expired` (Datum abgelaufen) oder `Underused` (war zu lang unbenutzt oder es wurde Speicher gebraucht).

CachedItemRemovedReason

Die Anwendung der Klasse `CacheDependency` ist vor allem dann sinnvoll, wenn Sie Elemente mit XML-Daten füllen, wie es an vielen Stellen in diesem Buch gemacht wird. Wenn sich nun die Daten in diesen Dateien ändern, sollten die zwischengespeicherten Objekte schleunigst entfernt und gegen aktuellere Kopien ausgetauscht werden.

Anwendung

8.7.3 Allgemeine Tipps zur Optimierung

Es gibt einige Aspekte, die den Datenverkehr zwischen Server und Browser optimieren können. Einige hängen mit ASP.NET zusammen, andere nicht. Auch der Zusammenhang mit dem Verbrauch an Bandbreite, also dem Produkt aus Datenmenge und Übertragungsgeschwindigkeit, sollte betrachtet werden. Die Übertragungsgeschwindigkeit können Sie kaum beeinflussen, die Datenmenge dagegen schon.

Caching verwenden

Verwenden Sie das Caching, wie es zuvor beschrieben wurde. Nutzen Sie auch den Objektspeicher mit der Klasse `Cache`. Beachten Sie aber, dass hoher programmiertechnischer Aufwand für eine saubere Nutzung des Speichers den Vorteil zunichte machen kann. Der Einsatz lohnt nur, wenn es »einfach geht«.

Funktionen deaktivieren, die nicht benötigt werden

Sitzungs- Management

Wenn Sie das Sitzungs-Management nicht benötigen, schalten Sie es ab. Dies geht am einfachsten mit der `@Page`-Direktive:

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

Wenn Sie das Sitzungs-Management benötigen, aber keine Daten speichern, setzen Sie die Option wenigstens auf »nur lesen«:

```
<% @Page EnableSessionState="ReadOnly" %>
```

Wählen Sie auch den verwendeten Provider mit Bedacht. Informationen dazu finden Sie im Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778.

Anzeigestatus- Management

Wenn Sie das Anzeigestatus-Management (*ViewState*) nicht benötigen, schalten Sie es ab. Dies geht am einfachsten mit der `@Page`-Direktive:

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

Nutzen Sie diese Option auch für einzelne Steuerelemente; dort ist die `@Control`-Direktive zuständig:

```
<% @Control EnableViewState="false" %>
```


Sie können dies übrigens noch feiner differenzieren, indem der Anzeigestatus nur für einzelne Elemente unterdrückt wird:

```
<asp:DropDownList EnableViewState="false" runat="server"/>
```

Fehlerhilfen nach der Freigabe abschalten

Wenn Sie mit Visual Studio .NET-Debugging oder dem SDK-Debugger gearbeitet haben und Ihre Applikation fertig ist, vergessen Sie nicht, den Debug-Modus vor der letzten Übersetzung abzuschalten. Sonst schleppt Ihr Code immer Debug-Daten mit.

Schalten Sie unbedingt die Debug-Option in der @Page-Direktive aus:

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

Nachdenken über die Gestaltung von Applikationen

Ein paar stichwortartige Hinweise, den cleveren Einsatz von ASP.NET-Funktionen betreffend, finden Sie nachfolgend:

- ▶ Brauchen Sie *AutoPostBack*? Jede Runde zum Server kostet Zeit und Bandbreite.
- ▶ Brauchen Sie Datenbanken? Vielleicht können kleine Datenmengen in XML-Dateien gehalten werden, die sich leichter zwischenspeichern lassen.
- ▶ Haben Sie clientseitige Kontroll-Steuerelemente verwendet? Sparen Sie den Weg zum Server zurück, wenn Ihre Nutzer moderne Browser mit JavaScript verwenden.
- ▶ Nutzen Sie eigene Steuerelemente? Vorkompilierte Steuerelemente sind schneller geladen als einfache *aspx*-Seiten.
- ▶ Achten Sie immer auf *PostBack*? Denken Sie daran, dass Sie Daten nicht mehrfach beschaffen oder berechnen, wenn das Formular zurückgesendet wird.
- ▶ Geben Sie viel Text mit Steuerelementen aus? Fassen Sie möglichst große Blöcke zusammen, sodass die kleinstmögliche Anzahl Steuerelemente benötigt wird.
- ▶ Arbeiten Sie viel mit *Try-Catch*? Dies ist kein Mittel, um den Programmfluss zu steuern, sondern eine *Ausnahmebehandlung*, die also – wie der Name sagt – ausnahmsweise zum Einsatz kommt.

- Verwenden Sie COM? Es spielt in diesem Buch keine Rolle, aber vielleicht nutzen Sie alte Komponenten. Muss das sein? Das Umschalten zwischen verwaltetem und unverwaltetem Code ist aufwändig.

Leistungsfähige Funktionen sparen zwar einigen Entwicklungsaufwand, fordern aber auch mehr Ressourcen der Hardware. Überlegungen über den sinnvollen Einsatz sparen Geld und erfreuen auch den Nutzer, denn Seiten werden schneller geladen.

9 Datenbanken und ADO.NET

In diesem Kapitel wird praktisch mit Datenbanken umgegangen. Im Mittelpunkt steht ADO.NET. Behandelt werden auch weitere Aspekte der Steuerelemente, die Datenbankergebnisse aufnehmen und anzeigen können.

9.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

9.1.1 Über dieses Kapitel

Dieses Kapitel behandelt den praktischen Umgang mit Datenbanken und XML im Zusammenhang mit der Erstellung von ASP.NET-Programmen. Schwerpunkt bildet eine Einführung in ADO.NET. Dies ist der auf die Datenverarbeitung und den Datenbankzugriff spezialisierte Teil der Klassen des .NET-Frameworks. Vorausgesetzt werden die im Kapitel 5, »Grundlagen der Datenspeicherung« ab Seite 345 vermittelten Grundlagen. Nachdem mit den dort gezeigten Methoden die Abfrage von Daten oder auch das Einfügen gelang, soll nun die Schnittstelle zum laufenden Programm im Mittelpunkt stehen. Abschnitt 9.2, »Grundlagen zu ADO.NET« ab Seite 825 widmet sich diesem Thema. Neben den praktischen Techniken, der Programmierung mit den entsprechenden Klassen, wird auch Hintergrundwissen über die Philosophie hinter ADO.NET vermittelt, was wesentlich dazu beiträgt, eigene professionelle Anwendungen zu entwickeln.

Anschließend geht es um die Methoden der Ausgabe von strukturierten und hierarchischen Daten in Webseiten. Im Mittelpunkt des Abschnitts 9.4, »Umgang mit Datenbanken in Visual Studio .NET« ab Seite 921 stehen die Daten-Steuerelemente und allgemeine Techniken des Zugriffs auf Datenquellen aus Webformularen heraus. Mit Hilfe dieser Steuerelemente wird eine der fortschrittlichsten Funktionen in ASP.NET gezeigt. Denn durch die komplexen und trotzdem kompatiblen Steuerelemente ist die Programmierung umfangreicher Anwendungen ohne Schwierigkeiten möglich.

Als Datenquelle kommen nicht nur klassische SQL-Datenbanken und XML in Frage, sondern beispielsweise auch der Indexdienst. Der Abschnitt 9.6, »Indexdienst programmieren« ab Seite 937 zeigt, wie der Indexdienst programmiert wird und mit welchen Mitteln der Einsatz in einem ASP.NET-Programm zum Aufbau einer leistungsfähigen Suchmaschine möglich ist.

9.1.2 Wegweiser in die Referenz

Die gesamte Klassenhierarchie in ADO.NET ist sehr umfangreich. Die folgenden Darstellungen zeigen die »Hauptklassen«, deren Schnittstellen und einfache Abhängigkeiten. Wo es notwendig ist, wird dies im Text noch vertieft. Die Klassennamen und Namensräume sind hilfreich, wenn Sie sich in der Referenz orientieren müssen.

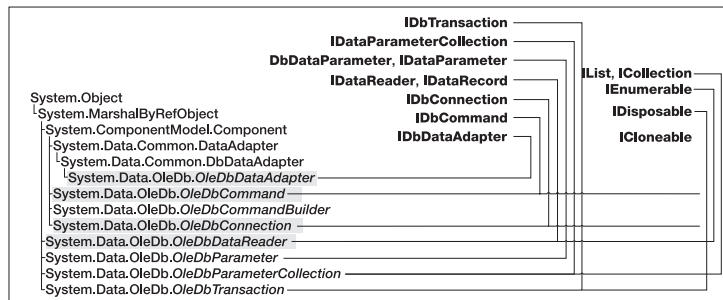


Abbildung 9.1: Schnittstellen und Klassenhierarchie des OleDb-Providers

Die Klassen unterhalb des Namensraumes `System.Data.OleDb` stammen von denselben Schnittstellen, wie die unter `System.Data.SqlClient`. Dies sichert die Kompatibilität der Parameter.

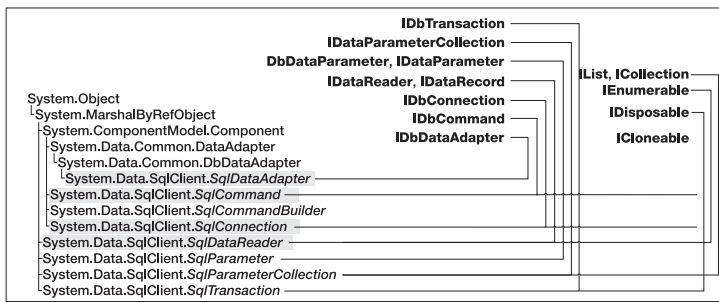


Abbildung 9.2: Schnittstellen und Klassenhierarchie des *SqlClient*-Providers

Die jeweils wichtigsten Klassen sind außerdem grau hinterlegt.

9.2 Grundlagen zu ADO.NET

Im Kapitel 5, »Grundlagen der Datenspeicherung« ab Seite 345 wurden Sie mit den Grundlagen der Datenspeicherung vertraut gemacht. Dabei wurden zwei grundlegende Techniken betrachtet:

- ▶ Relationale Datenbankmanagementsysteme und deren Abfragesprache SQL
- ▶ XML als hierarchisches Datenspeicherformat und die Substandards XSLT zur Transformation und XPath zur Selektion

In diesem Abschnitt geht es nun um die Verknüpfung der Datenquellen mit einem ASP.NET-Programm. Dabei spielt XML eine herausragende Rolle, weil es in .NET zur internen Speicherung von Daten verwendet wird und fast immer sowohl für den Im- als auch Export zur Verfügung steht.

9.2.1 Prinzip der Arbeit mit ADO.NET

Eines der herausragenden Merkmale der Datenhaltung und -verarbeitung in ADO.NET ist die verbindungslose Arbeitsweise. Dabei wird zwar zum Zeitpunkt der Abfrage von Daten eine Verbindung zur Datenbank hergestellt, die ermittelten Daten werden aber danach vollständig in .NET gehalten und der Applikation quasi lokal zur Verfügung gestellt. Das hat Vorteile, wenn das Design des Programms darauf hinreichend Rücksicht nimmt. Sie müssen dieses Prinzip deshalb vollständig verstanden haben, um davon profitieren zu können.

**Verbindungslose
Arbeitsweise**

**Persistierung
in XML**

Die Abkopplung der Daten von der Datenbank erlaubt es, diese leicht zwischen den Schichten einer Applikation zu transportieren. Ebenso einfach ist eine »Persistierung« möglich, also die Speicherung in einer nichtflüchtigen Form. Als Speicher- und Transportformat wird in allen Fällen XML verwendet. Um die Art und Weise kümmert sich .NET intern, sodass dafür keine umfassenden XML-Kenntnisse nötig sind.

Lokal liegt in ADO.NET eine mehr oder minder komplexe Kopie der Datenbank vor. Das hat zur Folge, dass die Objekte zur Datenspeicherung mehr als eine Tabelle enthalten können. Neben reinen Daten sind auch Relationen erfassbar. Letztlich ist es eine kleine und durchaus leistungsfähige Datenbank, die in den entsprechenden Klassen gebildet wird. Tatsächlich ist es möglich, auch ohne hinterlegte Datenbank mit diesen Klassen zu arbeiten. Für kleinere Applikationen, die die Leistung eines RDBMS nicht benötigen, mag dies ausreichend sein. Vor allem dann, wenn als Speicherformat XML in Erwägung gezogen wird, zeigt sich die wahre Leistungsfähigkeit der ADO.NET-Klassen.

Betrachtet man sich die Möglichkeiten in ADO.NET, stellt sich die Frage nach der Notwendigkeit eines Systems wie SQL Server. Es ist natürlich eine Tatsache, dass ein Datenbankserver weitaus leistungsfähiger und mächtiger ist als die .NET-Klassen. Es kommt auf das geschickte Zusammenspiel beider Welten an. So sollte der Einsatz von gespeicherten Prozeduren und Funktionen im SQL Server 2000 eine zentrale Rolle einnehmen. Denn nur so erreichen Sie eine optimale Geschwindigkeit beim Zugriff auf Daten. Der SQL Server ist dann »für's Grobe« zuständig, während die Feinauswahl an Daten in ADO.NET stattfindet.

Datenquellen

ADO.NET beschränkt den Zugriff auf Datenquellen nicht auf den SQL Server. Nachwievor spielt MS Access eine wichtige Rolle, denn bei kleinen Projekten oder für die Ablage von selten benötigten Daten ist der Einsatz eines teuren Datenbankservers nicht notwendig. Darüber hinaus stellen viele Provider zwar den Zugriff auf *mdb*-Dateien frei, verlangen jedoch für den SQL Server einen teilweise happigen Aufpreis. Sie müssen sich also immer sehr genau überlegen, ob es wirklich ein SQL Server 2000 sein muss.

Abgesehen von diesen beiden Datenbankmanagementsystemen aus dem Hause Microsoft kommen in praktischen Projekten weitere Systeme zum Einsatz. So wird sicher die eine oder andere Site mit Oracle oder IBMs DB2 betrieben werden. Nicht zu vergessen ist auch der Indexdienst, der ebenfalls als Datenquelle dienen kann.

ADO.NET kennt in der Version 1.0 standardmäßig zwei so genannte Provider, die jeweils durch ein eigenes Paket von Klassen gebildet werden:

Zwei Provider in der Version 1.0

- ▶ SqlClient
- ▶ OleDb

Der erste, »SqlClient«, dient ausschließlich dem optimierten Zugriff auf den SQL Server ab Version 7. Mit »OleDb« erhalten Sie die Möglichkeiten, auf alle Datenbanken zugreifen zu können, die über einen entsprechenden Treiber verfügen. OleDb ist als Satz von Schnittstellen ausgeführt, der zu jeder Datenquelle einen speziellen Treiber (den so genannten Provider) bereitstellt und die Übertragung der Daten auf einem einheitlichen Weg ermöglicht. OleDb basiert direkt auf dem Windows-API (Application Programming Interface), das für Sprachen wie C++ entwickelt wurde. Die .NET-Klassen mit dem Präfix OleDb sind ein Wrapper auf diese Schnittstelle. Der Zugriff ist etwas langsamer als beim SQL Server. OleDb wird typischerweise für MS Access verwendet.

Mitte 2002 waren die Klassen für die dritte Zugriffsmöglichkeit noch nicht fertig verfügbar: ODBC. ODBC (Open DataBase Connectivity) ist eine schon etwas ältere universelle Schnittstelle zu Datenbanken. Sie wurde von Microsoft entwickelt, hat aber auch in anderen Welten, auch unter Unix, weite Verbreitung gefunden. Da es weit mehr Datenbanken mit ODBC-Treibern als mit OleDb gibt, ist ODBC noch lange nicht tot. Allerdings sollte der Einsatz wirklich nur dann stattfinden, wenn OleDb nicht verwendbar ist, weil die betreffende Datenbank es nicht unterstützt.

Neu in Version 1.1

In diesem Buch wird auf ODBC nicht weiter eingegangen, weil ODBC für die hier bevorzugt behandelten Systeme MS Access und SQL Server 2000 nicht notwendig ist. Die Anwendung bereitet aber mit dem vermittelten Wissen wenig Probleme.



Eine weitere grundlegende Datenquelle ist XML. Damit kann sich der Softwareentwickler eine ganze Welt von Daten erschließen,

denn viele Programme verfügen inzwischen über entsprechende Exportmöglichkeiten. XML wird in .NET »per Definition« unterstützt und erfordert keinen gesonderten Treiber oder Provider oder dergleichen.

Die Geschichte der Datenzugriffsmethoden

ADO.NET ist die aktuellste Evolutionsstufe einer bereits sehr langen Kette von Methoden, Protokollen und Verfahren.

Das erste Verfahren zur Einführung einer universellen Schnittstelle zu Datenbanken war ODBC (Open DataBase Connectivity). Entwickelt wurde es von Microsoft und einigen Partnern. Erste Versionen tauchten 1990 auf. ODBC setzt voraus, dass ein entsprechender Treiber für jede Datenbank vorliegt, die darüber angesprochen werden soll. Eine sehr weit führende Abstraktionsschicht soll – zumindest theoretisch – den Wechsel der Datenbank ohne Änderungen an der Applikation erlauben. ODBC hat sehr weite Verbreitung gefunden und wird in .NET durch Wrapperklassen unterstützt.

Die »Data Access Objects« (DAO) bilden ein weiteres Datenzugriffsmodell, das mit Access und anderen ISAM-Datenbanken eingeführt wurde, unter anderem für Visual Basic 3. DAO setzt auf ODBC auf und kann universell verwendet werden, ist dann aber relativ langsam. Nur der Zugriff auf Jet-Datenbanken ist optimiert (dazu gehört praktisch nur Access).

Die »Remote Data Objects« (RDO) sind der Nachfolger von DAO und erlauben zusätzlich den Zugriff auf entfernte Datenquellen, beispielsweise über das Internet. Die Nachteile von DAO wurden nicht beseitigt, weshalb sich RDO nie richtig durchsetzen konnte.

All diese Technologien verursachten in der Praxis Probleme. So war die Programmierung mit der DBLib (die native Schnittstelle zum SQL Server) oder ODBC (auf Programmier Ebene) für den Datenbankzugriff sehr kompliziert. Skriptsprachen boten erst gar keine Möglichkeit der Nutzung. DAO und RDO dagegen bauten auf ODBC auf und legten damit eine weitere Schicht zwischen Datenbank und Programmierschnittstelle. Dies funktionierte zwar nun auch in Skriptspra-

chen, war aber sehr langsam. Alle Schnittstellen hatten ein streng hierarchisches Modell und damit einen großen Overhead. Es gab also gute Gründe, diese verschiedenen Versionen zu vereinen und dabei die Vorteile zu erhalten, die Nachteile aber zu reduzieren – dies genau ist mit OleDb gelungen. Eine Wrapperklasse setzt in .NET konsequent auf OleDb.

ADO wurde mit der Version 1.5 eingeführt. Die Vorgängerversion 1.0 kam kaum zum Einsatz. ADO 1.5 war Bestandteil des Option Packs von Windows NT 4, von Visual InterDev und vieler Programmiersprachen von Microsoft mit der Versionsnummer 5. ADO 2.0 wurde mit Visual Studio 6.0 eingeführt und damit mit den Programmiersprachen der Versionsnummer 6. Mit Office 2000 und den ersten Betaversionen von Windows 2000 erfolgte die Einführung von Version 2.1. Mit dem Erscheinen von Windows 2000 wurde ADO erneut überarbeitet, es stand dort mit der Version 2.6 zur Verfügung. Die letzte Version vor ADO.NET war 2.7.

ADO.NET vereint im Prinzip die Technologien des Datenzugriffs, wobei als eigentlicher Provider entweder OleDb oder ODBC zum Einsatz kommt. Für den SQL Server gibt es eine native Schnittstelle, die die Programmierung gegenüber DBLink signifikant vereinfacht. Darüber hinaus sind alle drei Bibliotheken weitgehend identisch aufgebaut, sodass Sie mit der Kenntnis von OleDb auch die SqlServer-Klassen beherrschen.

9.2.2 Die Architektur von ADO.NET

ADO.NET bietet, wie bereits beschrieben, zwei Provider, ab Version 1.1 derer vier. Aus Sicht der Architektur sind diese jedoch vergleichbar aufgebaut, sodass eine allgemeingültige Darstellung gelingt.

Die Basisobjekte der Provider

Jeder Provider besteht aus vier Basisobjekten, die aus den entsprechenden Klassen abgeleitet werden. Dazu kommen viele Hilfsklassen. Alle zusammen bilden ADO.NET. Konkret geht es immer um folgende Objekte:

- XxxConnection
- XxxCommand
- XxxDataReader
- XxxDataAdapter

Für den Präfix »Xxx« steht in der Praxis einer der Namen »SqlClient«, »OleDb« oder »Odbc«. Ansonsten verhalten sich die Objekte gleichartig, zumindest soweit keine Eigenschaften verwendet werden, die auf eine spezifische Datenbank beschränkt sind.

Mit den Basisobjekten gelingt der Zugriff auf eine Datenbank. Die ebenfalls bereits erwähnte Abbildung der Datenbank – mit Struktur und Daten – wird mit Hilfe eines `DataSet`-Objekts gebildet. Die Darstellung auf dieser Seite ist nun bereits unabhängig von der verwendeten Datenquelle.

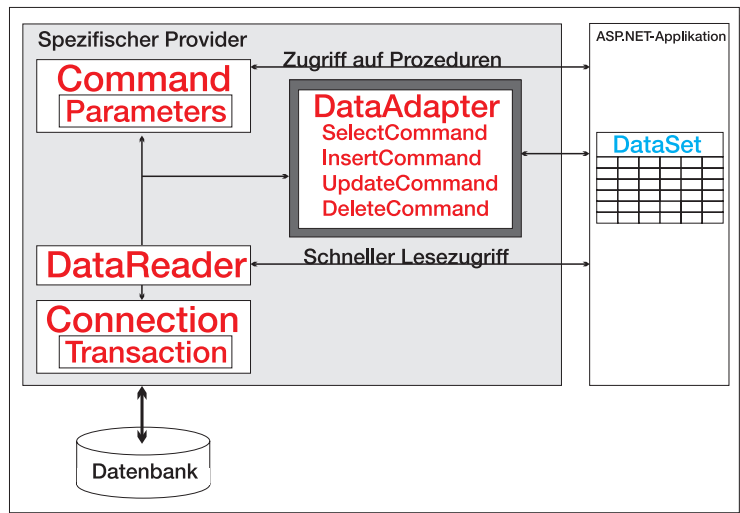


Abbildung 9.3: Bestandteile eines Providers

Die Abbildung zeigt den Aufbau eines spezifischen Providers. Der grau hinterlegte Teil ist also datenbankabhängig. Unbedingt benötigt wird immer eine Verbindung zur Datenbank (`Connection`). Die praktische Implementierung heißt dann entweder `SqlConnection` oder `OleDbConnection`. Ebenso lassen sich aus den anderen allgemeinen Namen die entsprechenden Klassennamen ableiten. Damit weitere Provider leicht und dazu passend entwickelt

werden können, gibt es natürlich entsprechende Schnittstellen. Konsultieren Sie die Abbildungen 9.1 und 9.2, um sich über die Abhängigkeiten zu informieren.

Verwendete Namensräume

Für jeden Provider gibt es einen eigenen Namensraum. Außerdem sind allgemeine Klassen in einer übergeordneten Ebene untergebracht. Sie benötigen auf jeden Fall folgenden Eintrag:

Namensraum

```
Imports System.Data
```

Wenn Sie mit OleDb arbeiten, wird außerdem folgender Namensraum erwartet:

```
Imports System.Data.OleDb
```

Für den Umgang mit den SQL Server wird dagegen dieser Namensraum verwendet:

```
Imports System.Data.SqlClient
```

Zwei weitere Namensräume unterhalb `System.Data` werden nur in speziellen Fällen benötigt. Darauf wird an den entsprechenden Stellen im Buch gesondert hingewiesen.

9.2.3 Verbindung zu einer Datenbank aufbauen

Der erste Programmschritt zur Nutzung einer Datenbank besteht immer im Aufbau einer Verbindung. Jeder Provider liefert dazu eine Implementierung der Schnittstelle `IDbConnection`. Für den SQL Server ist dies die Klasse `SqlConnection`, für OleDb-Datenbanken dagegen `OleDbConnection`. Kommandos, die direkt an die Datenbank gerichtet sind, können nur ausgeführt werden, wenn die Verbindung geöffnet ist. .NET arbeitet generell mit dem Verbindungspooling, bei dem eine Verbindung solange wie möglich offen gehalten wird, um allen Prozessen zur Verfügung zu stehen. Das beschleunigt den Ablauf der Datenzugriffe deutlich.

Allerdings ist zu beachten, dass die Verbindung am Ende eines Programms nicht geschlossen wird. Sie müssen die entsprechende `Close`-Methode aufrufen, um die Verbindung wieder zu schließen. Auch die Garbage Collection erledigt dies nicht. Das werden Sie schnell bemerken, wenn Sie MS Access verwenden, wo nur eine Verbindung möglich ist. Der SQL Server, der mehrere Verbindun-

**Explizites
Schließen
erforderlich**

gen erlaubt, wird möglicherweise nicht sofort mit einer Fehlermeldung reagieren. Viele offene Verbindungen kosten aber Leistung und sind deshalb zu vermeiden. Zudem sichert das verbindungslose Konzept von ADO.NET den Datenzugriff auch bei geschlossener Verbindung. Sie müssen sorgfältig überlegen, wann Sie die Daten holen, Teile daraus verwenden und wie die Aktualisierung der Datenbank, wenn erforderlich, erfolgen soll.

Eine Verbindung zum SQL Server 2000 aufbauen

Verbindungszeichenfolge

Um eine Verbindung zu eröffnen, gibt es mehrere Wege. In jedem Fall benötigen Sie ein entsprechendes Verbindungsobjekt und die passende Verbindungszeichenfolge. Wenn es sich um den SQL Server 2000 handelt, hat diese folgenden Aufbau:

```
server=<server>; database=<datenbank>; uid=<user>; pwd=<kennwort>
```

Anpassen der Einstellungen

Läuft Ihr SQL Server lokal, tragen Sie für `<server>` »localhost« ein. Der Name der Datenbank dürfte eindeutig sein. Wenn Sie die zum Üben in Kapitel 5 verwendete Datenbank »Shop« verwenden, tragen Sie diesen Namen ein. Die folgenden Beispiele beziehen sich ebenfalls darauf. Als Benutzername ist der Datenbankbenutzer anzugeben, meist ist dies »sa« (Server Administrator). Soweit es sich um eine ungesicherte Standardinstallation handelt, bleibt das Kennwortfeld leer. Sie können nun entweder das Objekt instanziiieren und die Methode `Open` verwenden oder die Verbindungszeichenfolge dem Konstruktor übergeben.



Die Anpassungen müssen Sie in allen folgenden Programmen vornehmen, die mit einer fest programmierten Verbindungszeichenfolge belegt sind.

Das folgende Beispiel testet die Verbindung und informiert über Erfolg oder Misserfolg:

```
Imports System.Data
Imports System.Data.SqlClient

Public Class SqlConnectionCheck
    Inherits System.Web.UI.Page
    Protected ConnectErgebnis As Label

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        Dim conSql As SqlConnection = New SqlConnection()
```

```

conSql.ConnectionString = "server=srv1; database=Shop; uid=sa; pwd=password"

Try
    conSql.Open()
    If conSql.State = ConnectionState.Open Then
        ConnectErgebnis.Text = "Verbindung geöffnet"
        conSql.Close()
    End If
Catch e As Exception
    ConnectErgebnis.BackColor = Color.Red
    ConnectErgebnis.Text = e.Message
End Try
End Sub

End Class

```

Listing 9.1: Programm zum Testen der Verbindung (SqlConnectionCheck.aspx.vb)

Hier wird zuerst ein einfaches Verbindungsobjekt erzeugt. Nun kann die Verbindungszeichenfolge der Eigenschaft `ConnectionString` zugewiesen werden. Die Methode `Open` öffnet dann diese Verbindung. Es ist immer sinnvoll, diesen Teil in einem *Try-Catch*-Zweig zu verarbeiten, da hier erfahrungsgemäß Fehler auftreten können. Die Ausgabe der Fehlermeldung ist möglicherweise aufschlussreich:

Wie es funktioniert

SQL Server Verbindung

Prüfen der Verbindung

SQL Server ist nicht vorhanden, oder der Zugriff wurde verweigert

Abbildung 9.4: Reaktion des Programms, wenn ein Verbindungsfehler auftrat

Die Ausgabe erfolgt in ein `Label`-Steuerelement. Statt der Ausgabe der Erfolgsmeldung schließt sich bei realen Programmen mit Datenbankzugriff dann der Code mit den Datenbankoperationen an.

SQL Server Verbindung

Prüfen der Verbindung

Verbindung geöffnet

Abbildung 9.5: Erfolgsmeldung, wenn die Verbindung gelang

Verbindung mit dem OleDb-Provider öffnen

Neben dem SQL Server werden Sie möglicherweise mit Access oder einer größeren Datenbank wie Oracle arbeiten wollen. Dann müssen Sie den OleDb-Provider verwenden.

Verbindungszeichenfolgen

Der OleDb-Provider wird durch den Schlüssel *Provider=* angesprochen. Die folgende Aufstellung zeigt einige typische Verbindungszeichenfolgen. Variable Angaben, die Sie entsprechend Ihren Bedingungen anpassen müssen, sind kursiv dargestellt. Die Übersicht ist nicht vollständig, sondern zeigt nur die wichtigsten Eigenschaften. Weitere Parameter hängen von der jeweiligen Datenbank ab und sind deren Dokumentation zu entnehmen.

OLEDB Access

Für den Zugriff auf MS Access (Version 2000 und 2002) verwenden Sie die folgende Verbindungszeichenfolge:

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=database.mdb
```

► *Provider*

Dieser Parameter bezeichnet den OleDb-Provider für die konkret verwendete Datenbank.

► *Data Source*

Name der Access-Datenbank, gegebenenfalls mit vollständigem, physischen Pfad. Verwenden Sie *Server.MapPath*, um den Pfad aus der virtuellen Angabe heraus zu ermitteln.

SQL Server

Für den Zugriff auf SQL Server 7 oder 2000 verwenden Sie folgende Verbindungszeichenfolge:

```
Provider=SQLOLEDB; Data Source=server;  
Database=datenbank; UID=nutzername; PWD=kennwort
```

► *Provider*

Dieser Parameter bezeichnet den OleDb-Provider für die konkret verwendete Datenbank.

► *Data Source*

Der Name des SQL Servers. Je nach Verbindung muss der Name vollständig qualifiziert sein.

► *Database*

Name der Datenbank, die standardmäßig verwendet werden soll.

► *UID*

Der Name des Datenbankbenutzers für die verwendete Datenbank. Der Standardbenutzer des SQL Servers heißt »sa«.

► *PWD*

Das Kennwort, wenn eines vergeben wurde.

Den OleDb-Provider für den SQL Server sollten Sie nur verwenden, wenn es unbedingt notwendig ist, portierbare Anwendungen zu schreiben, bei denen ein späterer Wechsel des Datenbankmanagementsystems absehbar ist. Wird grundsätzlich der SQL Server eingesetzt, ist der native Zugriff über den SqlClient-Provider immer vorzuziehen. Dieser Provider nutzt das spezielle TDS-Protokoll (Tabular Data Stream) für die Kommunikation mit dem SQL Server.



Für den Zugriff auf den Indexdienst von Windows 2000 oder XP Professional bzw. .NET-Server verwenden Sie folgende Verbindungszeichenfolge:

Index Server

Provider=MSIDX; Data Source=katalogname

► *Provider*

Dieser Parameter bezeichnet den OleDb-Provider für die konkret verwendete Datenbank.

► *Data Source*

Der Name des Katalogs, der verwendet werden soll. Der Katalog kann mit der Managementkonsole des Indexdienstes angelegt werden. Mehr Informationen zur Nutzung des Indexdienstes finden Sie im Abschnitt 9.6, »Indexdienst programmieren« ab Seite 937.

Für den Zugriff auf den Oracle (8i oder 9i) verwenden Sie folgende Verbindungszeichenfolge:

Oracle

Provider=MSDAORA; Data Source=server;

Initial Catalog=StandardDatenbank; Integrated Security=SSPI

► *Provider*

Dieser Parameter bezeichnet den OleDb-Provider für die konkret verwendete Datenbank.

► *Data Source*

Der Name des Oracle-Servers. Je nach Verbindung muss der Name vollständig qualifiziert sein.

► *Initial Catalog*

Der Name der Datenbank, die verwendet werden soll.

Mit diesem Wissen ausgestattet sollte es nun gelingen, die im Kapitel 5 bereits verwendete Access-Datenbank *shop.mdb* zu verwenden. Das folgende Programm erledigt dies:

```
Imports System.Collections
Imports System.Data
Imports System.Data.OleDb

Public Class OleDbConnectionCheck
    Inherits System.Web.UI.Page
    Protected ConnectErgebnis As Label
    Private Sub Page_Load(ByVal sender As Object, _
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        Dim conSql As OleDbConnection = New OleDbConnection()
        Dim Datenbank As String _
            = Server.MapPath("data/shop.mdb")
        conSql.ConnectionString _
            = "Provider=Microsoft.Jet.OLEDB.4.0; _
            Data Source=" + Datenbank
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                ConnectErgebnis.Text = "Verbindung geöffnet"
            Else
                ConnectErgebnis.Text = "Verbindung konnte _
                    nicht geöffnet werden"
                conSql.Close()
            End If
        Catch e As Exception
            ConnectErgebnis.BackColor = Color.Red
            ConnectErgebnis.Text = "Fehler: " & e.Message
        End Try
    End Sub
End Class
```

Listing 9.2: Testen der Verbindung zu einer Access-Datenbank
(*OleDbConnectionCheck.aspx.vb*)

Abgesehen vom Namensraum, der Basisklasse und der Verbindungszeichenfolge gibt es keine Unterschiede zu dem in Listing 9.1 gezeigten Programm. Dies zeigt, dass die Kenntnis der einen Methode (SqlClient) sehr leicht auf die andere (OleDb) übertragbar ist. Ebenso sind künftige Provider, die auf denselben Schnittstellen basieren, ohne großen Lernaufwand einsetzbar.

Zentrale Verwaltung der Verbindungszeichenfolgen

Wenn Sie ein größeres Projekt schreiben, das aus mehreren Code-Dateien besteht, ist die wiederholte Angabe der Verbindungszeichenfolge unbedingt zu vermeiden. Änderungen sind sonst mit mehreren Code-Eingriffen umzusetzen, was nicht im Sinne eines guten Programmierstils ist.

Eine Möglichkeit ist schon beim alten ASP 3.0 häufig verwendet worden. Dort wurde eine Applikationsvariable angelegt, die die Verbindungszeichenfolge enthält. Die Erzeugung erfolgte zum Startzeitpunkt der Applikation, die Definition in der Datei *global.asa*. Sicher ist dieses Prinzip einfach auf ASP.NET übertragbar, in Analogie zum alten System wird nun die *global.asax* verwendet. Informationen über Applikationsvariablen finden Sie im Abschnitt 8.6, »Applikationsmanagement« ab Seite 802.

Nachteil dieser Methode ist die Notwendigkeit eines Applikationsstarts, um Änderungen an den Daten wirksam werden zu lassen. Der Einbau der Einträge in eines der Ereignisse, die zu Beginn einer Anforderung (OnRequest) ausgelöst werden, ist wiederum mit einer geringen Leistungseinbuße verbunden.

**global.asax
verwenden**

Eleganter ist die Nutzung der Konfigurationsdatei *web.config*. Änderungen wirken sich sofort aus, da ASP.NET die Werte dann neu einliest, wenn sich der Dateiinhalt ändert. Ansonsten verbleibt der Inhalt immer im Speicher und steht damit unmittelbar zur Verfügung. Zudem sind individuelle Einträge hier zusammen mit allen anderen konfigurierbaren Parametern untergebracht, was die Verwaltung vereinfacht. Grundlegende Informationen über den Umgang mit der Datei *web.config* finden Sie im Abschnitt 10.3, »Konfiguration von Applikationen: web.config« ab Seite 962.

**web.config
verwenden**

Für den hier verfolgten Zweck, die Verbindungszeichenfolge zu speichern, eignet sich der Abschnitt `<appSetting>` im Zweig `<configuration>`. Sie können beliebig viele `<add>`-Tags einfügen, die sich dann aus dem Code heraus abfragen lassen. Praktisch könnte das folgendermaßen aussehen:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="SqlConnection"
      value="server=srv1; database=Shop; uid=sa; pwd=password"/>
    <add key="OleDbConnection"
```

```

        value="Provider=Microsoft.Jet.OLEDB.4.0; ➤
        Data Source={0}"/>
    <add key="MDB"
        value="data/Shop.mdb"/>
    </appSettings>
    ...

```

Listing 9.3: Ausschnitt aus der Datei web.config mit Verbindungszeichenfolgen

Beim in Listing 9.1 gezeigten Beispiel würde die Übernahme der Verbindungszeichenfolge aus der Konfigurationsdatei nun folgendermaßen aussehen:

```

Dim conSql As SqlConnection = New SqlConnection()
conSql.ConnectionString ➤
    = ConfigurationSettings.AppSettings("SqlConnection")

```

*Listing 9.4: Übernahme der Verbindungszeichenfolge aus web.config
(Ausschnitt aus SqlConnectionCheckConfig.aspx.vb)*

Namensraum Voraussetzung für die Nutzung der statischen Klasse `ConfigurationSettings` ist die Einbindung des entsprechenden Namensraumes:

```
Imports System.Configuration
```

Für die Access-Datenbank wurden im Beispiel zwei Parameter vorbereitet. Dies zeigt, dass auch Verknüpfungen mit Formatparametern (hier: {0}) möglich sind.

```

Dim conSql As OleDbConnection = New OleDbConnection()
Dim Datenbank As String ➤
    = Server.MapPath(ConfigurationSettings.AppSettings("MDB"))
conSql.ConnectionString ➤
    = String.Format(ConfigurationSettings.AppSettings ➤
        ("OleDbConnection"), ➤
        Datenbank)

```

*Listing 9.5: Übernahme der Verbindungszeichenfolge aus web.config
(Ausschnitt aus OleDbConnectionCheckConfig.aspx.vb)*

Tipps! Hier wird die generelle Verbindungszeichenfolge unabhängig vom Datenbanknamen verwaltet. Ob eine derartige Aufsplittung notwendig ist, hängt von der Komplexität der Anwendung ab. Es ist nicht sinnvoll, jede Kleinigkeit einzeln konfigurierbar zu machen. Versuchen Sie, diejenigen Werte herauszulösen, die sich wirklich ändern können.

Ein anderer Grund für die Nutzung der Datei *web.config* besteht in der Möglichkeit, diese Werte auch dann ändern zu können, wenn die Applikation nur als Assembly weitergegeben wird. Kunden bekommen dann nicht den Quelltext zu Gesicht, können aber elementarste Konfigurationen selbst vornehmen. Vor allem bei Datei- und Pfadnamen ist diese Möglichkeit immer angebracht.

Verbindungsorientierte Ereignisse

Die Klassen `SqlConnection` und `OleDbConnection` kennen zwei praktisch bedeutsame Ereignisse, die beim Auftreten bestimmter Zustände ausgelöst werden. Zum einen kann der Wechsel des Status von offen nach geschlossen erfasst werden. Dazu dient das Ereignis `StateChange`. Zum anderen können Nachrichten des SQL Servers mit `InfoMessage` abgefangen werden. Die Ereignisbehandlungsmethoden sind vom Typ `StateChangeEventHandler` und `SqlInfoMessageEventHandler`.

**StateChange
InfoMessage**

Die Nutzung kann bei cleverem Einsatz die Programmierung etwas vereinfachen. Allerdings muss man sich darüber im Klaren sein, dass in ASP.NET die Ereignisverarbeitung nicht so einfach und flexibel einsetzbar ist, wie in der WinForms-Programmierung.

Nutzung

```
Public Class SqlEventHandling
    Inherits System.Web.UI.Page
    Protected Verbindung As Label
    Dim commSql As SqlCommand
    Dim conSql As SqlConnection

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection( _

ConfigurationSettings.AppSettings("SqlConnection"))
        AddHandler conSql.StateChange, _
            New StateChangeEventHandler(AddressOf _

Verbindung_OnChange)
        AddHandler conSql.InfoMessage, _
            New SqlInfoMessageEventHandler(AddressOf _
```

```

Information_OnEvent)
    conSql.Open()
    commSql = New SqlCommand("PRINT 'T-SQL kennt das " &
        PRINT-Kommando'", conSql)

    commSql.ExecuteNonQuery()
    conSql.Close()
End Sub

Public Sub Verbindung_OnChange(ByVal sender As Object,
    ByVal e As
    StateChangeEventArgs)
    Verbindung.Text &
        += String.Format("Verbindungsänderung von {0} nach
    {1}<br/>",
        e.OriginalState.ToString(),
    e.CurrentState.ToString())
End Sub

Public Sub Information_OnEvent(ByVal sender As Object,
    ByVal e As
    SqlInfoMessageEventArgs)
    Verbindung.Text &
        += String.Format("Neue Info eingetroffen: {0} von
    {1}<br/>",
        e.Message, e.Source)
End Sub
End Class

```

*Listing 9.6: Programmierung mit Ereignisbehandlungsmethoden
(SqlEventHandler.aspx.vb)*

Wie es funktioniert

Das Programm trennt die Reaktion auf die Ereignisse »Statusänderung« und »Information« völlig vom eigentlichen Ablauf. Zuerst sind die zwei Ereignisbehandlungsmethoden zu definieren. Dazu wird die Methode *Verbindung_OnChange* zur Behandlung der Statusänderung bestimmt:

```

AddHandler conSql.StateChange,
    New StateChangeEventHandler(AddressOf Verbindung_OnChange)

```

Dann wird *Information_OnEvent* zur Behandlung von Nachrichten erklärt:

```

AddHandler conSql.InfoMessage,
    New SqlInfoMessageEventHandler(AddressOf Information_OnEvent)

```

Der Rest des Hauptprogramms besteht nun darin, die Ereignisbehandlungsmethoden zu definieren. Der Kopf wird jeweils mit den üblichen Parametern »Sender« und »Ereignisargumente« ausgestattet:

```
Public Sub Verbindung_OnChange(ByVal sender As Object,   
                                ByVal e As StateChangeEventArgs)
```

Der Typ der Ereignisargumente für das Ereignis `StateChange` ist `StateChangeEventArgs`. Dieses Ereignisargument kennt folgende besondere Eigenschaften:

► `CurrentState`

Diese Eigenschaft enthält den neuen Status, der nach dem Wechsel gilt.

► `OriginalState`

In dieser Eigenschaft finden Sie den alten Status, der vor dem Wechsel galt.

Vergleichbar wird das zweite Ereignis definiert:

```
Public Sub Information_OnEvent(ByVal sender As Object,   
                                ByVal e As SqlInfoMessageEventArgs)
```

Der Typ der Ereignisargumente für das Ereignis `InfoMessage` ist `SqlInfoMessageEventArgs`. Auch dieses Ereignisargument kennt zwei spezielle Eigenschaften:

► `Message`

Diese Eigenschaft enthält die Nachricht, die der SQL Server gesendet hat. Im Fall des `PRINT`-Kommandos ist dies natürlich der Text des Kommandos.

► `Source`

In dieser Eigenschaft wird die Quelle der Nachricht ausgegeben. Normalerweise ist dies der Name des Providers.

Innerhalb der Ereignisbehandlungsmethoden werden lediglich die `Label`-Steuerelemente mit entsprechenden Ausgaben gefüllt. So ergibt sich bei der Ausgabe ein Abbild der Vorgänge (Abbildung 9.6).

SQL Ereignisbehandlung

Reaktion auf Ereignisse des Verbindungs-Objekts

Die folgenden Angaben erfolgen durch den Aufruf von Ereignisbehandlungsmethoden.

Verbindungsänderung von Closed nach Open

Neue Info eingetroffen: T-SQL kennt das PRINT-Kommando von .Net SqlClient Data Provider

Verbindungsänderung von Open nach Closed

Abbildung 9.6: Kontrolle des Verbindungsablaufes durch Ereignisse

Zusammenfassung

Mit diesen Informationen können Sie nun Ihre ASP.NET-Applikation mit jeder Datenbank verbinden. Im nächsten Schritt geht es darum, die Techniken des aktiven Zugriffs auf die Datenbank kennen zu lernen. Auch hier bietet ADO.NET ein reichhaltiges Repertoire an Klassen.

9.2.4 SQL-Befehle an die Datenbank senden

Wenn eine SQL-Datenbank vom Programm aus abgefragt oder gesteuert werden soll, muss es auch eine Methode geben, SQL-Befehle zu senden und die daraus entstehenden Ergebnisse auszuwerten. In ADO.NET gibt es dafür die Schnittstelle `IDbCommand`, die in den beiden Providern in den Klassen `OleDbCommand` und `SqlCommand` implementiert ist. Beide Klassen sind weitgehend identisch. Lediglich die Möglichkeit, Daten direkt als XML abzurufen, ist auf `SqlCommand` beschränkt, weil hier eine Eigenschaft des SQL Servers genutzt wird, die andere Datenbanken nicht bieten. Vergleichen Sie dazu die Informationen im Abschnitt »Abruf von Daten im XML-Format« ab Seite 422.

Kommando-Objekt erzeugen

Ein Kommando-Objekt wird erzeugt, indem der auszuführende Befehl und die zu verwendende Verbindung dem Konstruktor übergeben wird:

```
Dim commSql As SqlCommand = New SqlCommand(sCommand, oConnection)
```

Beide Parameter sind optional und die benötigten Angaben können auch über spezielle Eigenschaften zugewiesen werden. Die folgenden Beispiele zeigen bewusst verschiedene Überladungen des Konstruktors, die sich jedoch alle funktional gleichen.

Die Ausführungsmethoden

Existiert das Kommando-Objekt, kann eine Ausführungsmethode verwendet werden, um die Daten zur Datenbank zu senden und – wenn erforderlich – Ergebnisse abzurufen. Vier Methoden stehen zur Verfügung:

► `ExecuteNonQuery`

Diese Methode sendet einen Befehl an die Datenbank, erwartet aber kein Resultat (von einer Erfolgsmeldung abgesehen). Sie verwenden diese mit den Befehlen `DELETE`, `UPDATE`, `CREATE`, `DROP` usw.

► `ExecuteReader`

Mit dieser Methode wird normalerweise eine `SELECT`-Abfrage gesendet. Als Ergebnis wird ein Objekt vom Typ `DataReader` (je nach Provider als `SqlDataReader` oder `OleDbDataReader` implementiert) zurückgegeben. Sie finden im Abschnitt 9.2.5, »Datensätze lesen« ab Seite 846 mehr Informationen darüber, wie mit diesem Objekt umgegangen werden kann.

► `ExecuteScalar`

Manchmal ist es ausreichend, einen einzigen Wert aus der Datenbank abzurufen. Dann findet diese Methode Verwendung, die in der ersten Reihe das erste Feld liest, unabhängig davon, wie viele Daten die Abfrage ergab. Um den tatsächlich gewünschten Wert zu ermitteln, ist also normalerweise ein entsprechend gestalteter `SELECT`-Befehl nötig.

► `ExecuteXmlReader`

Diese Methode steht nur im Provider `SqlClient`, das heißt der Klasse `SqlCommand`, zur Verfügung. Er fragt den SQL Server ab und erwartet die Daten im XML-Format. Zur besseren Weiterverarbeitung wird ein `XmlReader`-Objekt zurückgegeben. Die Anwendung setzt voraus, dass der Befehl XML-Daten erzeugt, wozu `SELECT` in Verbindung mit `FOR XML` zum Einsatz kommt. Allerdings sind die `SqlXml`-Klassen des SQLXML 3.0-Paketes zum SQL Server für derartige Aufgaben besser geeignet.

Die folgenden Beispiele werden nur ausschnittsweise wiedergegeben. Sie finden die vollständigen Codes in den Projektdateien auf der CD oder auf der Website zum Buch. Alle Datenbankzugriffe basieren auf der in Listing 9.4 gezeigten Zugriffsmethode und setzen die Einstellung der richtigen Verbindungszeichenfolgen in der Datei `web.config` voraus.



Eine Anwendung der Methode `ExecuteNonQuery` wurde bereits im Listing 9.6 gezeigt. Die Abfrage eines skalaren Wertes zeigt das folgende kleine Programm. Es ermittelt die Anzahl der Datensätze der *Artikel*-Tabelle des Shops.

```
Public Class SqlCommandExScalar
    Inherits System.Web.UI.Page
    Protected Anzahl As Label
    Protected Fehler As Label

    Private conSql As SqlConnection
    Private commSql As SqlCommand

    Private Sub Page_Load(ByVal sender As Object,
                          ByVal e As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection()
        conSql.ConnectionString =
            ConfigurationSettings.AppSettings("SqlConnection")
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                commSql = New SqlCommand("SELECT COUNT(*)
^                                     FROM Artikel",
                                     conSql)
                Anzahl.Text = commSql.ExecuteScalar().ToString()
                conSql.Close()
            End If
            Catch ex As Exception
                Fehler.BackColor = Color.Red
                Fehler.Text = ex.Message
            End Try
        End Sub
    End Class
```

Listing 9.7: Lesen eines skalaren Wertes aus einer Datenbank
(*SqlCommandExScalar.aspx.vb*)

Der gesamte Vorgang besteht nur aus dem Erzeugen des Kommando-Objekts und der Ausführung der Methode `ExecuteScalar`. Diese Methode gibt den skalaren Wert immer als varianten Typ zurück, also als `Object`. Es ist daher notwendig (und immer möglich) in den benötigten Datentyp zu konvertieren. Im Fall der Übergabe an ein `Label`-Steuerelement wird die Umwandlung in eine Zeichenkette benötigt.

Kommando-Klasse

Lesen skalarer Werte

Die Tabelle *Artikel* enthält 4 Datensätze.

Abbildung 9.7: Ausgabe eines skalaren Wertes aus der Datenbankabfrage

Im Vergleich dazu soll auch die Abfrage von XML-Daten gezeigt werden. Der folgende Ausschnitt zeigt sowohl die Abfrage als auch den Umgang mit `XmlReader`:

XML abfragen
»FOR XML AUTO«

```
Public Class SqlCommandXmlReader
    Inherits System.Web.UI.Page
    Protected Ausgabe As HtmlGenericControl
    Protected Fehler As Label

    Private conSql As SqlConnection
    Private commSql As SqlCommand

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection()
        conSql.ConnectionString _
            = ConfigurationSettings.AppSettings("SqlConnection")
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                commSql = New SqlCommand("SELECT * FROM Artikel _
                    FOR XML AUTO", conSql)

                Dim xReader As XmlReader = commSql.ExecuteXmlReader()
                xReader.MoveToContent()
                While Not xReader.EOF
                    Ausgabe.InnerText += xReader.ReadOuterXml()
                End While
                conSql.Close()
            End If
        Catch e As Exception
            Fehler.BackColor = Color.Red
            Fehler.Text = e.Message
        End Try
    End Sub
End Class
```

Listing 9.8: Daten aus dem SQL Server im XML-Format abfragen
(*SqlCommandXmlReader.aspx*)

Die Ausgabe erfolgt wieder in ein HTML Server-Steuerelement (<div>-Tag), hier unter dem Namen *Ausgabe*. Eingesetzt wird neben dem entsprechenden SELECT-Kommando mit FOR XML auch die Methode `ExecuteXmlReader`. Der zurückgegebene `XmlReader` wird komplett durchlaufen. `ReadOuterXml` liest jeden Knoten vollständig und übergibt ihn an das Steuerelement. Die Eigenschaft `InnerText` stellt sicher, dass die XML-Tags im Browser auch angezeigt werden. Dies dient hier nur der Kontrolle, zeigt aber, wie unmittelbar nach der Abfrage auf die XML-Daten zugegriffen werden kann.

Kommando-Klasse

Lesen einer Tabelle im XML-Format

```
<Artikel id="1" name="Windows XP Pro" preis="199.99" nummer="1325"/><Artikel
id="2" name="MSDN Abo Professional" preis="2850.00" nummer="6668"/><Artikel
id="3" name="Windows XP Home" preis="139.99" nummer="1326"/><Artikel id="4"
name="Big Color Keyboard" preis="99.00" nummer="7738"/>
```

Abbildung 9.8: Ausgabe der Artikel-Tabelle im XML-Format

Eine weitere Option beim Einsatz des Kommando-Objekts ist die Steuerung gespeicherter Prozeduren. Dazu finden Sie im Abschnitt 9.2.6, »Detaillierte Informationen über eine Tabelle ermitteln« ab Seite 863 weitere Informationen. Vorerst steht die Abfrage der Datensätze mit *DataReader*, ausgelöst durch die Methode `ExecuteReader` im Vordergrund.

9.2.5 Datensätze lesen

Web-Applikationen erzeugen häufiger reine Lese- und weniger Schreibzugriffe als beispielsweise Windows-Programme. Das liegt daran, dass Daten einem breiten Publikum bereit gestellt und von diesem häufiger abgerufen werden, wenn die Site gut besucht ist. Es ist also nicht nur öfter der Fall, dass Lesevorgänge zu programmieren sind, sondern es kommt gerade hier auf beste Leistung an. Halten Sie sich dabei wieder das Beispiel eines Shopsystems vor Augen. Hier werden die meisten Kunden viel im Katalog blättern, die Angebote vergleichen und damit immer wieder Lesevorgänge auf Kategorie, Artikel und Zusatzinformationen auslösen. Nachwievor ist E-Commerce kein sicheres Geschäft. Von den vielen Besuchern werden nur sehr wenige zu Käufern. Und erst dann, wenn ein Kunde einen Artikel in den Warenkorb packt, erfolgt ein Schreibvorgang in eine Bestelltabelle. Aber auch bei anderen

Anwendungen ist die »Leselast« höher als die »Schreiblast«. Bei einer datenbankgestützten Authentifizierung erfolgt die Anmeldung nur einmal – in diesem Augenblick werden Anmeldedaten in die Datenbank geschrieben –; aber bei jedem Besuch werden die Anmeldedaten aus einem Formular mit dem Inhalt der Datenbank verglichen – dies sind Lesevorgänge.

Datensätze effizient mit *DataReader* lesen

Die Klassen *SqlDataReader* und *OleDbDataReader* erfüllen genau den beschriebenen Zweck: Sie lesen Daten. Aktualisierungen der Datenbank können Sie damit nicht vornehmen. Dafür realisieren die Klassen einen extrem schnellen Zugriff. Der *DataReader* ist auch in anderer Beziehung etwas Besonderes. Er verlangt nach einer bestehenden Verbindung zur Datenbank. Am Anfang wurde von »abgekoppelten« Datensätzen als Kernfunktion von ADO.NET gesprochen. Dieses Verhalten realisiert das *DataSet*, das in diesem Kapitel noch genauer betrachtet wird und weitaus komplexer ist.

Der *DataReader* ist insofern eine Ausnahme, denn die Datensätze werden nicht lokal gespeichert. Der Zugriff funktioniert nur in eine Richtung. Sie können mehrere Datensätze nur vorwärts lesen.

Nur vorwärts

Auch im Hinblick auf die Klassenhierarchie ist der *DataReader* exotisch. Er hat keine eindeutige Schnittstelle als Grundlage der Implementierung, sondern gleich zwei davon. Als Basis dienen *IDataReader* und *IDataRecord*. Es wird also zwischen dem Lesen und dem Speichern der Ergebnisse unterschieden. Die konkreten Implementierungen in den Klassen *SqlDataReader* und *OleDbDataReader* fassen die Schnittstellen immer zusammen, sodass eine getrennte Betrachtung wenig sinnvoll erscheint. Die Namen helfen aber auf der Suche nach speziellen Eigenschaften oder Methoden in der Online-Referenz.

Die Namensräume, die den *DataReader* beherbergen, sind dieselben wie für das Verbindungs-Objekt, da die Implementierungen Bestandteile des Providers sind. Vergleichen Sie dazu auch Abbildung 9.3:

Namensräume

```
Imports System.Data.SqlClient
```

```
Imports System.Data.OleDb
```

Ein *DataReader*-Objekt kann nicht explizit erzeugt werden. Es entsteht immer als Folge einer Kommandoausführung – sinnvoller-

weise eines `SELECT`-Befehls – gegenüber der Datenbank. Das folgende Beispiel zeigt, wie die *Artikel*-Tabellen ausgelesen werden können:

```
Try
    conSql.Open()
    If conSql.State = ConnectionState.Open Then
        commSql = New SqlCommand("SELECT * FROM Artikel", conSql)
        Dim dataReader As SqlDataReader = commSql.ExecuteReader()
        While dataReader.Read()
            Ausgabe.InnerHtml += dataReader.GetInt64(0).ToString() + ", "
            Ausgabe.InnerHtml += dataReader.GetString(1) + ", "
            Ausgabe.InnerHtml += dataReader.GetDecimal(2).ToString() +
            ", "
            Ausgabe.InnerHtml += dataReader.GetInt32(3).ToString() +
            ", "
            Ausgabe.InnerHtml += "<br />"
        End While
        dataReader.Close()
        conSql.Close()
    End If
```

Listing 9.9: Auslesen einer Tabelle mit dem `DataReader` (Ausschnitt aus dem Try-Zweig der Datei `SqlCommandExReader.aspx.vb`)

Wie es funktioniert

Der `SqlDataReader` verwaltet intern eine Tabelle mit den Ergebnissen, die sequenziell durchlaufen werden kann. Der nächste Datensatz wird mit `Read` ermittelt:

```
While dataReader.Read()
```



Die Methode `Read` muss vor dem ersten Abruf von Daten mindestens einmal aufgerufen werden, da der Datensatzzeiger sich am Anfang »vor« den Datensätzen befindet.

Nun stehen eine ganze Reihe von `Get`-Methoden zur Verfügung, um die Daten abzurufen. Dabei ist es wichtig, dass die zum jeweiligen Datensatz passende Methode verwendet wird. Im Beispiel hat die *Artikel*-Tabelle folgenden Aufbau:

	Spaltenname	Datentyp	Größe	NULL zulassen
	id	int	4	
	name	nvarchar	50	✓
	preis	numeric	9	✓
	nummer	int	4	✓

Daraus ergeben sich die verwendeten Methoden `GetInt32` für `Int`, `GetString` für `VarChar` und `GetDecimal` für `Numeric`. Obligatorisch ist die Konvertierung mit `ToString` für Ausgabe an das `Label`-Steuerelement.

Abschließend sollte nicht vergessen werden, auch einen `DataReader` zu schließen:

```
dataReader.Close()
```

Es ist wichtig, dass der `DataReader` geschlossen wird, denn er verwendet die Verbindung zur Datenbank exklusiv. Andere `DataReader` können erst zugreifen, wenn die Verbindung wieder freigegeben wird. Das betrifft nicht konkurrierende Zugriffe anderer Benutzer, sondern Ihre eigenen Zugriffsmöglichkeiten auf die aktuelle Verbindung.



In der Praxis ist der Zugriff auf die abgefragten Daten nicht immer so trivial, wie es bei einem einfachen `SELECT * FROM` der Fall ist. Tatsächlich bietet `DataReader` einiges mehr.

Kommando-Klasse

Lesen einer Tabelle mit `DataReader`

```
1, Windows XP Pro, 199, 99, 1325,  
2, MSDN Abo Professional, 2850, 6668,  
3, Windows XP Home, 139, 99, 1326,  
4, Big Color Keyboard, 99, 7738,
```

Abbildung 9.9: Ausgabe einer kompletten Tabelle mit `DataReader`

An dieser Stelle muss klar betont werden, dass die Ausgabe von Sequenzen an ein derart einfaches Steuerelement wie `Label` hier nur der Demonstration und Vereinfachung dient. Die vielen leistungsfähigen Daten-Steuerelemente in ASP.NET erlauben einen weit komfortableren Umgang mit Daten durch das Verfahren der Datenbindung. In Abschnitt 9.4, »Umgang mit Datenbanken in Visual Studio .NET« ab Seite 921 finden Sie dazu ausführliche Informationen.



Der DataReader im Detail

Schon beim Erzeugen des *DataReader*-Objekts können Sie mit verschiedenen Parametern das spätere Verhalten beeinflussen. Eine zweite Überladung der Methode `ExecuteReader` verarbeitet folgende Parameter, die aus der Aufzählung `CommandBehavior` stammen, also das Verhalten des Kommandos steuern:

Wert	Bedeutung
<code>CloseConnection</code>	Schließt die Verbindung, wenn der <i>DataReader</i> geschlossen wird
<code>KeyInfo</code>	Informationen über die Spalten und den Primärschlüssel werden zurückgegeben. Dies unterdrückt möglicherweise die Daten.
<code>SchemaOnly</code>	Es werden nur Spalteninformationen ermittelt, aber keine Kommandos ausgeführt.
<code>SequentialAccess</code>	Spalten werden sequenziell gelesen. Das ist interessant, wenn große Mengen an Binärdaten geholt werden, die den Ergebnissatz aufblähen.
<code>SingleResult</code>	Es wird nur ein Feld geholt.
<code>SingleRow</code>	Es wird nur eine Ergebniszeile geholt.

Tabelle 9.1: CommandBehavior-Parameter der Methode ExecuteReader

Sie können mehrere dieser Parameter kombinieren, indem Sie sie mit ODER-Verknüpfen:

```
CommandBehavior.CloseConnection or CommandBehavior.SingleRow
```

Der Zugriff auf die Tabelle kann auf mehreren Wegen erfolgen. Im letzten Beispiel wurde die Spaltennummer, nullbasierend gezählt, eingesetzt. Außerdem kann der Name der Spalte verwendet werden; die entsprechende Methode `Item` muss nicht geschrieben werden, da es sich um die Standardeigenschaft des Objekts handelt:

```
dataReader("preis")
```

Ergebnisinfor-
mationen dynamisch
ermitteln

Es werden Ihnen immer wieder Fälle begegnen, in denen eine präzise Vorhersage über die Anzahl der Felder, deren Datentypen und dem möglicherweise vorhandenen Inhalt nicht möglich ist. Das folgende Beispiel zeigt, wie mit Hilfe der Implementierung der *DataReader*-Klasse eine beliebige Tabelle ausgelesen wird. In der *aspx*-Datei (*SqlCommandExReader2.aspx*) ist lediglich ein `PlaceHolder`-Steuerelement platziert, das nun mit einer HTML-Tabelle gefüllt wird:

```
<asp:Placeholder Runat="server" ID="Tabelle"/>
```

Vergleichen Sie das auch mit den im Abschnitt 9.4, »Umgang mit Datenbanken in Visual Studio .NET« ab Seite 921 gezeigten Daten-Steuerelementen. Die Anwendung der Elemente `Repeater` oder `DataGrid` erlaubt leistungsfähigere Darstellungen, dagegen ist die »selbst gebaute« Tabelle im Detail flexibler.

```
Imports System.Data
Imports System.Data.SqlClient

Public Class SqlCommandExReader2
    Inherits System.Web.UI.Page
    Protected Tabelle As Placeholder
    Protected Fehler As Label

    Private conSql As SqlConnection
    Private commSql As SqlCommand

    Private t As Table
    Private tr As TableRow
    Private tc As TableCell

    Private Sub Page_Load(ByVal sender As Object, _
                          ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection()
        conSql.ConnectionString =
            ConfigurationSettings.AppSettings("SqlConnection")
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                commSql = New SqlCommand("SELECT * FROM Artikel", _
                                         conSql)
                Dim dataReader As SqlDataReader
                =
            commSql.ExecuteReader(CommandBehavior.CloseConnection)
            t = New Table()
            t.BorderStyle = BorderStyle.Solid
            t.GridLines = GridLines.Both
            tr = New TableRow()
            Dim i As Integer
            For i = 0 To dataReader.FieldCount - 1 Step 1
                tc = New TableCell()
                tc.BackColor = Color.Silver
                tc.Text = dataReader.GetName(i).ToUpper()
                tr.Cells.Add(tc)
            Next i
        Catch ex As Exception
            Fehler.Text = ex.Message
        End Try
    End Sub
End Class
```

```

Next
t.Rows.Add(tr)
Dim field As Object
While dataReader.Read()
    tr = New TableRow()
    Dim j As Integer
    For j = 0 To dataReader.FieldCount - 1
        tc = New TableCell()
        field = dataReader.GetValue(j)
        tc.Text = field
        tr.Cells.Add(tc)
    Next
    t.Rows.Add(tr)
End While
Tabelle.Controls.Add(t)
dataReader.Close()
conSql.Close()
End If
Catch e As Exception
    Fehler.BackColor = Color.Red
    Fehler.Text = e.Message & "<br/>" & e.StackTrace
End Try
End Sub
End Class

```

Listing 9.10: Ausgabe einer beliebigen Tabelle des SQL Servers in HTML (SqlCommandExReader2.aspx.vb)

Wie es funktioniert

Die Abfrage der Tabelle erfolgt wieder in der bereits gezeigten Weise. Zusätzlich wird bei der Ausführung von `ExecuteReader` ein Kommando-Parameter gesetzt, mit dem die unterliegende Verbindung beim Schließen des *DataReader* auch geschlossen wird:

```
CommandBehavior.CloseConnection
```

Damit existiert das Abfrageergebnis bereits. Es muss nun in eine HTML-Tabelle platziert werden. Dazu wird zuerst eine neue Tabelle erzeugt:

```
t = New Table()
```

FieldCount

Es folgen einige Gestaltungseigenschaften, die hier nicht weiter betrachtet werden sollen. Nun muss die Kopfreihe erzeugt werden:

```
tr = New TableRow()
```

Die Feldnamen sollen denen der Definition der Tabellen entsprechen und in Großbuchstaben angezeigt werden. Dazu ist zuerst die Anzahl der Felder mit der Eigenschaft `FieldCount` zu ermitteln:


```
For i = 0 To dataReader.FieldCount - 1
```

Es wird jetzt für jedes Feld eine Zelle erstellt und gestaltet. Der Inhalt der Zelle wird dann mit der Eigenschaft `Text` auf den Feldnamen gesetzt:

GetName

```
tc.Text = dataReader.GetName(i).ToUpper()
```

Die Methode `GetName` gibt den Namen der angegebenen Spalte zurück.

Nach dem Anhängen der Reihe an die Kollektion werden nun alle Datensätze durchlaufen und feldweise in Zellen ausgegeben:

```
While dataReader.Read()
```

Beliebigen Inhalt ermitteln Sie am besten mit `GetValue`. Diese Methode gibt immer den Typ `Object` zurück:

GetValue

```
field = dataReader.GetValue(i)
```

Die Konvertierung und die Zuweisung an den Zellentext kann implizit erfolgen:

```
tc.Text = field
```

Zuletzt wird noch die fertige Reihe an die Reihen-Kollektion der Tabelle angehängt:

```
t.Rows.Add(tr)
```

Der letzte Schritt besteht nun im Einbau der fertigen Tabelle in den Platzhalter:

```
Tabelle.Controls.Add(t)
```

Die Abbildung zeigt, wie dieses Programm arbeitet, wenn die *Artikel*-Tabelle abgefragt wird:

Kommando-Klasse

Universelles Lesen mit DataReader

ID	NAME	PREIS	NUMMER
1	Windows XP Pro	199,99	1325
2	MSDN Abo Professional	2850	6668
3	Windows XP Home	139,99	1326
4	Big Color Keyboard	99	7738

Abbildung 9.10: Ausgabe einer beliebigen Tabelle mit DataReader

Eine andere Möglichkeit des Abrufes der Daten aus *DataReader* soll noch gezeigt werden. So können Sie mit der Methode *GetValues* ein Array der Elemente des aktuellen Datensatzes erhalten. Der folgende Ausschnitt aus dem Programm *SqlCommandExReader2.aspx* ist dort als auskommentierte »Version 2« enthalten. Das Ergebnis entspricht exakt dem in Abbildung 9.10 gezeigten. Es ersetzt den Inhalt der bisherigen *While*-Schleife:

```
Dim dataRows As ArrayList = New ArrayList()
Dim fields() As Object
While dataReader.Read()
    fields = New Object(dataReader.FieldCount) {}
    dataReader.GetValues(fields)
    dataRows.Add(fields)
End While
Dim row() As Object
For Each row In dataRows
    tr = New TableRow()
    Dim field As Object
    For Each field In row
        If field Is Nothing Then
            Exit For
        End If
        tc = New TableCell()
        tc.Text = field
        tr.Cells.Add(tc)
    Next
    t.Rows.Add(tr)
Next
```

Listing 9.11: Version 2 des universellen Abrufs (Ausschnitt aus SqlCommandExReader2.aspx.vb)

Hier wird nun für jeden Datensatz der Abfrage ein Array vom Typ *Object* erzeugt:

```
fields = New Object(dataReader.FieldCount) {}
```

GetValues Diesem Array werden dann die Werte zugewiesen:

```
dataReader.GetValues(fields)
```

Damit die Reihen später verarbeitet werden können, werden Sie als Elemente des *ArrayList*-Objekts *dataRows* gespeichert.

```
dataRows.Add(fields)
```

Die Umsetzung in die Tabelle kann nun mit *For Each* erfolgen. Da es sich um ein Array aus Arrays handelt, sind zwei derartige Schleifen verschachtelt.

Auf den ersten Blick erscheint die zweite Version komplizierter. Warum also sollten Sie das tun? Am Anfang wurde kurz erwähnt, dass der Zugriff eines *DataReader* die Verbindung zur Datenbank blockiert. Bei hoher Last kann es kritisch sein, die Verbindung lange zu okkupieren. Programme sollten so geschrieben werden, dass die *DataReader* möglichst schnell geschlossen werden. Im Beispiel wird dies erreicht, indem diese Methode sofort nach der kleinen *While*-Schleife steht:

```
dataReader.Close()
```

Die Ausgabe der Tabelle erscheint zwar kompakt, ist aber nicht sehr effizient, denn in der Praxis kommen viele Gestaltungseigenschaften, Berechnungen oder weitere Steuerelemente hinzu, so dass das Laufzeitverhalten ungünstig wird. Durch die Verlagerung der Abfrage in ein Array können Sie die Verbindung unabhängig vom Laufzeitverhalten der restlichen Applikation schnell wieder freigeben.

Vorteil: Laufzeitverhalten

Nachteilig bei der Nutzung des *Object*-Arrays ist der Verlust der Datentypen. In diesem konkreten Fall ist das Ausgabeziel sowieso nur eine Zeichenkette; deswegen war die Wahl des Datentyps *Object* vertretbar. Soll dagegen eine individuelle Weiterverarbeitung der Werte erfolgen, beispielsweise für Berechnungen mit den Preisen, ist das nicht so günstig. Sie können den ursprünglichen Typ zwar jederzeit mit *GetType* ermitteln, aber derartige Reflexion-Methoden sind nicht besonders schnell und im Detail auch nicht immer einfach zu programmieren.

Nachteil: Verlust der Datentypen

Welche Methode Sie verwenden, hängt von der Art der Weiterverarbeitung und der zu erwartenden Last ab. Gegebenenfalls müssen beide Varianten programmiert und dann unter realen Bedingungen verglichen werden.

Typsicherer Zugriff auf Daten

Wenn es um den typsicheren Zugriff geht, wurden im letzten Abschnitt bereits mögliche Probleme genannt. Es gibt natürlich eine Lösung für den typsicheren Zugriff. Sicher kennen Sie noch aus dem ersten Beispiel in Listing 9.9 die Methoden, die gezielt einen bestimmten Datentyp ermitteln, *GetInt32* usw. Nachteilig ist, dass Sie hier immer Spaltennummern angeben müssen, um das betreffende Feld auszulesen. Es ist aber dringend zu empfehlen, Spal-

GetOrdinal

tennamen zu verwenden, denn die Reihenfolge kann sich durchaus ändern. Es ist deshalb möglich, die aktuelle Ordnungszahl einer Spalte auf Basis des Namens zu ermitteln:

```
Public Class SqlCommandGetOrdinal
    Inherits System.Web.UI.Page
    Protected Ausgabe As HtmlGenericControl
    Protected Fehler As Label

    Private conSql As SqlConnection
    Private commSql As SqlCommand

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection()
        conSql.ConnectionString _
            = ConfigurationSettings.AppSettings("SqlConnection")
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                Dim sr As StringBuilder = New StringBuilder()
                commSql = New SqlCommand("SELECT * FROM Artikel", conSql)
                Dim dataReader As SqlDataReader = commSql.ExecuteReader()
                Dim iID As Integer = dataReader.GetOrdinal("id")
                Dim iName As Integer = dataReader.GetOrdinal("name")
                Dim iPreis As Integer = dataReader.GetOrdinal("preis")
                Dim iNummer As Integer = dataReader.GetOrdinal("nummer")
                While dataReader.Read()
                    sr.AppendFormat("ID: {0,5}<br/>", _
                        dataReader.GetInt64(iID).ToString())
                    sr.AppendFormat("Name: {0}<br/>", _
                        dataReader.GetString(iName))
                    sr.AppendFormat("Preis: {0}<br/>", _
                        dataReader.GetDecimal(iPreis).ToString())
                    sr.AppendFormat("Nummer: {0}<br/>", _
                        dataReader.GetInt32(iNummer).ToString())
                End While
                Ausgabe.InnerHtml += sr.ToString()
                dataReader.Close()
                conSql.Close()
            End If
        Catch e As Exception
            Fehler.BackColor = Color.Red
        End Try
    End Sub
End Class
```

```
Fehler.Text = e.Message  
End Try  
End Sub  
End Class
```

Listing 9.12: Zugriff auf Spalten über Spaltennamen auch an den Stellen, wo Spaltennummern erwartet werden (SqlCommand.GetOrdinal.aspx.vb)

Dieses Programm ermittelt zuerst die Spaltennummern mit Hilfe der Methode `GetOrdinal` aus den Spaltennamen:

Wie es funktioniert

```
Dim iID As Integer = dataReader.GetOrdinal("id")
```

Diese Nummer wird dann für nachfolgende Zugriffe verwendet. Der außerdem eingesetzte `StringBuilder` zeigt nur eine andere Art der Zusammensetzung von längeren Zeichenketten. Mehr Informationen darüber finden Sie im Abschnitt »Die Klassen `StringWriter` und `StringBuilder` im Detail« ab Seite 262. Das Ergebnis zeigt alle Datensätze der Tabelle *Artikel*:

Kommando-Klasse

DataReader und GetOrdinal

ID: 1
Name: Windows XP Pro
Preis: 199,99
Nummer: 1325
ID: 2
Name: MSDN Abo Professional
Preis: 2850
Nummer: 6668
ID: 3
Name: Windows XP Home
Preis: 139,99
Nummer: 1326
ID: 4
Name: Big Color Keyboard
Preis: 99
Nummer: 7738

Abbildung 9.11: Datensätze, ermittelt mit Hilfe von `GetOrdinal`

Diese Methode ist eher als vorbereitende Maßnahme für eine weitere Technik zu verstehen. Wenn Sie mit dem SQL Server 2000 arbeiten, werden Sie sicher bemerkt haben, dass die SQL-Datentypen nicht denen im CTS entsprechen. Beide Typwelten sind

zwar reichhaltig ausgestattet und können gut harmonisieren, im Sinne eines guten Programmierstils ist jedoch eine weitere Festigung des Typkonzepts besser.

Zu diesem Zweck gibt es einen Namensraum mit Datentypen speziell für den SQL Server:

`System.Data.SqlTypes`

Damit können Sie nun direkt mit den Typen arbeiten, die auch zur Definition der Tabellen verwendet werden. So kennt der OleDb-Treiber keinen Datentyp »Money«. Die Verarbeitung als numerischer Wert ist zwar problemlos, aber in komplexeren Programmen könnte der Verlust der exakten Zuordnung stören, weil an anderer Stelle nicht mehr erkennbar ist, dass es sich nicht nur um eine Zahl, sondern einen Währungswert handelt.

In diesem Sinne soll das Programm aus Listing 9.12 nochmals verändert werden, diesmal im Hinblick auf die Datentypen:

```
Public Class SqlCommandGetSqlTypes
    Inherits System.Web.UI.Page
    Protected Ausgabe As HtmlGenericControl
    Protected Fehler As Label

    Private conSql As SqlConnection
    Private commSql As SqlCommand

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection()
        conSql.ConnectionString = _
            = ConfigurationSettings.AppSettings("SqlConnection")
        Dim sQuery As String
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                Dim sr As StringBuilder = New StringBuilder()
                sQuery = "SELECT b.[a-id], " _
                    + "    SUM(b.menge) AS menge, " _
                    + "    SUM(b.summe) AS summe, " _
                    + "    a.name AS name " _
                    + " FROM Bestellungen b " _
                    + " JOIN Artikel a ON a.id = b.[a-id] " _
                    + " GROUP BY b.[a-id], a.name"
```

```

        commSql = New SqlCommand(sQuery, conSql)
        Dim dataReader As SqlDataReader =
commSql.ExecuteReader()
        Dim iID As Integer = dataReader.GetOrdinal("a-id")
        Dim iMenge As Integer =
dataReader.GetOrdinal("menge")
        Dim iPreis As Integer =
dataReader.GetOrdinal("summe")
        Dim iName As Integer = dataReader.GetOrdinal("name")
        While dataReader.Read()
            Dim oID As SqlInt64 = dataReader.GetSqlInt64(iID)
            Dim oMenge As SqlDouble =
dataReader.GetSqlDouble(iMenge)
            Dim oPreis As SqlMoney =
dataReader.GetSqlMoney(iPreis)
            Dim oName As SqlString =
dataReader.GetSqlString(iName)
            sr.AppendFormat("ID: {0}<br/>", oID)
            sr.AppendFormat("Nummer: {0}<br/>", oMenge)
            sr.AppendFormat("Preis: {0:F2}<br/>",
oPreis.ToDouble())
            sr.AppendFormat("Name: {0}<br/>", oName)
        End While
        Ausgabe.InnerHtml += sr.ToString()
        dataReader.Close()
        conSql.Close()
    End If
Catch e As Exception
    Fehler.BackColor = Color.Red
    Fehler.Text = e.Message & "<br/>" & sQuery
End Try
End Sub
End Class

```

Listing 9.13: Direkte Verarbeitung von SQL-Datentypen in .NET (SqlCommandGetSqlTypes.aspx.vb)

Hier wird eine komplexere Abfrage verwendet. Die SQL-Anweisung ermittelt eine Liste der bestellten Artikel, einschließlich einer Summe der Bestellwerte und der Artikelnamen. Die Spaltennummern werden wie bisher auch ermittelt:

```
Dim iID As Integer = dataReader.GetOrdinal("a-id")
```

Innerhalb der Schleife wird nun direkt auf die SQL Server-Datentypen zugegriffen:

```
Dim oPreis As SqlMoney = dataReader.GetSqlMoney(iPreis)
```

Wie es funktioniert

Die Variable `oPreis` ist nun »typrein«, enthält also den Money-Typ des SQL-Servers, der kein Äquivalent im CTS kennt. Bei der weiteren Verarbeitung können Sie unabhängig davon die benötigten Umwandlungen vornehmen. Die Typklassen sind hier ausreichend ausgestattet, hier zur Umwandlung nach `Double`, damit die Formatierung gelingt:

```
sr.AppendFormat("Preis: {0:F2}<br/>", oPreis.ToDouble())
```

Die Umwandlung mit den To-Methoden ist dabei sowohl zwischen SQL-Typen als auch hin zu CTS-Typen möglich.

Kommando-Klasse	
DataReader und SqlTypes	
ID: 4	
Nummer: 2	
Preis: 198,00	
Name: Big Color Keyboard	
ID: 2	
Nummer: 1	
Preis: 2850,00	
Name: MSDN Abo Professional	
ID: 1	
Nummer: 1	
Preis: 199,99	
Name: Windows XP Pro	

Abbildung 9.12: Ausgabe der Abfrage aus Listing 9.13

Behandlung komplexer Abfragen mit DataReader

Nicht immer sind es einfache `SELECT`-Befehle, die an die Datenbank gesendet werden. T-SQL erlaubt eine effektive Stapelverarbeitung. Es kann vorkommen, dass dann mehrere Tabellen im *DataReader* stecken. Betrachten Sie das folgende T-SQL-Programm:

```
SELECT * FROM Artikel;
SELECT * FROM Bestellungen;
```



Wenn Sie mehrere Befehle in T-SQL zusammenfassen, müssen Sie jeden mit einem Semikolon abschließen.

Zwangsläufig entstehen hier zwei verschiedene Tabellen:

	id	name	preis	nummer
1	1	Windows XP Pro	199.99	1325
2	2	MSDN Abo Professional	2850.00	6668
3	3	Windows XP Home	139.99	1326
4	4	Big Color Keyboard	99.00	7738

	id	k-id	a-id	menge	summe
1	21	3	4	1.0	99.00
2	67	1	2	1.0	2850.00
3	72	1	4	1.0	99.00
4	73	1	1	1.0	199.99

Abbildung 9.13: Die Ausführung des kleinen T-SQL-Programms im Query Analyzer

Um an die Effizienz-Diskussion des letzten Abschnitts anzuknüpfen: Eine Verbindung, eine Abfrage und eine verzögerte Auswertung sind manchmal empfehlenswerter als mehrere direkt verwendete *DataReader* einzusetzen. Betrachten Sie die Beispiele daher als eine mögliche Variante und nicht als Vorschrift.

Die Darstellung in HTML sollte nun folgendermaßen aussehen:

Kommando-Klasse

Lesen mehrerer Tabellen mit DataReader

ID	NAME	PREIS	NUMMER
1	Windows XP Pro	199,99	1325
2	MSDN Abo Professional	2850	6668
3	Windows XP Home	139,99	1326
4	Big Color Keyboard	99	7738

ID	K-ID	A-ID	MENGE	SUMME
21	3	4	1	99
67	1	2	1	2850
72	1	4	1	99
73	1	1	1	199,99

Abbildung 9.14: Ausgabe mehrerer Tabellen aus einem DataReader

Das fertige Programm finden Sie in *SqlCommandMultiReader.aspx*. Es unterscheidet sich kaum vom bereits gezeigten Prinzip, lediglich eine zusätzliche *While*-Schleife kam hinzu:

```
Private Sub Page_Load(ByVal sender As Object,   
                        ByVal eArgs As System.EventArgs)   
Handles MyBase.Load   
    conSql = New SqlConnection(  
        conSql.ConnectionString
```

NextResult

```

        = ConfigurationSettings.AppSettings("SqlConnection")
Try
    conSql.Open()
    If conSql.State = ConnectionState.Open Then
        Dim sr As StringBuilder = New StringBuilder()
        sr.Append("SELECT * FROM Artikel;")
        sr.Append("SELECT * FROM Bestellungen;")
        commSql = New SqlCommand(sr.ToString(), conSql)
        Dim dataReader As SqlDataReader ←
            = commSql.ExecuteReader(CommandBehavior.CloseConnection)
        Do
            t = New Table()
            t.BorderStyle = BorderStyle.Solid
            t.GridLines = GridLines.Both
            t.Rows.Add(HeaderRow(dataReader))
            Dim dataRows As ArrayList = New ArrayList()
            Dim fields() As Object
            While dataReader.Read()
                fields = New Object(dataReader.FieldCount) {}
                dataReader.GetValues(fields)
                dataRows.Add(fields)
            End While
            Dim row() As Object
            For Each row In dataRows
                t.Rows.Add(GetRow(row))
            Next
            Tabelle.Controls.Add(t)
        Loop While dataReader.NextResult()
        dataReader.Close()
        conSql.Close()
    End If
Catch e As Exception
    Fehler.BackColor = Color.Red
    Fehler.Text = e.Message & e.StackTrace
End Try
End Sub

```

Listing 9.14: Zugriff auf mehrere Tabellen bei Abfragen einer T-SQL-Stapelverarbeitungsdatei (Ausschnitt aus SqlCommandMultiReader.aspx.vb)

Die entscheidende Zeile findet sich am Schluss der Schleife:

```
Loop While dataReader.NextResult()
```

Hier wird nach jedem Durchlauf zum nächsten Datensatz weitergeschaltet.

**Die weiteren
Methoden der
Klasse**

Innerhalb dieser Schleife wird dann wie bisher jede einzelne Tabelle erzeugt. Das sieht etwas überschaubarer aus als bei den bis-

herigen Beispielen, weil die Erzeugung der Reihen in zwei Methoden ausgelagert wurde. *HeaderRow* erzeugt die Kopfreihe:

```
Private Function HeaderRow(ByRef dataReader As SqlDataReader) As TableRow
    tr = New TableRow()
    Dim i As Integer
    For i = 0 To dataReader.FieldCount - 1 Step i + 1
        tc = New TableCell()
        tc.BackColor = Color.Silver
        tc.Text = dataReader.GetName(i).ToUpper()
        tr.Cells.Add(tc)
    Next
    Return tr
End Function
```

Listing 9.15: Zugriff auf mehrere Tabellen bei Abfragen einer T-SQL-Stapelverarbeitungsdatei (Fortsetzung von `SqlCommandMultiReader.aspx.vb`)

Mit der Methode *GetRow* werden dann die einzelnen Zeilen generiert:

```
Private Function GetRow(ByVal row() As Object) As TableRow
    tr = New TableRow()
    Dim field As Object
    For Each field In row
        If field Is Nothing Then
            Exit For
        End If
        tc = New TableCell()
        tc.Text = field
        tr.Cells.Add(tc)
    Next
    Return tr
End Function
```

Listing 9.16: Zugriff auf mehrere Tabellen bei Abfragen einer T-SQL-Stapelverarbeitungsdatei (Fortsetzung von `SqlCommandMultiReader.aspx.vb`)

9.2.6 Detaillierte Informationen über eine Tabelle ermitteln

Manchmal wird mehr benötigt als nur der Name einer Spalte. Wenn Sie eine Datenbankverwaltung ähnlich dem Enterprise Manager für Ihre Website planen, ist ein Blick auf die Methode *GetSchemaTable* des *DataReaders* angebracht. Das folgende Listing zeigt die Anwendung:

```

Public Class SqlCommandGetSchema
    Inherits System.Web.UI.Page
    Protected Schema As DataGrid
    Protected Fehler As Label

    Private conSql As SqlConnection
    Private dataReader As SqlDataReader
    Private commSql As SqlCommand

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Load
        conSql = New SqlConnection()
        conSql.ConnectionString _
            = ConfigurationSettings.AppSettings("SqlConnection")
        Try
            conSql.Open()
            If conSql.State = ConnectionState.Open Then
                commSql = New SqlCommand("SELECT * FROM Artikel", conSql)
                dataReader = commSql.ExecuteReader(_
                    CommandBehavior.SchemaOnly)
                Schema.DataSource = dataReader.GetSchemaTable()
                Schema.DataBind()
                dataReader.Close()
                conSql.Close()
            End If
        Catch e As Exception
            Fehler.BackColor = Color.Red
            Fehler.Text = e.Message
        End Try
    End Sub
End Class

```

Listing 9.17: Abruf der gesamten Tabellendefinition (SqlCommandGetSchema.aspx.vb)

Wie es funktioniert

Hier erfolgt, wie bei den anderen Beispielen auch, der Abruf einer kompletten Tabelle:

```
commSql = New SqlCommand("SELECT * FROM Artikel", conSql)
```

Bei der Ausführung wird der Vorgang jedoch auf die Ermittlung des Schemas der Tabelle beschränkt:

```
dataReader = commSql.ExecuteReader(CommandBehavior.SchemaOnly)
```

In der *aspx*-Datei wurde ein `DataGrid` definiert. Dies ist hier die beste Methode der Ausgabe und wird im Vorgriff auf die Daten-Steuererelemente verwendet:

```
<asp:DataGrid Runat="server" ID="Schema" />
```

Im Programm wird dem Steuerelement das Schema als Daten-

GetSchemaTable

quelle zugewiesen:

```
Schema.DataSource = dataReader.GetSchemaTable()
```

Mit der Ausführung der Datenbindung erscheint das Ergebnis:

```
Schema.DataBind()
```

Das Ergebnis ist, gemessen am Aufwand, durchaus beeindruckend:

Kommando-Klasse	
Schema mit DataReader ermitteln	
ColumnsName	ColumnsOrdinal
id	0
name	1
price	2
quantity	3
id	4
name	5
price	6
quantity	7
id	8
name	9
price	10
quantity	11
id	12
name	13
price	14
quantity	15
id	16
name	17
price	18
quantity	19
id	20
name	21
price	22
quantity	23
id	24
name	25
price	26
quantity	27
id	28
name	29
price	30
quantity	31
id	32
name	33
price	34
quantity	35
id	36
name	37
price	38
quantity	39
id	40
name	41
price	42
quantity	43
id	44
name	45
price	46
quantity	47
id	48
name	49
price	50
quantity	51
id	52
name	53
price	54
quantity	55
id	56
name	57
price	58
quantity	59
id	60
name	61
price	62
quantity	63
id	64
name	65
price	66
quantity	67
id	68
name	69
price	70
quantity	71
id	72
name	73
price	74
quantity	75
id	76
name	77
price	78
quantity	79
id	80
name	81
price	82
quantity	83
id	84
name	85
price	86
quantity	87
id	88
name	89
price	90
quantity	91
id	92
name	93
price	94
quantity	95
id	96
name	97
price	98
quantity	99
id	100
name	101
price	102
quantity	103
id	104
name	105
price	106
quantity	107
id	108
name	109
price	110
quantity	111
id	112
name	113
price	114
quantity	115
id	116
name	117
price	118
quantity	119
id	120
name	121
price	122
quantity	123
id	124
name	125
price	126
quantity	127
id	128
name	129
price	130
quantity	131
id	132
name	133
price	134
quantity	135
id	136
name	137
price	138
quantity	139
id	140
name	141
price	142
quantity	143
id	144
name	145
price	146
quantity	147
id	148
name	149
price	150
quantity	151
id	152
name	153
price	154
quantity	155
id	156
name	157
price	158
quantity	159
id	160
name	161
price	162
quantity	163
id	164
name	165
price	166
quantity	167
id	168
name	169
price	170
quantity	171
id	172
name	173
price	174
quantity	175
id	176
name	177
price	178
quantity	179
id	180
name	181
price	182
quantity	183
id	184
name	185
price	186
quantity	187
id	188
name	189
price	190
quantity	191
id	192
name	193
price	194
quantity	195
id	196
name	197
price	198
quantity	199
id	200
name	201
price	202
quantity	203
id	204
name	205
price	206
quantity	207
id	208
name	209
price	210
quantity	211
id	212
name	213
price	214
quantity	215
id	216
name	217
price	218
quantity	219
id	220
name	221
price	222
quantity	223
id	224
name	225
price	226
quantity	227
id	228
name	229
price	230
quantity	231
id	232
name	233
price	234
quantity	235
id	236
name	237
price	238
quantity	239
id	240
name	241
price	242
quantity	243
id	244
name	245
price	246
quantity	247
id	248
name	249
price	250
quantity	251
id	252
name	253
price	254
quantity	255
id	256
name	257
price	258
quantity	259
id	260
name	261
price	262
quantity	263
id	264
name	265
price	266
quantity	267
id	268
name	269
price	270
quantity	271
id	272
name	273
price	274
quantity	275
id	276
name	277
price	278
quantity	279
id	280
name	281
price	282
quantity	283
id	284
name	285
price	286
quantity	287
id	288
name	289
price	290
quantity	291
id	292
name	293
price	294
quantity	295
id	296
name	297
price	298
quantity	299
id	300
name	301
price	302
quantity	303
id	304
name	305
price	306
quantity	307
id	308
name	309
price	310
quantity	311
id	312
name	313
price	314
quantity	315
id	316
name	317
price	318
quantity	319
id	320
name	321
price	322
quantity	323
id	324
name	325
price	326
quantity	327
id	328
name	329
price	330
quantity	331
id	332
name	333
price	334
quantity	335
id	336
name	337
price	338
quantity	339
id	340
name	341
price	342
quantity	343
id	344
name	345
price	346
quantity	347
id	348
name	349
price	350
quantity	351
id	352
name	353
price	354
quantity	355
id	356
name	357
price	358
quantity	359
id	360
name	361
price	362
quantity	363
id	364
name	365
price	366
quantity	367
id	368
name	369
price	370
quantity	371
id	372
name	373
price	374
quantity	375
id	376
name	377
price	378
quantity	379
id	380
name	381
price	382
quantity	383
id	384
name	385
price	386
quantity	387
id	388
name	389
price	390
quantity	391
id	392
name	393
price	394
quantity	395
id	396
name	397
price	398
quantity	399
id	400
name	401
price	402
quantity	403
id	404
name	405
price	406
quantity	407
id	408
name	409
price	410
quantity	411
id	412
name	413
price	414
quantity	415
id	416
name	417
price	418
quantity	419
id	420
name	421
price	422
quantity	423
id	424
name	425
price	426
quantity	427
id	428
name	429
price	430
quantity	431
id	432
name	433
price	434
quantity	435
id	436
name	437
price	438
quantity	439
id	440
name	441
price	442
quantity	443
id	444
name	445
price	446
quantity	447
id	448
name	449
price	450
quantity	451
id	452
name	453
price	454
quantity	455
id	456
name	457
price	458
quantity	459
id	460
name	461
price	462
quantity	463
id	464
name	465
price	466
quantity	467
id	468
name	469
price	470
quantity	471
id	472
name	473
price	474
quantity	475
id	476
name	477
price	478
quantity	479
id	480
name	481
price	482
quantity	483
id	484
name	485
price	486
quantity	487
id	488
name	489
price	490
quantity	491
id	492
name	493
price	494
quantity	495
id	496
name	497
price	498
quantity	499
id	500
name	501
price	502
quantity	503
id	504
name	505
price	506
quantity	507
id	508
name	509
price	510
quantity	511
id	512
name	513
price	514
quantity	515
id	516
name	517
price	518
quantity	519
id	520
name	521
price	522
quantity	523
id	524
name	525
price	526
quantity	527
id	528
name	529
price	530
quantity	531
id	532
name	533
price	534
quantity	535
id	536
name	537
price	538
quantity	539
id	540
name	541
price	542
quantity	543
id	544
name	545
price	546
quantity	547
id	548
name	549
price	550
quantity	551
id	552
name	553
price	554
quantity	555
id	556
name	557
price	558
quantity	559
id	560
name	561
price	562
quantity	563
id	564
name	565
price	566
quantity	567
id	568
name	569
price	570
quantity	571
id	572
name	573
price	574
quantity	575
id	576
name	577
price	578
quantity	579
id	580
name	581
price	582
quantity	583
id	584
name	585
price	586
quantity	587
id	588
name	589
price	590
quantity	591
id	592
name	593
price	594
quantity	595
id	596
name	597
price	598
quantity	599
id	600
name	601
price	602
quantity	603
id	604
name	605
price	606
quantity	607
id	608
name	609
price	610
quantity	611
id	612
name	613
price	614
quantity	615
id	616
name	617
price	618
quantity	619
id	620
name	621
price	622
quantity	623
id	624
name	625
price	626
quantity	627
id	628
name	629
price	630
quantity	631
id	632
name	633
price	634
quantity	635
id	636
name	637
price	638
quantity	639
id	640
name	641
price	642
quantity	643
id	644
name	645
price	646
quantity	647
id	648
name	649
price	650
quantity	651
id	652
name	653
price	654
quantity	655
id	656
name	657
price	658
quantity	659
id	660
name	661
price	662
quantity	663
id	664
name	665
price	666
quantity	667
id	668
name	669
price	670
quantity	671
id	672</

Die gespeicherte Prozedur definieren

Vor der ersten Verwendung steht die Definition der Prozedur. Für das folgende Beispiel wird die Anwendung aus Listing 9.13 weiter ausgebaut. Im Enterprise Manager können Sie die Prozedur leicht anlegen. Wählen Sie dazu Ihre Datenbank aus und dort den Eintrag GESPEICHERTE PROZEDUREN. Im Kontextmenü klicken Sie auf NEUE GESPEICHERTE PROZEDUR....



Abbildung 9.16: So wird die gespeicherte Prozedur definiert.

Die spannende Frage ist nun, wie Sie diese Prozedur von ASP.NET aus aufrufen und den Parameter übergeben.

Die gespeicherte Prozedur im Code verwenden

Die folgende Applikation bietet eine Auswahlliste aller Artikelnummern. Nach erfolgter Auswahl werden die Bestellinformationen dieses Artikels ermittelt, also wie viele Bestellungen erfolgten und zu welchem Gesamtpreis.

Da die Ausgaben etwas umfangreicher sind, lohnt diesmal ein Blick auf die *aspx*-Datei:

```
<body MS_POSITIONING="GridLayout">
<h1>Kommando-Klasse</h1>
<h2>Gespeicherte Prozedur verwenden</h2>
<form runat="server">
    <asp:DropDownList Runat="server" ID="ArtikelNummern" />
    <asp:Button Runat="server" ID="ArtikelAuswahl" ⚡
        OnClick="ArtikelAuswahl_Click" ⚡
        Text="Daten zu dieser Artikelnummer anzeigen" />
</form>
<asp:DataGrid Runat="server" ID="Ausgabe"/>
<asp:Label Runat="server" ID="Fehler" /></div>
</body>
```

Listing 9.18: Steuerelemente für Auswahlformular und Ausgabe
(*SqlCommandStoredProc.aspx*)

Definiert wird hier ein `DropDownList`-Steuerelement, worin sich die Artikelnummern befinden. Eine Schaltfläche (`<asp:Button>`) verweist auf eine Ereignisbehandlungsmethode *ArtikelAuswahl_Click*. Diese Methode wird in Listing 9.20 vorgestellt. Das Ergebnis der Abfrage wird in einem `DataGrid`-Steuerelement ausgegeben. Sollten Fehler auftreten ist auch dafür wieder Platz in einem `Label`.

Der erste Teil des Codes muss sich nun um das Füllen der Auswahlliste kümmern:

```
Public Class SqlCommandStoredProc
    Inherits System.Web.UI.Page
    Protected Fehler As Label
    Protected ArtikelNummern As DropDownList
    Protected WithEvents ArtikelAuswahl As Button
    Protected Ausgabe As DataGrid

    Public conSql As SqlConnection = New SqlConnection()
    Private dataReader As SqlDataReader
    Private commSql As SqlCommand

    Public Sub Page_Init(ByVal sender As Object, ⚡
        ByVal eArgs As System.EventArgs) _
        Handles MyBase.Init
        conSql.ConnectionString ⚡
            = ConfigurationSettings.AppSettings("SqlConnection")
    End Sub

    Private Sub Page_Load(ByVal sender As Object, ⚡
        ByVal eArgs As System.EventArgs) _
```

```

Handles MyBase.Load
If Not Page.IsPostBack Then
    Try
        conSql.Open()
        If conSql.State = ConnectionState.Open Then
            commSql = New SqlCommand("SELECT id FROM Artikel",
conSql)
            dataReader = commSql.ExecuteReader(
                CommandBehavior.CloseConnection)
            ArtikelNummern.DataSource = dataReader
            ArtikelNummern.DataValueField = "id"
            ArtikelNummern.DataBind()
            dataReader.Close()
            conSql.Close()
        End If
    Catch e As Exception
        Fehler.BackColor = Color.Red
        Fehler.Text = e.Message
    End Try
End If
End Sub
'... siehe nächstes Listing für fehlenden Methoden
End Class

```

*Listing 9.19: Füllen der Auswahlliste und Initialisierung
(Ausschnitt aus SqlCommandStoredProc.aspx.vb)*

Hinweise Gegenüber der bereits gezeigten Form der Abfrage mit `SqlCommand` und dem daraus entstehenden `DataReader` gibt es hier keine Neuigkeiten. Lediglich die Art der Initialisierung der Verbindung wurde geändert. Die Zuweisung des Verbindungs-Objektes erfolgt nun schon bei der Instanziierung der Klasse. Die Verbindungszeichenfolge wird in der Methode `Page_Init` übernommen. Das hat keine Bedeutung für den Ablauf, sondern zeigt lediglich eine Variante, wie die elementaren Einstellungen sicher an den Anfang des Prozesses verlegt werden.

Formularbehandlung Die Änderung in `Page_Load` bezieht sich auf die Formularbehandlung. Es ist nun nicht mehr notwendig, bei jedem Abruf die Datenbank abzufragen, denn der Inhalt der Liste bleibt durch den `ViewState` der Seite erhalten. Deshalb kommt hier die Eigenschaft `IsPostBack` zum Einsatz.

Die Nutzung der gespeicherten Prozedur erfolgt erst, wenn auf die Schaltfläche geklickt wurde. Dann wird die Ereignisbehandlungsmethode `ArtikelAuswahl_Click` ausgelöst.


```

Public Sub ArtikelAuswahl_Click(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles ArtikelAuswahl.Click
    Try
        conSql.Open()
        If conSql.State = ConnectionState.Open Then
            commSql = New SqlCommand("BestellterArtikel", conSql)
            commSql.CommandType = CommandType.StoredProcedure
            commSql.Parameters.Add("@ArtikelNummer", 

ArtikelNummern.SelectedItem.Value.ToString())
            dataReader = commSql.ExecuteReader(
                CommandBehavior.CloseConnection)
            Ausgabe.DataSource = dataReader
            Ausgabe.DataBind()
            dataReader.Close()
            conSql.Close()
        End If
        Catch ex As Exception
            Fehler.BackColor = Color.Red
            Fehler.Text = ex.Message
        End Try
    End Sub

```

Listing 9.20: Die Ereignisbehandlungsmethode (Zweiter Ausschnitt aus SqlCommandStoredProc.aspx.vb)

Der an die Datenbank zu sendende Befehl besteht nun lediglich aus dem Prozedurnamen (»BestellterArtikel«):

Wie es funktioniert

```
commSql = New SqlCommand("BestellterArtikel", conSql)
```

Damit der Provider dies in ein richtiges Kommando umsetzen kann (T-SQL führt Prozeduren über den Befehl EXECUTE aus), muss der Kommandotyp gesetzt werden:

```
commSql.CommandType = CommandType.StoredProcedure
```

Prozeduren sind erst dann richtig leistungsfähig, wenn man Parameter übergeben kann. Dies erfolgt durch Hinzufügen der Werte an die Kollektion Parameters:

Parameter übergeben

```
commSql.Parameters.Add("@ArtikelNummer", 
```

```
ArtikelNummern.SelectedItem.Value.ToString())
```

Der Schlüssel bestimmt den Namen des Parameters, wie er in der Prozedur definiert wurde, einschließlich des führenden @-Zeichens. Der Wert – hier aus dem selektierten Element der Liste entnommen – wird dann als eigentlicher Parameterwert übergeben.

Dann erfolgt die Ausführung der Abfrage und es wird ein *DataReader* zurückgegeben:

```
dataReader = sqlCommand.ExecuteReader ←  
    (CommandBehavior.CloseConnection)
```

Damit ist der ganze Vorgang auch schon beendet. Das Ergebnis wird dem DataGrid-Steuerelement zugewiesen:

```
Ausgabe.DataSource = dataReader
```

Mit der Bindung erscheinen nun die Daten.

a-id	menge	summe	name	einzelpreis
4	2	198	Big Color Keyboard	99

Abbildung 9.17: Auswahl und Ergebnis – die Abfrage erfolgt mit einer gespeicherten Prozedur

Das Ergebnis zeigt, wie mit wenigen Steuerelementen und leistungsfähigen Datenbankbefehlen eine interessante Anwendung programmiert werden kann. Weit mehr Aufwand müsste man in der Praxis in die Gestaltung der Ausgabe, insbesondere des DataGrid, stecken. Mehr Informationen dazu finden Sie im Abschnitt 9.4, »Umgang mit Datenbanken in Visual Studio .NET« ab Seite 921.

Umgang mit Datentypen für Parameter

Datentypen

Im Beispiel wurde die einfachste Version der Datenübergabe gewählt – eine Zeichenkette. Die Parameterkollektion erwartet den Typ `Object`, sodass eigentlich auch jeder andere Typ akzeptiert wird. Vor allem können Sie die korrekten .NET-Datentypen übergeben, wenn Sie mit den SQL-Datentypen aus dem Namensraum `SqlTypes` arbeiten. Informationen darüber finden Sie im Abschnitt »Typsicherer Zugriff auf Daten« ab Seite 855.

9.3 Datenverwaltung mit ADO.NET

Den Kern der verbindungslosen Verwaltung von Daten in der ADO.NET-Welt bildet das so genannte *DataSet*. Daraus abgeleitete Objekte speichern die Daten. Gefüllt wird das *DataSet* aus den Ergebnissen einer Abfrage mit dem *DataAdapter*. Optional ist es möglich, die Daten in einem *DataSet* in mehreren Ansichten dem Benutzer bereit zu stellen. Hierfür ist die Klasse *DataView* verantwortlich. Damit die Präsentation dann auch in HTML gelingt, greifen Sie auf Daten-Steuerelemente wie *DataGrid*, *DataList* oder *Repeater* zurück.

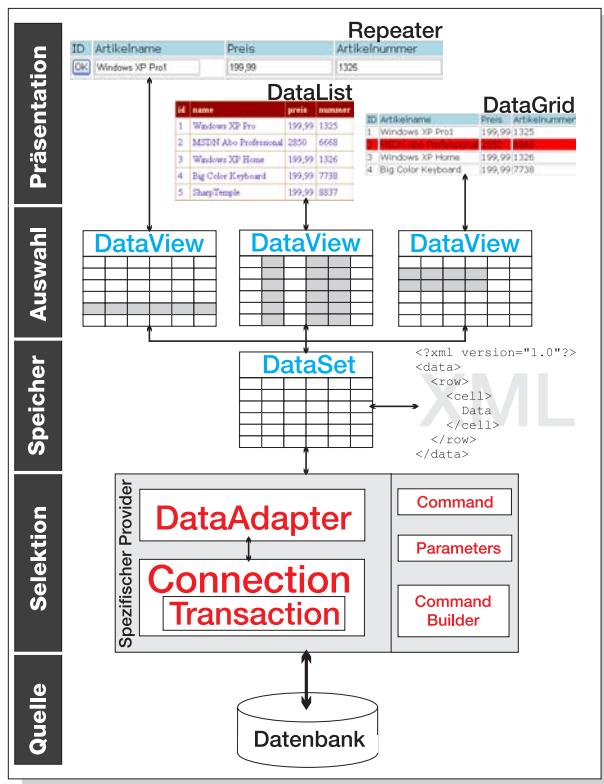


Abbildung 9.18: ADO.NET auf einen Blick. So wirken die Basisklassen zusammen.

Alle Basisklassen und damit verbundene werden in diesem Abschnitt ausführlich vorgestellt. Denken Sie auch bei diesen Ausführungen daran, dass Sie nicht alles verwenden müssen, was zu verwenden geht. Suchen Sie sich den für Ihre spezifische Anwendung sichersten Weg.

9.3.1 Der Datenspeicher: DataSet

Der letzte Abschnitt behandelte die Klassen und deren Eigenschaften und Methoden, die Bestandteil des Providers sind, also abhängig von der unterliegenden Datenbank. Das *DataSet* ist davon unabhängig, es wird hier keine Unterscheidung mehr zwischen SQL Server und OleDb getroffen. Die entsprechenden Zusammenhänge wurden bereits in Abbildung 9.3 gezeigt. Da viele Funktionen, die früher direkt mit der Datenbank zu tun hatten, im *DataSet* untergebracht sind, ist dieses Objekt entsprechend komplex.

Grundlagen zum DataSet

Das *DataSet* nutzt für die interne Speicherung XML. Dies erleichtert den Datenaustausch zwischen den Datenquellen und kann im Extremfall zum Verzicht auf eine Datenbank führen. Allerdings sind der lokalen Handhabung von Daten natürlich Grenzen gesetzt, vor allem im Hinblick auf Anwendungen unter hoher Last, wie es Websites häufig sind.

Um das *DataSet* zu verstehen, ist ein Blick auf die innere Struktur angebracht. Alle Teile in der folgenden Abbildung manifestieren sich in entsprechenden Klassen.

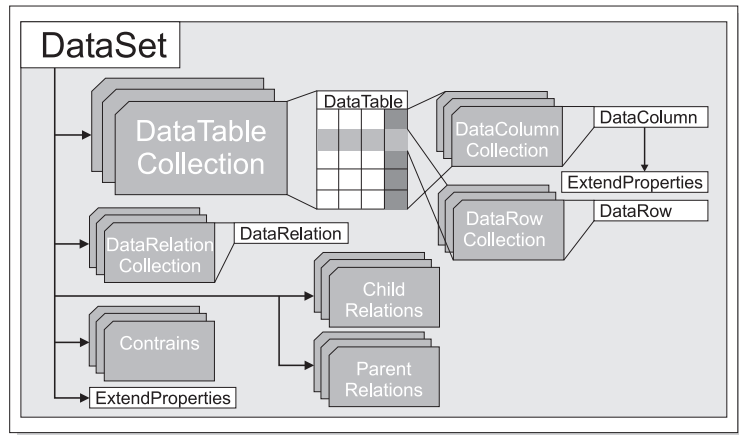


Abbildung 9.19: Aufbau eines DataSets

Tabellen Zentraler Bestandteil des *DataSet* ist eine Kollektion von Tabellen (*DataTableCollection*), die jeweils durch die Klasse *DataTable* repräsentiert werden. Das ist ein wesentlicher Unterschied zum alten

ADO: *DataSet* kann mehrere Tabellen speichern. Jede Tabelle besteht aus Spalten (Klasse *DataColumnCollection*), und optional Einschränkungen (Klasse *ContrainCollection*). Diese Klassen definieren quasi das Schema der Tabelle. Wenn Daten enthalten sind, werden diese in Reihen (Klasse *DataRowCollection*) gehalten. Aus diesen Kollektionen leiten sich dann die für die eigentlichen Datenoperationen wichtigen Klassen ab, die jeweils durch die entsprechenden Klassen gebildet werden, das heißt, *DataColumn* repräsentiert eine Spalte, *DataRow* eine Reihe usw.

Neben Tabellen werden in relationalen Datenbanken auch Beziehungen definiert. Da ein *DataSet* losgelöst von der Datenbank existiert, müssen auch diese Beziehungen nachgebildet werden. Sie werden auch als Kollektion *DataRelationCollection* gespeichert, jede einzelne repräsentiert durch *DataRelation*.

Beziehungen

Innerhalb des *DataSet* gibt es eine Kollektion von erweiterten Eigenschaften, erreichbar über die Eigenschaft *ExtendedProperties*. Darin können Sie eigene Informationen speichern. Diese Eigenschaften haben keinen Einfluss auf das *DataSet* selbst. Sie können aber Daten damit persistent machen und an andere Datensenzen übertragen. *ExtendedProperties* ist vom Typ *PropertyCollection*, der direkt von *Hashtable* erbt. In Abschnitt 4.3, »Aufzählungen und Kollektionen« ab Seite 222 finden Sie Informationen über den Umgang mit derartigen Kollektionen.

**Erweiterte
Eigenschaften**

Datenansicht und Ausgaben

Nachdem die Daten in der einen oder anderen Form im *DataSet* gespeichert sind, müssen Sie natürlich einen Weg finden, diese zu verarbeiten. Dazu gibt es mehrere Varianten.

Der einfachste besteht darin, die Daten direkt an ein Daten-Steuer-element zu binden. Das wurde bereits bei einigen Beispielen gezeigt und wird in Abschnitt 9.4, »Umgang mit Datenbanken in Visual Studio .NET« ab Seite 921 vertieft. Unabhängig davon wird diese Methode – vor allem wegen ihrer frappierenden Einfachheit – auch schon bei den ersten Schritten mit dem *DataSet* gezeigt. Lassen Sie sich davon nicht irritieren, die Details folgen später.

**Daten-
Steuerelemente**

Des Weiteren können Sie die Daten aus dem *DataSet* natürlich immer direkt extrahieren. Über die Kollektionen von Reihen und Spalten ist jeder einzelne Wert selektierbar. Die Inhalte lassen sich dann normalen Steuerelementen zuweisen oder anderweitig ver-

Direkte Extraktion

wenden. Das setzt natürlich voraus, dass Sie die genaue Position der benötigten Daten kennen.

DataView Ein weiterer und sehr eleganter Weg besteht in der Nutzung einer Datenansicht, der so genannten *DataView*. Diese Datenansichten beeinflussen nicht die Art der Speicherung oder den Inhalt eines *DataSet*, sondern filtern, sortieren oder manipulieren die Daten nur zum Zweck der Anzeige. Der Umgang mit *DataView* ist nicht besonders schwierig, deshalb wird dieser Teil »nebenbei« im Zusammenhang mit den Beispielen zum *DataSet* abgehandelt.

Allgemeine Hinweise zu Einschränkungen und Beziehungen

Das *DataSet* erlaubt die Definition von so genannten Relationen über die Klasse *DataRelation* und die Eigenschaft *DataRelations*. Darüber hinaus können aber auch Einschränkungen mit *ForeignKeyConstraint* definiert werden. Die Relationen definieren automatisch die benötigten Einschränkungen in Bezug auf die Beziehungen der Spalten in den betroffenen Tabellen. Das heißt konkret, dass das Erzeugen einer Relation im *DataSet* auch zum Erzeugen der FOREIGN KEY- und UNIQUE-Einschränkung führen kann. Der Vorgang kann durch einen entsprechenden Parameter gesteuert werden.

Wenn nun die Einschränkungen über FOREIGN KEY und *DataRelation* gleichermaßen definiert werden können, ergibt sich die Frage, welche Variante in der Praxis gewählt werden sollte. Tatsächlich gibt es Unterschiede. *DataRelation* führt dazu, dass bei der Auswahl von Daten die Beziehungen berücksichtigt werden können. Die Informationen werden dazu verwendet, beim Abruf von Daten aus einer Tabelle die verknüpften Informationen in einer anderen Tabelle zu lesen. Allerdings führt ein Verstoß gegen die Beziehung, beispielsweise durch Löschen eines Datensatzes, nicht zu einem Fehler bzw. dem Auslösen einer Ausnahme. Genau das ist Aufgabe der Einschränkungen, die – ebenso wie beim SQL Server – Verstöße nicht zulassen und die Operation wirkungsvoll unterbinden. In Umgebungen, die überwiegend nur lesende Zugriffe benötigen, ist die Programmierung einer *DataRelation* ausreichend. Sind Datenbeziehungen dagegen vor allem dafür festgelegt worden, die Konsistenz von abhängigen Tabellen bei Schreibvorgängen zu sichern, sind echte Einschränkungen sinnvoller.

Konkret bedeutet das, dass FOREIGN KEY-Einschränkungen gelten, wenn Daten geändert oder hinzugefügt werden. Das passiert beim Aufruf folgender Methoden eines DataSet-Objekts oder untergeordneter Objekte:

- ▶ DataSet.Merge
- ▶ DataTable.LoadDataRow
- ▶ DataRowCollection.Add
- ▶ DataRow.EndEdit
- ▶ DataRow.ItemArray

Isolierte DataSet-Objekte verwenden

Für einfache Anwendungen kann es ausreichend sein, das DataSet lokal zu erstellen und dann zu verwenden. Das setzt voraus, dass die entsprechenden Methoden zum Erzeugen von Tabellen, Spalten und Referenzen bekannt sind. Das folgende Beispiel nutzt dies, um Adressen und eine Liste zugehöriger Telefonnummern anzuzeigen. Betrachten Sie zuerst die Definition der *aspx*-Seite:

```
<h1>Umgang mit lokalem DataSet</h1>
<form id="DataSetCreateLocal" method="post" runat="server">
  Wählen Sie einen Kunden aus:<br>
  <asp:DropDownList id="Adressen" runat="server"
    AutoPostBack="True" Width="166px"
    OnSelectedIndexChanged="Adressen_OnChange"/>
</form>
<asp:DataGrid Runat="server" ID="Telefon" />
```

Listing 9.21: Auswahlliste für Adressen und Telefonnummernanzeige (*DataSetCreateLocal.aspx*)

Das fertige Programm soll dann etwa folgende Ausgaben erzeugen:

Umgang mit lokalem DataSet

Wählen Sie einen Kunden aus:

Lukas Werlich ▼

id	aid	nummer
3	2	089-9987653
4	2	0174-9992834

Abbildung 9.20: Nach der Auswahl eines Namens erscheinen die passenden Telefonnummern.

Auf dieser Basis werden mindestens zwei Schritte zur Darstellung der Daten benötigt. Zum einen muss beim ersten Start des Programms die Auswahlliste gefüllt werden. Danach bleibt sie durch den Anzeigestatus der Seite erhalten. Nachdem eine Auswahl getroffen wurde, muss das `DataGrid`-Steuerelement mit den passenden Telefonnummern gefüllt werden.

In dieser Applikation sollen zentrale relationale Daten vorgehalten werden, ohne dass eine Datenbank zum Einsatz kommt. Gleichzeitig sollen die Daten allen Benutzern zur Verfügung stehen. Es ist eine gute Idee, bei relativ geringen Datenmengen dazu Applikationsvariablen einzusetzen. Sie finden in Abschnitt 8.6, »Applikationsmanagement« ab Seite 802 ausführliche Informationen über den Einsatz und die Wirkungsweise. Die für das hier beschriebene Beispiel erforderliche Modifikation der *global.asax.vb* wird in Listing 9.24 beschrieben.

Betrachten Sie zuerst die `Page_Load`-Methode, in der das `DataSet` aus der Applikationsvariablen abgerufen und dann zum Füllen der Auswahlliste eingesetzt wird:

```
Private Sub Page_Load(ByVal sender As Object, 
                        ByVal e As System.EventArgs) Handles
MyBase.Load
    ds = CType(Application("Adressen"), DataSet)
    If Not Page.IsPostBack Then
        Adressen.DataSource = ds
        Adressen.DataMember = "Adressen"
        Adressen.DataValueField = "id"
        Adressen.DataTextField = "name"
        Adressen.DataBind()
    End If
End Sub
```

*Listing 9.22: Abruf des DataSet und Füllen der Auswahlliste
(Ausschnitt aus DataSetCreateLocal.aspx.vb)*

**Wie es
funktioniert**

Applikationsvariablen speichern Elemente immer als Typ `Object`. Das hat den Vorteil, dass Sie nicht in Bezug auf mögliche Speicherobjekte eingeschränkt sind. Beim Abruf der Elemente muss natürlich eine Typumwandlung in den ursprünglichen Typ vorgenommen werden, hier das `DataSet`:

```
ds = CType(Application("Adressen"), DataSet)
```

Der folgende Teil wird nur ausgeführt, wenn das Formular noch nicht zurückgesendet wurde, denn nur dann muss die Auswahl-

liste gefüllt werden. Bei allen folgenden Aufrufen erledigt dies der Anzeigestatus (*ViewState*). Dann wird dem Steuerelement das *DataSet* als Datenquelle zugewiesen:

```
Adressen.DataSource = ds
```

Da die Adressen von Interesse sind, wird die entsprechende Tabelle zum Datenmitglied erklärt:

```
Adressen.DataMember = "Adressen"
```

Nun wird bestimmt, dass die Optionsfelder mit der *id* und die Anzeige mit *name* gefüllt wird. Mit der folgenden Bindung erscheinen die Daten im Steuerelement.

Wie an der Definition in Listing 9.21 zu sehen ist, fehlt eine Sendeschaltfläche. Stattdessen sorgt das Attribut `AutoPostBack` dafür, dass bei Änderungen das Formular gesendet wird. Da nur dann eine Aktualisierung der Anzeige erforderlich ist, wenn die Listenauswahl auch wirklich geändert wurde, wird das Ereignis `OnChange` ausgewertet, basierend auf folgendem Attribut:

```
OnSelectedIndexChanged="Adressen_OnChange"
```

Der zweite Teil der Applikation besteht in der Definition der Ereignisbehandlungsmethode:

```
Public Sub Adressen_OnChange(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles Telefon.SelectedIndexChanged
    Dim dv As DataView = New DataView(ds.Tables("Telefone"))
    dv.RowFilter = String.Format("aid={0}", 
        Adressen.SelectedItem.Value.ToString())
    Telefon.DataSource = dv
    Telefon.DataMember = "Telefone"
    Telefon.DataBind()
End Sub
```

Listing 9.23: Anzeigen der Nummern an Hand der Auswahl
(Ausschnitt aus *DataSetCreateLocal.aspx.vb*)

Hier wird eine *DataView* eingesetzt, um die passenden Daten zu selektieren. Angelegt wird sie auf Basis der zweiten Tabelle, *Telefone*.

Wie es funktioniert

```
Dim dv As DataView = New DataView(ds.Tables("Telefone"))
```

Ein *DataView*-Objekt kann Daten filtern, hier auf Basis der Auswahlstewerte, die die *id*-Nummern der Adressen enthalten:

```
dv.RowFilter = String.Format("aid={0}", 
    Adressen.SelectedItem.Value.ToString())
```

Der Rest der Methode besteht wieder im Zuweisen der Datenansicht (anstatt des gesamten *DataSet*) an das Steuerelement, das diesmal ein *DataGrid* ist. Zum *DataGrid* finden Sie im Abschnitt 6.6.5, »DataGrid« ab Seite 605 detaillierte Informationen.

Das DataSet erzeugen

Nun bleibt noch die größere Aufgabe, das eigentliche *DataSet* zu erzeugen und als Applikationsvariable zu übergeben. Da die Daten allen Benutzern gleichermaßen, quasi statisch, zur Verfügung stehen sollen, genügt es, diese zum Start der Applikation zu generieren. Dazu eignet sich das Ereignis *Application_Start*, dessen Ereignisbehandlungsmethode Sie in der Datei *global.asax* im Stammverzeichnis der Applikation finden. Dort wird zuerst Folgendes programmiert:

```
Protected Sub Application_Start(ByVal sender As Object,
                                ByVal e As EventArgs)
    Dim localDs As DataSet = CreateDataSet()
    FillDataSet(localDs)
    Application("Adressen") = localDs
End Sub
```

Listing 9.24: Erzeugen des DataSet in der Datei global.asax.vb (Ausschnitt aus der Code Behind-Datei der global.asax)

Statische Daten hinterlegen

Zwei etwas umfangreichere Methoden, die noch programmiert werden müssen, werden hier aufgerufen. *CreateDataSet* erzeugt die Struktur der Tabellen. In SQL entspräche dieser Vorgang dem Absenden von *CREATE TABLE*-Befehlen. Mit *FillDataSet* werden die beiden Tabellen dann mit einigen Daten gefüllt. Zuletzt wird das fertige *DataSet* der Applikationsvariablen *Adressen* zugewiesen:

```
Application("Adressen") = localDs
```

Die zwei Methoden werden hier nur ausschnittsweise wiedergegeben. Den vollständigen Code finden Sie in der Datei *global.asax.vb* des Projekts *vbnetdbase*.

```
Private tempRow As DataRow
Private tempTel As ArrayList = New ArrayList()
Private currentId As Integer

Private Function CreateDataSet() As DataSet
    Dim ds As DataSet = New DataSet()
    Dim dtAdressen As DataTable = New DataTable("Adressen")
    Dim dtTelefon As DataTable = New DataTable("Telefon")
    ' Adressen
    dtAdressen.Columns.Add("id", Type.GetType("System.Int32"))
    dtAdressen.Columns.Add("name", Type.GetType("System.String"))
```

```

    dtAdressen.Columns.Add("strasse",
Type.GetType("System.String"))
    dtAdressen.Columns.Add("plz", Type.GetType("System.String"))
    dtAdressen.Columns.Add("ort", Type.GetType("System.String"))
    ' Telefone
    dtTelefon.Columns.Add("id", Type.GetType("System.Int32"))
    dtTelefon.Columns.Add("aid", Type.GetType("System.Int32"))
    dtTelefon.Columns.Add("nummer", Type.GetType("System.String"))
    dtTelefon.Columns("nummer").Caption = "Telefonnummer"
    ' Referenzen
    Dim pk() As DataColumn = New DataColumn(1) {}
    dtAdressen.Columns("id").AutoIncrement = True
    dtAdressen.Columns("id").AutoIncrementSeed = 1
    dtAdressen.Columns("id").ReadOnly = True
    pk(0) = dtAdressen.Columns("id")
    dtAdressen.PrimaryKey = pk
    dtTelefon.Columns("id").AutoIncrement = True
    dtTelefon.Columns("id").AutoIncrementSeed = 1
    dtTelefon.Columns("id").ReadOnly = True
    pk(0) = dtTelefon.Columns("id")
    dtTelefon.PrimaryKey = pk
    ds.Tables.Add(dtAdressen)
    ds.Tables.Add(dtTelefon)
    ds.Relations.Add("AdresseTelefon", ↵
                                dtAdressen.Columns("id"), ↵
                                dtTelefon.Columns("aid"))

    Return ds
End Function

```

```

Private Sub FillDataSet(ByRef ds As DataSet)
    ' Daten für Adresse 1
    tempRow = ds.Tables("Adressen").NewRow()
    tempRow("name") = "Jörg Krause"
    tempRow("strasse") = "Dornenweg 33"
    tempRow("plz") = "12683"
    tempRow("ort") = "Berlin"
    ds.Tables("Adressen").Rows.Add(tempRow)
    tempTel.Add("030-1234567")
    tempTel.Add("0172-9876544")
    currentId = Convert.ToInt32(tempRow("id"))
    Dim s As String
    For Each s In tempTel
        tempRow = ds.Tables("Telefone").NewRow()
        tempRow("nummer") = s
        tempRow("aid") = currentId
        ds.Tables("Telefone").Rows.Add(tempRow)
    Next
    tempTel.Clear()
    ' Daten für Adresse 2

```

```

tempRow = ds.Tables("Adressen").NewRow()
tempRow("name") = "Lukas Werlich"
tempRow("strasse") = "Spielallee 11"
tempRow("plz") = "89487"
tempRow("ort") = "München"
ds.Tables("Adressen").Rows.Add(tempRow)
tempTel.Add("089-9987653")
tempTel.Add("0174-9992834")
currentId = Convert.ToInt32(tempRow("id"))
For Each s In tempTel
    tempRow = ds.Tables("Telefone").NewRow()
    tempRow("nummer") = s
    tempRow("aid") = currentId
    ds.Tables("Telefone").Rows.Add(tempRow)
Next
tempTel.Clear()

' Daten für Adresse 3
tempRow = ds.Tables("Adressen").NewRow()
tempRow("name") = "Carolin Scholz"
tempRow("strasse") = "Eichenwall 4"
tempRow("plz") = "40748"
tempRow("ort") = "Düsseldorf"
ds.Tables("Adressen").Rows.Add(tempRow)
tempTel.Add("0211-50023875")
tempTel.Add("0161-98237547")
currentId = Convert.ToInt32(tempRow("id"))
For Each s In tempTel
    tempRow = ds.Tables("Telefone").NewRow()
    tempRow("nummer") = s
    tempRow("aid") = currentId
    ds.Tables("Telefone").Rows.Add(tempRow)
Next
tempTel.Clear()

```

End Sub

Listing 9.25: Prinzip der Hinterlegung statischer Daten (Ausschnitt aus der Code Behind-Datei der global.aspx)

Wie es funktioniert

Die Methode *CreateDataSet* erzeugt ein neues *DataSet*-Objekt und füllt es dann mit einer – vorerst leeren – Tabellenstruktur:

```
Dim ds As DataSet = New DataSet()
```

DataTable

Am Beispiel der Tabelle *Adressen* soll gezeigt werden, wie Sie eine solche Struktur entwerfen. Zuerst wird die Tabelle selbst erzeugt:

```
Dim dtAdressen As DataTable = New DataTable("Adressen")
```

Dann werden der Tabelle die Spalten hinzugefügt. Dies geschieht durch Anfügen von Spalten an die `Columns`-Kollektion, die über die Eigenschaft `Columns` zugänglich ist. Benötigt wird mindestens ein Spaltenname und der Datentyp:

```
dtAdressen.Columns.Add("id", Type.GetType("Int32"))
```

Tatsächlich hat die Methode `Add` für Spalten, wie alle anderen Konstrukturen auch, viele Überladungen, die die Angabe der Parameter erlauben. So können Sie auch eine leere Spalte anlegen und später mit dem Namen und verschiedenen Eigenschaften versehen. Einige Tipps dazu finden Sie im Abschnitt 9.3.2, »Die Struktur einer Tabelle im DataSet festlegen« ab Seite 884.



Die anderen Spalten werden ähnlich konstruiert, nur der Name und natürlich der Datentyp unterscheiden sich.

Mit der Spalte `id` ist noch besonders zu verfahren, denn dies ist sowohl der Primärschlüssel (PRIMARY KEY) und außerdem soll der Schlüssel automatisch vergeben werden. Der Primärschlüssel wird einer Spaltenkollektion entnommen, auch wenn nur eine Spalte dafür in Frage kommt. Erzeugen Sie dazu also eine solche Kollektion:

```
Dim pk() As DataColumn = New DataColumn(1) {}
```

Dem ersten Element wird dann die Spalte zugeordnet, die den Primärschlüssel darstellt:

```
pk(0) = dtAdressen.Columns("id")
```

Die Kollektion wird dann der Eigenschaft `PrimaryKey` der Tabelle zugewiesen:

```
dtAdressen.PrimaryKey = pk
```

Nun werden noch die Eigenschaften zur automatischen Zählung (AUTOINCREMENT) der Spalte bestimmt:

```
dtAdressen.Columns("id").AutoIncrement = True
```

Festgelegt wird auch der Startwert:

```
dtAdressen.Columns("id").AutoIncrementSeed = 1
```

Damit im Code nicht versehentlich der automatisch erzeugte Wert überschrieben wird, kann die Spalte als »readonly« gekennzeichnet werden:

```
dtAdressen.Columns("id").ReadOnly = True
```

Wird dennoch ein Schreibversuch unternommen, wird ein Laufzeitfehler ausgelöst, den Sie abfangen und verarbeiten können.

Die fertige Tabelle wird nun dem DataSet hinzugefügt:

```
ds.Tables.Add(dtAdressen)
```

Abschließend soll noch eine Beziehung definiert werden, die die Tabelle *Telefone* mit der Tabelle *Adressen* verknüpft. Diese Beziehung erzeugt, wie zuvor bereits beschrieben, eine FOREIGN KEY-Einschränkung. Wenn Sie nun versuchen, Telefonnummern einzutragen, zu denen noch keine Adresse existiert, wird die Ausnahme `InvalidConstraintException` erzeugt.

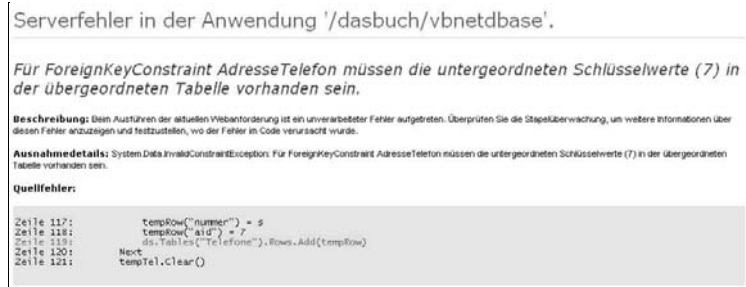


Abbildung 9.21: Der Verstoß gegen die Beziehung führt zu einer Ausnahme

In der Abbildung sehen Sie, dass der Fehler beim Hinzufügen der Reihe zur Tabelle *Telefone* erzeugt wurde (der Wert »7« existierte nicht). Schauen Sie sich nun den Code der Methode *FillDataSet* an, um zu erfahren, wie die Programmierung korrekt erfolgt.

Prinzipiell werden hier alle Reihen quasi »von Hand« erzeugt. Das ist nur dann ein typisches Verfahren, wenn Sie die Daten tatsächlich »hart« kodieren wollen oder müssen. Typischer ist sicher die Übernahme der Einträge aus einem Formular oder einer externen Datenquelle. Das ändert aber nichts an der Vorgehensweise, wie sie das vorliegende Beispiel zeigt. Durch die feste Vergabe der Werte wird der Code hier jedoch vereinfacht.

Zuerst wird eine neue Reihe erzeugt, vorerst noch leer:

```
tempRow = ds.Tables("Adressen").NewRow()
```



Der folgende – eigentlich nahe liegende – Code funktioniert nicht:

```
Dim tempRow As DataRow = New DataRow()
```

Warum ist das so? Die *DataRow*-Klasse stellt die tatsächlich in einer Tabelle enthaltenen Daten dar. Das heißt, Sie müssen auch eine leere Reihe aus einer vorhandenen Tabelle ableiten, wenn Sie dieser Daten zuweisen wollen. Dazu dient die Methode *NewRow*.

Nun werden den Spalten der neuen Reihe die Werte spaltenweise zugewiesen:

```
tempRow("name") = "Jörg Krause"
```

Am Ende wird die neue Spalte der Kollektion der Reihen der Tabelle angehängt. Da die Reihe vorher aus der Tabelle selbst erzeugt wurde, stimmen die Spalten überein und der Vorgang gelingt immer:

```
ds.Tables("Adressen").Rows.Add(tempRow)
```

Bei der Beschreibung der Erstellung der Tabellenstruktur wurde gezeigt, dass die Spalte *id* nicht beschrieben werden kann und die Werte automatisch, fortlaufend zählend, vergeben werden. Für die Verknüpfung der Werte in der Tabelle *Telefone* wird dieser automatisch erzeugte Wert benötigt. Er soll der Spalte *aid* zugewiesen werden. Unmittelbar nach dem Anhängen der neuen Reihe an die Tabelle *Adressen* ist dieser Wert schon bekannt:

```
currentId = Convert.ToInt32(tempRow("id"))
```

Nun kann die Tabelle *Telefone* beschrieben werden. Zur Demonstration der Möglichkeiten wird hier ein *ArrayList*-Objekt verwendet und mit einer *For Each*-Schleife durchlaufen. Bei jedem Schleifendurchlauf wird eine Reihe erzeugt und die Nummer sowie die ID aus der Variablen *currentId* zugewiesen:

```
For Each s In tempTel
    tempRow = ds.Tables("Telefone").NewRow()
    tempRow("nummer") = s
    tempRow("aid") = currentId
    ds.Tables("Telefone").Rows.Add(tempRow)
Next
```

Damit ist das Programm auch schon lauffähig. Nicht gezeigt wurden hier die benötigten Zuweisungen der Namensräume und die Deklarationen globaler Felder. Konsultieren Sie also unbedingt den Code auf der CD zum Buch. Das fertige Programm führt zu einer Anzeige, wie sie in Abbildung 9.20 bereits gezeigt wurde.

9.3.2 Die Struktur einer Tabelle im DataSet festlegen

Um die Struktur einer Tabelle im *DataSet* bestimmen zu können, gibt es mehrere Wege. Dieser Abschnitt zeigt die prinzipiellen Techniken. Was in der Praxis eingesetzt wird, hängt von der Aufgabenstellung ab. Auch hier gilt: Nicht alles, was möglich ist, muss auch verwendet werden.

Erzeugen einer Spalte und Zuweisen spezifischer Eigenschaften

Im letzten Abschnitt wurde die Struktur direkt im Code erzeugt, durch Zuweisen der Eigenschaften von Spalten. Dies ist möglicherweise umständlich, aber auch sehr flexibel. Denkbar wäre es, einige Strukturen davon abhängig zu machen, welche Bedingungen zur Laufzeit herrschen. So könnten die Tabellen auf verschiedenen Installationen unterschiedlich gestaltet sein.

Die folgende Tabelle zeigt einige wichtige Eigenschaften für Spalten:

Eigenschaft	Datentyp	Beschreibung
AllowDBNull	Boolean	Der Wert NULL wird erlaubt, wenn diese Eigenschaft true ist.
ColumnName	String	Name der Spalte
Caption	String	Beschriftung für Spalte für Steuerelemente, die das unterstützen
DataTyp	Type	Datentyp der Spalte, beispielsweise mit GetType oder typeof ermittelt
DefaultValue	Object	Standardwert (entspricht DEFAULT in SQL)
Expression	String	Ausdruck für eine berechnete Spalte. Zulässig sind alle Aggregatfunktionen und Operatoren aus SQL.
ExtendedProperties	PropertyCollection	Nutzerdefinierte Eigenschaften. PropertyCollection erbt direkt von Hashtable und verhält sich entsprechend.
MaxLength	Integer	Maximale Breite der Spalte in Zeichen

Tabelle 9.2: Wichtige Eigenschaften der Klasse *DataColumn* zum Festlegen von Spalteneigenschaften

Eigenschaft	Datentyp	Beschreibung
AutoIncrement	Boolean	Steuerung der automatischen Wertvergabe für Index-Spalten
AutoIncrement	Integer	
Seed	Integer	
AutoIncrement Step		
ReadOnly	Boolean	Spalte als schreibgeschützt festlegen
Unique	Boolean	Spalte als <code>UNIQUE</code> festlegen (Werte müssen eindeutig sein)

Tabelle 9.2: Wichtige Eigenschaften der Klasse `DataColumn` zum Festlegen von Spalteneigenschaften (Forts.)

Darüber hinaus gibt es weitere Eigenschaften, die beim Umgang mit Spalten hilfreich sein können. Dazu gehört die Eigenschaft `Table`, mit der der Name der Tabelle ermittelt werden kann, zu dem diese Spalte gehört. Der Wert ist `null`, solange noch keine Zuweisung erfolgte. Benötigt wird gelegentlich auch `Ordinal`. Mit dieser Eigenschaft ist es möglich, die Nummer der Spalte in der Auflistung zu ermitteln. Spalten zählen nullbasiert.

Wenn Sie im `DataSet`-Objekt Daten speichern, die explizit als XML ausgegeben werden sollen oder die aus einer XML-Quelle stammen, können die Spalten zu einem speziellen Namensraum gehören. Dann ruft die Eigenschaft `Namespace` diesen ab oder legt ihn fest. Mit `Prefix` wird der Aliasname des Namespace festgelegt oder ermittelt.

XML

Ein kleines Programm soll zeigen, wie Sie die Definition von berechneten Spalten mit der Eigenschaft `Expression` verwenden können. Zuerst folgt wieder ein Blick auf den HTML-Quelltext. Realisiert wird ein Euro-Währungsumrechner mit einer Besonderheit. Nach der Auswahl der Ausgangswährung in Euro kann der Benutzer eine Liste von Werten eingeben und sich so die älteren Umrechnungen anschauen:

```
<h1>Umgang mit lokalem DataSet</h1>
<h2>Nutzung berechneter Spalten</h2>
<form id="DataSetLocalExpression" method="post" runat="server">
  Wählen Sie zuerst die benötigte Umrechnung:<br>
  <asp:RadioButtonList Runat="server" ID="Waehrungen"
    AutoPostBack="True"
    OnSelectedIndexChanged=
      "Waehrungen_Changed"
    RepeatColumns="3"
```

```

RepeatDirection="Horizontal" />
Tragen Sie einen Wert ein, der der Liste hinzugefügt werden soll:
<br>
<asp:TextBox Runat="server" ID="Eingabe" />
<asp:Button Runat="server" ID="Absenden" 
            OnClick="Absenden_OnClick" Text="Berechnen" />
<asp:Label Runat="server" ID="Fehler" />
</form>
<asp:DataGrid Runat="server" ID="Ausgabe" />

```

Listing 9.26: Ein komfortabler Währungsrechner (DataSetLocalExpression.aspx)

Bevor Sie sich den Code näher anschauen, soll ein Blick auf das erwartete Ergebnis gewährt werden:

Umgang mit lokalem DataSet

Nutzung berechneter Spalten

Wählen Sie zuerst die benötigte Umrechnung:

☒ DEM
 ☐ PTE
 ☐ FIM
☐ BEF
 ☐ IRP
 ☐ LUF
☐ ESP
 ☐ NLG
 ☐ FRF
☐ GRD
 ☐ ITL
 ☐ ATS

Tragen Sie einen Wert ein, der der Liste hinzugefügt werden soll:

Original Euro	
12	6,13550257435462
100	51,1291881196219
1000	511,291881196218
19,99	10,2207247051124
24,9	12,7311678417858
48,9	25,0021729904951

Abbildung 9.22: Der Währungsrechner, mit DataSet programmiert

Im oberen Teil findet der Benutzer eine Reihe von Optionsfeldern, die die Auswahl einer der Währungen der Euroländer erlauben. Darunter ist ein Eingabefeld zu sehen, in das der Betrag in der alten Landeswährung eingegeben werden kann. Jeder erfasste Wert wird dann der Liste im unteren Teil hinzugefügt, wobei alte Währung und Euro nebeneinander stehen. Wie im HTML-Code zu sehen ist, wird die Liste durch ein DataGrid-Steuerelement dargestellt. Es werden insgesamt drei Ereignisse ausgewertet, für die drei Ereignisbehandlungsmethoden programmiert werden müssen. Das erste ist – wie üblich – Page_Load. Hier erfolgt das Füllen der Optionsfelder mit den Währungsnamen und Umrechnungsfaktoren.

```

Public Class DataSetLocalExpression
    Inherits System.Web.UI.Page
    Protected Eingabe As TextBox
    Protected Fehler As Label
    Protected WithEvents Absenden As Button
    Protected Ausgabe As DataGrid
    Protected WithEvents Waehrungen As RadioButtonList

    Private ds As DataSet = New DataSet()
    Private tempC As DataColumn
    Private faktor As String
    Private faktoren As Hashtable = New Hashtable()

    Private Sub Page_Load(ByVal sender As Object,
                          ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Page.IsPostBack Then
            faktoren.Add("DEM", 1.95583)
            faktoren.Add("ATS", 13.7603)
            faktoren.Add("BEF", 40.339901)
            faktoren.Add("ESP", 166.386)
            faktoren.Add("FIM", 5.94573)
            faktoren.Add("FRF", 6.55957)
            faktoren.Add("IRP", 0.787564)
            faktoren.Add("ITL", 1936.27)
            faktoren.Add("LUF", 40.3399)
            faktoren.Add("NLG", 2.20371)
            faktoren.Add("PTE", 200.482)
            faktoren.Add("GRD", 340.75)
            Waehrungen.DataSource = faktoren
            Waehrungen.DataTextField = "Key"
            Waehrungen.DataValueField = "Value"
            Waehrungen.DataBind()
        End If
    End Sub

```

Listing 9.27: Die Deklaration der globalen Felder und die Methode `Page_Load` (Ausschnitt aus `DataSetLocalExpression.aspx.vb`)

Da die Anzahl der Währungen und die Umrechnungsfaktoren in Euroland keinen Schwankungen unterliegen, ist eine feste Kodierung vertretbar. Hier wird der Typ `Hashtable` verwendet, um die Währungskürzel und die Faktoren aufzunehmen. Die Bindung erfolgt dann an das `RadioButtonList`-Steuerelement, wobei der Schlüsselwert »Key« zur Anzeige des Textes und »Value« den Optionswerten zugewiesen wird. Die Liste reagiert auf einen Klick mit dem Absenden des Formulars durch *AutoPostBack*.

Wie es funktioniert

Nach dem durch *AutoPostBack* ausgelösten Ereignis setzt das Programm seine Arbeit mit der Ereignisbehandlungsmethode *Waehrungen_Changed* fort. Dort wird nun die Tabelle erzeugt, die später zur Anzeige der Liste der Währungen führt. Die Konstruktion der Tabelle erfolgt dynamisch und der Umrechnungsfaktor wird als *Expression*-Eigenschaft programmiert. Auch hier gibt es eine kleine Besonderheit. Bei der Verarbeitung der internen Gleitkommazahlen wird die standardmäßige Kultur verwendet. Mehr Informationen zu Kulturen finden Sie im Abschnitt 4.10.2, »*Mehrsprachige Seiten programmieren*« ab Seite 296. Das führt dazu, dass *ToString* die Zahlen in das deutsche Format wandelt; mit einem Komma als Dezimaltrennzeichen. Durch Verändern der Kultur auf »en-US« wird das korrekte Format für die Verarbeitung von Ausdrücken verwendet.

```
Public Sub Waehrungen_Changed(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
Handles Waehrungen.SelectedIndexChanged
    Dim us As CultureInfo = New CultureInfo("en-US")
    faktor =
Convert.ToDouble(Waehrungen.SelectedItem.Value).ToString(us)
    Dim waehrungsTabelle As DataTable = New DataTable("Currency")
    tempC = New DataColumn()
    tempC.ColumnName = "Original"
    tempC.DataType = Type.GetType("System.Double")
    tempC.Unique = True
    waehrungsTabelle.Columns.Add(tempC)
    tempC = New DataColumn()
    tempC.ColumnName = "Euro"
    tempC.DataType = Type.GetType("System.Double")
    tempC.Expression = "Original / " + faktor
    waehrungsTabelle.Columns.Add(tempC)
    ds.Tables.Add(waehrungsTabelle)
    Session("MyDataSet") = ds
    Ausgabe.DataSource = ds
    Ausgabe.DataMember = "Currency"
    Ausgabe.DataBind()
End Sub
```

*Listing 9.28: Erzeugen des DataSet in Abhängigkeit von der Währungsauswahl
(Ausschnitt aus DataSetLocalExpression.aspx.vb)*

**Wie es
funktioniert**

Das Erzeugen einer anderen Kultur ist sehr einfach. Voraussetzung ist die Einbindung des passenden Namensraumes:

```
Imports System.Globalization
```

Das Anlegen der Kultur ist dann sehr einfach:

```
Dim us As CultureInfo = New CultureInfo("en-US")
```

Dann wird der Wert des angeklickten Elements des `RadioButtonList`-Steuerelementes mit `Wahrungen.SelectedItem.Value` ermittelt. Die Konvertierung erfolgt erst in den Typ `double` und dann wieder in eine Zeichenkette, diesmal unter Verwendung der Kultur »en-US«, sodass die Dezimaltrennzeichen den Bedingungen in SQL entsprechend: `ToString(us)`.

Dann wird eine neue Tabelle erzeugt – *Currency* –, die später die Werte aufnimmt, die der Benutzer eingibt:

```
Dim waehrungsTabelle As DataTable = New DataTable("Currency")
```

Es folgen die Definitionen der Spalten:

```
tempC = New DataColumn()
```

Die Tabelle enthält lediglich zwei Spalten: *Original* zur Aufnahme der Eingabewerte und *Euro* zur Ausgabe der berechneten Beträge. Die Originalwerte müssen eindeutig sein, deshalb wird eine `UNIQUE`-Einschränkung programmiert:

```
tempC.Unique = True
```

Die *Euro*-Spalte wird dagegen überhaupt nicht durch Eingaben gesteuert, sondern soll ihren Inhalt selbst berechnen. Dazu wird die Eigenschaft `Expression` gesetzt:

```
tempC.Expression = "Original / " & faktor
```

Die Variable *faktor* enthält den Umrechnungsfaktor für die gewählte Währung. Die Spalten werden dann der Tabelle hinzugefügt:

```
waehrungsTabelle.Columns.Add(tempC)
```

Dann wird die Tabelle an das *DataSet* übergeben:

```
ds.Tables.Add(waehrungsTabelle)
```

Das so erzeugte `DataSet`-Objekt darf nicht jedes Mal wieder generiert werden, denn sonst gehen die bereits erfassten Werte verloren. Andererseits enthalten die Daten nun nutzerspezifische Informationen, sodass sich eine zentrale Ablage in Applikationsvariablen verbietet. Für solche Daten sind Sitzungsvariablen gut geeignet.

Ausführliche Informationen dazu finden Sie im Abschnitt 8.4, »Sitzungen (Sessions)« ab Seite 778. Hier wird das fertige Objekt einfach in einer solchen Variablen gespeichert:

```
Session("MyDataSet") = ds
```

Nun ist es noch notwendig, die Eingabe im Textfeld zu verarbeiten. Dazu ist die bereits erwähnte Einschränkung zu beachten, die die Eindeutigkeit der Spalte *Original* sicher stellt. Eine *Try-Catch*-Anweisung regelt das und erzeugt gegebenenfalls eine Fehlermeldung.

```
Public Sub Absenden_OnClick(ByVal sender As Object, 
                                ByVal e As System.EventArgs) _
Handles Absenden.Click
    ds = CType(Session("MyDataSet"), DataSet)
    If Not ds Is Nothing Then
        Dim original() As Object = {Eingabe.Text}
        Fehler.Text = ""
        Try
            ds.Tables("Currency").Rows.Add(original)
        Catch ex As ConstraintException
            Fehler.Text = "Der Wert ist bereits vorhanden 
                        oder keine Zahl"
            Fehler.ForeColor = Color.Red
        End Try
        Ausgabe.DataSource = ds
        Ausgabe.DataMember = "Currency"
        Ausgabe.DataBind()
    End If
End Sub
```

Listing 9.29: Die Methode Absenden_OnClick (Ausschnitt aus DataSetLocalExpression.aspx.vb)

Wie es funktioniert

Das in der Methode *Waehrungen_Changed* erzeugte und in einer Sitzungsvariablen gespeicherte *DataSet* muss zuerst wieder bereit gestellt werden:

```
ds = CType(Session("MyDataSet"), DataSet)
```

Falls die Methode versehentlich aufgerufen wurde, ohne dass das *DataSet* existierte, ist die Variable *ds* *Nothing*. Dies wird geprüft, bevor die eigentliche Nutzung beginnt.

Da Tabellen normalerweise mehrere Spalten enthalten und diese beim Erzeugen einer neuen Reihe mit einem Mal zugewiesen werden sollen, erwartet die Methode *Add* der Kollektion *Rows* ein Array vom Typ *Object()*. Das ist auch der Fall, wenn wie vorliegend nur

eine Spalte bedient und die zweite berechnet wird. Das Array besteht nur aus einem Element, dem Wert des `TextBox`-Steuerelements:

```
Dim original() As Object = {Eingabe.Text}
```

Dieses Array wird dann als Parameter zum Erzeugen der neuen Reihe verwendet. Das passiert innerhalb eines *Try*-Zweiges, denn hier kann ein Fehler auftreten. Bei der Definition der Spalte wurde festgelegt, dass die Werte eindeutig sein müssen (`UNIQUE`). Wenn nun das Formular mit demselben Wert erneut abgesendet wird, tritt eine entsprechende Ausnahme bei der Ausführung der folgenden Zeile auf:

```
ds.Tables("Currency").Rows.Add(original)
```

Diese wird mit folgendem Code abgefangen:

```
Catch ex As ConstraintException
```

Im Beispielprogramm erscheint daraufhin eine entsprechende Meldung neben der Absende-Schaltfläche. Dies ist durchaus praktikabel, denn es handelt sich hier um keinen schwer wiegenden Fehler, der einen Programmabbruch rechtfertigen würde.

Zuletzt folgt noch die Ausgabe der Daten des *DataSet* an das *DataGrid*-Steuerelement.

Zusammenfassung

Das vollständige Programm finden Sie unter dem Namen *DataSet-LocalExpression.aspx.vb*. Es zeigt, wie Sie ein *DataSet*-Objekt ohne Datenbank erzeugen, mit Tabellen füllen und praktisch verwenden können. Gegenüber konventioneller Programmierung ist die Komplexität der Speicherung, Berechnung und Darstellung in ein leistungsfähiges Objekt integriert worden, was den Code schlanker und schneller macht. Auch bei steigenden Ansprüchen an das Produkt wird die Programmierung mit *DataSet* der beste Weg bleiben, denn es gibt weit mehr Funktionen und Einsatzfälle, als hier beschrieben werden konnten. Lassen Sie sich durch die beiden Beispiele mit datenbanklosen *DataSet*-Anwendungen inspirieren, eigene Ideen mit dieser Technik umzusetzen.

In den folgenden Abschnitten wird der »klassische« Weg näher betrachtet: Der Einsatz der Klasse *DataSet* mit einer Datenbank. Dazu kommt vorzugsweise der SQL Server und Transact-SQL zum Einsatz.

9.3.3 Der Datenadapter (DataAdapter)

Sollen Daten einer großen Benutzergruppe bereitgestellt werden oder sind die Datenmengen selbst sehr umfangreich, führt kaum ein Weg an einem Datenbankmanagementsystem vorbei. Wenn sich Daten optimal in Relationen darstellen lassen und Such- oder Filterfunktionen intensiv genutzt werden, bieten RDBMS immer noch klare Vorteile auch gegenüber XML. In diesem Abschnitt soll die Kopplung des *DataSet* mit der Datenbank im Vordergrund stehen. Schauen Sie sich dazu noch ein Mal Abbildung 9.3 an. Dort ging es um den Aufbau des Providers, also des datenbankabhängigen Teils von ADO.NET. Im Mittelpunkt der Abbildung stand der so genannte Datenadapter (*DataAdapter*), der jedoch beim Zugriff mit *DataReader* nicht benötigt wird. Der Datenadapter ist quasi der große Bruder des *DataReaders*.

Einführung in die Klasse *DataAdapter*

Der Datenadapter stellt eine Verbindung zwischen einer Datenbank und einem *DataSet* her. Dabei gibt es keine Einschränkungen in der Art des Zugriffs. Daten können gelesen und geschrieben werden, es gibt Transaktionen und Datensatzverriegelungen. Wie bei den anderen Bestandteilen eines Providers ist die Implementierung auch hier für jede Zugriffsmethode individuell, es gibt also einen *SqlDataAdapter* und einen vergleichbaren *OleDbDataAdapter*. Wenn im folgenden Text allgemein von *DataAdapter* oder dem Datenadapter die Rede ist, gelten die Ausführungen für beide Provider (und vermutlich auch für andere, wie *OdbcDataAdapter*, die hier nicht betrachtet werden).



*In den Beispielen wird durchgehend der SQL Server 2000 als Datenquelle verwendet, sodass Sie nur den Einsatz von *SqlDataAdapter* sehen. Es dürfte sehr einfach sein, das Prinzip auf *OleDb* zu übertragen. Leser, die nur MS Access verwenden können, sollten sich unbedingt dazu ermutigt fühlen, die Programme entsprechend zu verändern.*

Daten aus der Datenbank holen

Der erste Schritt bei der Nutzung des Datenadapters besteht in der Abfrage von Daten aus einer existierenden Datenbank. Das folgende Beispiel zeigt, wie dies erfolgt und wie die Übergabe der Daten an ein *DataSet* und nachfolgend an ein *DataGrid* (hier mit dem Namen *Artikel*) erfolgt. Ausführliche Informationen zu den

Daten-Steuerelementen finden Sie unter anderem im Abschnitt 6.6.5, »DataGrid« ab Seite 605.

```
Public Class DataAdapterSelect
    Inherits System.Web.UI.Page
    Protected Artikel As DataGrid

    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Page.IsPostBack Then
            Dim query As String = "SELECT * FROM Artikel"
            Dim connection As String = "Server=localhost; 
                Database=Shop; uid=sa; 
                pwd=password"

            Dim da As SqlDataAdapter = New SqlDataAdapter(query,
connection)
            da.MissingSchemaAction = MissingSchemaAction.AddWithKey
            Dim ds As DataSet = New DataSet()
            da.Fill(ds, "Artikel")
            Artikel.DataSource = ds
            Artikel.DataBind()
        End If
    End Sub
End Class
```

*Listing 9.30: Nutzung eines Datenadapters zur Abfrage einer Datenbank
(DataAdapterSelect.aspx.vb)*

Im Beispiel werden eine Datenbankabfrage (SELECT) und eine Verbindungszeichenfolge vorbereitet. Dann wird diese Abfrage ausgeführt, indem die entsprechenden Informationen dem Konstruktor des Datenadapters übergeben werden:

Wie es funktioniert

```
Dim da As SqlDataAdapter = New SqlDataAdapter(query, connection)
```

Nun wird zur Aufnahme der Daten ein *DataSet* erzeugt:

```
Dim ds As DataSet = New DataSet()
```

Es ist Aufgabe des Datenadapters, die Daten an das Ziel zu übergeben. In diesem Fall ist das Ziel das *DataSet*-Objekt und die Aufgabe der Übertragung übernimmt die Methode *Fill*:

```
da.Fill(ds, "Artikel")
```

Fill kennt mehrere Überladungen, von denen die gebräuchlichste eingesetzt wurde. Der erste Parameter ist das *DataSet*-Objekt, der zweite benennt die Tabelle, der die Daten zugewiesen werden. Die Tabelle selbst wird dabei erzeugt. Der Rest des Programms

realisiert die bereits mehrfach verwendete Methodik zur Ausgabe der Daten mit einem `DataGrid`-Steuerelement, was mangels aufwändiger Gestaltung zu einer primitiven Ausgabe führt.



Wenn Sie die Gestaltung sehr schnell vornehmen möchten, können Sie den Designer in Visual Studio .NET verwenden. Wechseln Sie dazu in die Entwurfsansicht und wählen Sie im Kontextmenü des `DataGrid`-Steuerelementes die Option AUTOMATISCHE FORMATIERUNG. Im folgenden Dialog wählen Sie eine Gestaltungsvorlage aus und klicken dann auf OK.

Mit ein wenig Gestaltung wirkt es schon professioneller, was angesichts der wenigen Programmzeilen erstaunlich ist, denn auch die `aspx`-Datei enthält weiterhin nur ein einziges Steuerelement.

DataAdapter			
Daten mit DataAdapter aus SQL Server holen			
id	name	preis	nummer
1	Windows XP Pro	199,99	1325
2	MSDN Abo Professional	2850	6668
3	Windows XP Home	139,99	1326
4	Big Color Keyboard	118,8	7738
5	SharpTemple Handbuch	499	8837
6	Werbegag	11,99	3456
7	Werbegag2	11,99	1234
8	Werbegag2	11,99	1234
9	Werbegag2	11,99	1234
10	Sonderanfertigung	19,99	9999

Abbildung 9.23: Kleine Ursache – große Wirkung: Die Artikelstabelle in ASP.NET

Fill Freilich steckt weit mehr im Datenadapter als dieses Beispiel zeigen konnte. Einiges bietet allein die Methode `Fill`. Hier folgt eine alternative Überladung:

```
da.Fill(ds, startDs, maxDs, "TabellenName")
```

Der Parameter `startDs` bestimmt den ersten zu lesenden Datensatz, während `maxDs` die Anzahl festlegt.

DataTable Abgesehen davon kann `Fill` auch direkt Daten an ein `DataTable`-Objekt übergeben, dass dann wiederum in der bereits gezeigten Art und Weise an ein `DataSet` weitergereicht wird.

Mit `FillSchema` können Sie außerdem veranlassen, dass nur die Struktur der abgefragten Tabelle übertragen wird. Dies ist dann interessant, wenn die Datenbank noch leer ist und erst durch Benutzeraktionen gefüllt werden soll.

FillSchema

Über die Arbeitsweise des Datenadapters und das Zusammenspiel mit `DataSet`

Wie im Beispiel zu sehen war, erzeugt der Datenadapter die Tabelle im `DataSet`, wenn es erforderlich sein sollte. Ist die Tabelle schon vorhanden, wird die bestehende Definition verwendet. Das setzt natürlich voraus, dass die Daten dazu passen. Leider ist die Methode der Generierung der Tabellenstruktur nicht vollautomatisiert. So werden Spalten standardmäßig nicht erkannt und der in der SQL-Tabellen zweifelsfrei vorhandene Primärschlüssel ist in der Tabelle *Artikel* im `DataSet`-Objekt verschwunden. Es gibt zwei Lösungen für das Problem:

- ▶ Nutzung eines typisierten `DataSet`s
- ▶ Programmierung von Instruktionen zum Umgang mit fehlenden Schematas

An dieser Stelle müssen Sie sich zwangsläufig ein wenig mit Schemata auseinander setzen. Einige Informationen finden Sie im Abschnitt »Eine sehr kurze Einführung in XSD« ab Seite 450. Denn das interne Speicherformat im `DataSet` ist XML. Die Strukturinformationen der Tabellen werden passend dazu in XSD (XML Schema Definition Language) gespeichert.

Wenn die Daten einer Abfrage nicht zum vorhandenen Schema passen, muss es eine Strategie geben, damit umzugehen. Allein das auslösen einer Ausnahme ist keine Lösung, denn Teilabfragen sind typisch, sodass häufiger dieser Vorgang auftritt. Die Klasse `SqlDataAdapter` bzw. `OleDbDataAdapter` bringt dazu eine Eigenschaft mit: `MissingSchemaAction`. Damit steuern Sie, wie mit fehlenden oder überzähligen Spalten und bestimmten Spalteneigenschaften verfahren wird, wenn das Füllen eines `DataSet` erfolgt. Der Eigenschaft kann eine der folgenden Aufzählungen zugewiesen werden.

Umgang mit vorhandenen Schematas

Option	Beschreibung
Add	Fügt zusätzliche Spalten hinzu, unabhängig davon, ob sie im Schema schon existieren. Dies ist die Standardeinstellung.
AddWithKey	Fügt zusätzliche Spalten hinzu und erkennt Primärschlüssel.
Error	Löst eine Ausnahme aus, wenn eine Spalte im Schema nicht existiert.
Ignore	Spalten, die nicht im Schema existieren, werden ignoriert.

Tabelle 9.3: Steuerung des Verhaltens des Datenadapters bei der Datenübergabe an ein DataSet

MissingSchemaAction

Diese Optionen wenden Sie folgendermaßen an:

```
da.MissingSchemaAction = MissingSchemaAction.AddWithKey
```

Die Grundlage derartiger Aktionen bildet natürlich immer ein Schema. Man spricht in diesem Fall von typisierten *DataSets*, solchen also, die wie eine Datenbank über die kompletten Typinformationen verfügen, wie sie auch im SQL Server oder in Access definiert werden können.

9.3.4 Datenansichten mit DataView

Auch *DataView* war schon kurz im Einsatz. Diese Klasse gehört nicht zum Provider und ist universell einsetzbar. Es ist damit möglich, eine bindungsfähige Sicht einer spezifischen Tabelle zu erstellen. Durch die Möglichkeit der Bindung kann ein *DataView*-Objekt genauso wie *DataSet* als Quelle eines Daten-Steuerelements genutzt werden. Der Einsatz besteht dann auch im Wesentlichen darin, die Daten einer Abfrage für eine konkrete Situation anzupassen. Diese Technik erleichtert die Gestaltung von SQL-Abfragen, weil diese gegebenenfalls einfacher ausfallen können. Sie ist außerdem flexibler, weil Sie viele verschiedene Sichten auf die Daten erzeugen können, ohne immer wieder den Datenbankserver damit zu belasten.

Die Auswahl von Daten basiert auf zwei elementaren Vorgängen:

- Sortieren
- Filtern

Darüber hinaus können Sie die Daten in der Sicht auch ändern, löschen oder ergänzen. Die dahinter liegende Tabelle wird dann ebenfalls aktualisiert und dies wird wiederum in der Datenbank repräsentiert.

Daten filtern und sortieren

Für das nächste Beispiel sollen die Filter- und Sortiermöglichkeiten der Klasse `DataGridView` herangezogen werden. Um die Zuordnung der zu programmierenden Methoden einfacher zu gestalten, machen Sie sich zuerst mit dem Code der *aspx*-Seite vertraut. Die Gestaltungsstile des `DataGrid`-Steuerelements wurden mit dem Visual Studio .NET-Designer erzeugt. Sie sind für die Funktion unbedeutend.

```
<h1>DataGridView</h1>
<h2>Daten mit DataAdapter holen und mit 
    DataGridView filtern/sortieren</h2>
<form runat="server">
    (<asp:CheckBox Runat="server" ID="SortierRichtung" 
        Text="Absteigend"/>)
    sortieren nach:
    <asp:RadioButtonList Runat="server" ID="SortierTyp" 
        RepeatDirection="Horizontal">
        <asp:ListItem Selected="True" Value="preis">Preis</asp:ListItem>
        <asp:ListItem Value="name">Name</asp:ListItem>
        <asp:ListItem Value="filter">Filter:</asp:ListItem>
    </asp:RadioButtonList>
    <asp:TextBox Runat="server" ID="Filter" Width="100px"/>
    (Filter-Option wählen)
    <asp:Button Runat="server" ID="Absenden" Text="Auswahl" 
        OnClick="Optionen_Changed"/>
</form>
<asp:DataGrid Runat="server" ID="Artikel" 
    BorderColor="#CC9966" 
    BorderStyle="None" BorderWidth="1px" 
    BackColor="White" CellPadding="4">
    <SelectedItemStyle Font-Bold="True" 
        ForeColor="#663399" 
        BackColor="#FFCC66">
    </SelectedItemStyle>
    <ItemStyle ForeColor="#330099" BackColor="White">
    </ItemStyle>
    <HeaderStyle Font-Bold="True" ForeColor="#FFFFCC" 
        BackColor="#990000"></HeaderStyle>
    <FooterStyle ForeColor="#330099" BackColor="#FFFFCC">
    </FooterStyle>
    <PagerStyle HorizontalAlign="Center" 
        ForeColor="#330099" 
        BackColor="#FFFFCC">
    </PagerStyle>
</asp:DataGrid>
```

Listing 9.31: Sortieren und Filtern mit `DataGridView` (*DataGridViewFilterSort.aspx*)

Erstellt wird eine Liste von Optionsfeldern, die die Auswahl der Spalte zulassen und als dritte Option das Filtern der Namen nach einem Bestandteil. Das `TextBox`-Steuerelement enthält den Filterwert. Mit dem Klick auf die Schaltfläche wird eine Methode `Optionen_Changed` aufgerufen, die noch erstellt werden muss.

DataView

Daten mit DataAdapter holen und mit DataView filtern/sortieren

(☒ Absteigend) sortieren nach:

☐ Preis
 ☒ Name
 ☐ Filter:

(Filter-Option wählen)

id	name	preis	nummer
1	Windows XP Pro2	199,99	1325
3	Windows XP Home	199,99	1326
10	Werbegag 4	1,99	9996
9	Werbegag 3	2,99	9997
7	Werbegag 2	4,99	9999
6	Werbegag 1	9,99	9998
5	SharpTemple	199,99	8837
25	Muster5	9,99	1000
22	Muster455	9,99	1000
2	MSDN Abo Professional	2228	6668
4	Big Color Keyboard	199,99	7738

Abbildung 9.24: Das Filterprogramm in Aktion

Zuvor ist ein Blick auf die Erstellung des `DataSet` angebracht. Das passiert in der Methode `Page_Load`:

```

Private Sub Page_Load(ByVal sender As Object,   

                                ByVal e As System.EventArgs)   

Handles MyBase.Load
    If Not Page.IsPostBack Then
        Dim query As String = "SELECT * FROM Artikel"
        Dim connection As String = "Server=localhost;   

                                Database=Shop; uid=sa;   

                                pwd=password"
        Dim da As SqlDataAdapter   

            = New SqlDataAdapter(query, connection)
        da.MissingSchemaAction = MissingSchemaAction.AddWithKey
        ds = New DataSet()
        da.Fill(ds, "Artikel")
        Artikel.DataSource = ds
        Artikel.DataBind()
    
```

```

        Session("ArtikelDs") = ds
    End If
End Sub

```

*Listing 9.32: Erzeugen des DataSet und Speicherung in einer Sitzungsvariablen
(Ausschnitt aus DataViewFilterSort.aspx.vb)*

Der Vorgang entspricht dem bereits im vorhergehenden Beispiel gezeigten Prinzip. Mit der Bindung an das `DataGrid`-Steuerelement *Artikel* wird nach der Abfrage der Datenbank sofort eine unsortierte und ungefilterte Liste erzeugt.

**Die einfache
Ausgabe**

Interessanter ist die Methode *Optionen_Changed*, die durch Anklicken der Schaltfläche ausgelöst wird:

```

Public Sub Optionen_Changed(ByVal sender As Object, 
                               ByVal e As System.EventArgs) _
Handles SortierRichtung.CheckedChanged
    ds = CType(Session("ArtikelDs"), DataSet)
    ViewState.Clear()
    If Not ds Is Nothing Then
        Dim dv As DataView = New DataView(ds.Tables("Artikel"))
        If SortierTyp.SelectedItem.Value.ToString() = "filter" Then
            dv.RowFilter = String.Format("name LIKE '%{0}%'",
Filter.Text)
        Else
            If SortierRichtung.Checked Then
                dv.Sort = SortierTyp.SelectedItem.Value.ToString() + "
DESC"
            Else
                dv.Sort = SortierTyp.SelectedItem.Value.ToString() + "
ASC"
            End If
        End If
        Artikel.DataSource = dv
        Artikel.DataBind()
    End If
End Sub

```

*Listing 9.33: Holen des DataSet und Sortieren bzw. Filtern über DataView
(Ausschnitt aus DataViewFilterSort.aspx.vb)*

Zuerst wird das `DataSet` aus der Sitzungsvariablen geholt:

**Wie es
funktioniert**

```
ds = CType(Session("ArtikelDs"), DataSet)
```

Dann wird ein `DataView`-Objekt erzeugt und mit der *Artikel*-Tabelle gefüllt.

```
Dim dv As DataView = New DataView(ds.Tables("Artikel"))
```

RowFilter Wenn die Option *filter* gewählt wurde, wird aus dem Inhalt des TextBox-Steuerelements eine Filteranweisung formuliert. Die Möglichkeiten, die hier bestehen, entsprechen denen der Datenbank, so wie es dort nach der *WHERE*-Klausel geschrieben wird. Sie können also beispielsweise einen Filter mit der SQL-Option *LIKE* entwerfen:

```
dv.RowFilter = String.Format("name LIKE '{0}%', Filter.Text)
```

Sort Alternativ zum Filtern kann auch sortiert werden. Die Spaltennamen wurden bereits als Werte der Optionsfelder verwendet, so dass die Filteranweisungen direkt erzeugt werden. Je nach Zustand des Kontrollkästchens wird außerdem *ASC* oder *DESC* angehängt. Auch entsprechen die Möglichkeiten denen von SQL nach der *ORDER BY*-Bedingung.

```
dv.Sort = SortierTyp.SelectedItem.Value.ToString() + " ASC"
```



Die Abfragesprache, die mit Sort und RowFilter verwendet wird, entspricht zwar SQL, wird aber nicht direkt an die Datenbank weitergegeben. Der Sprachumfang ist etwas geringer als von T-SQL vorgegeben. Tatsächlich wird auch nicht die Datenbank abgefragt, sondern es werden nur die lokalen Daten in der Datenansicht gefiltert.

Die Abfragesprache in RowFilter

Die Abfragesprache, die mit *RowFilter* verwendet werden kann, umfasst vier Bereiche:

- ▶ Operatoren
- ▶ Aggregat-Funktionen
- ▶ Allgemeine Funktionen
- ▶ Filterstatus-Eigenschaften

Operatoren Die Operatoren können Sie der folgenden Tabelle entnehmen:

Operator	Beschreibung
AND	Logische Verknüpfungen zwischen Teilausdrücken
OR	(Und, Oder, Nicht)
NOT	

Operator	Beschreibung
> < <= >= <> =	Bedingungsoperatoren zur Bildung von Ausdrücken
+ - * / %	Mathematische Operatoren zum Berechnen in Ausdrücken
IN	Spezifizierung einer Liste: IN (Wert1, Wert2, Wert3)
LIKE	Suchen von Teilen eines Ausdrucks mit Hilfe von Platzhalterzeichen (»%« für beliebige und »_« für ein beliebiges Zeichen)

Die Aggregat-Funktionen entsprechen denen von SQL, sind aber weniger umfangreich anzutreffen als in T-SQL:

Aggregat-Funktionen

Funktion	Beschreibung (Rückgabewert)
AVG	Average. Der Durchschnitt der Felder (INT, DECIMAL, MONEY oder FLOAT)
COUNT	Die Anzahl der Felder (INT)
SUM	Summary. Die Summe der Felder (Addition).
MAX	Maximum. Das Feld mit dem größten Wert bestimmt das Ergebnis.
MIN	Minimum. Das Feld mit dem kleinsten Wert bestimmt das Ergebnis.
STDEV	Statistische Standardabweichung aller Werte der Liste
VAR	Statistische Varianz

Tabelle 9.4: Aggregat-Funktionen, die mit DataView verwendet werden können

Wenn der Umfang für Ihre Anwendung nicht ausreicht, müssen Sie auf T-SQL ausweichen und die Abfrage bereits im Datenadapter formulieren.

Die allgemeinen Funktionen können zur Behandlung von speziellen Zuständen von Spalten herangezogen werden. Sie dienen

Allgemeine Funktionen

auch der Ermittlung von Eigenschaften und zur Typumwandlung. Eine Verwandtschaft mit Visual Basic kann nicht verleugnet werden¹:

Funktion	Beschreibung (Rückgabewert)
Convert (<i>wert</i> , <i>typ</i>)	Umwandlung eines Datentyps
Len (<i>ausdruck</i>)	Länge des Ausdrucks (Anzahl Zeichen)
IsNull (<i>ausdruck</i> , <i>ersatz</i>)	Wenn der Inhalt NULL ist, wird der Wert <i>ersatz</i> zurückgegeben
IIF (<i>ausdruck</i> , <i>true</i> , <i>false</i>)	Erstellt eine eingebettete Bedingung und gibt nach der Auswertung des Ausdrucks den Teil <i>true</i> oder <i>false</i> zurück.
SubString (<i>zeichen</i> , <i>start</i> , <i>laenge</i>)	Extrahiert eine Teilzeichenkette aus einer anderen

Tabelle 9.5: Allgemeine Umwandlungs- und Auswertefunktionen

Filterstatus-Eigenschaften

Die Filterstatus-Eigenschaften werden mit Hilfe der Eigenschaft `RowStateFilter` festgelegt. Die meisten Optionen sind nur sinnvoll einsetzbar, wenn die Reihen in der `DataGridView` verändert werden. Dazu finden Sie im Folgenden weitere Informationen. Angewendet wird die Eigenschaft folgendermaßen (am Beispiel `CurrentRows`):

```
dv.RowStateFilter = RowStateFilter.CurrentRows
```

Die zulässigen Werte finden Sie in der folgenden Tabelle:

Wert	Bedeutung
Added	Nur in dieser Ansicht hinzugefügte Reihen
CurrentRows	Aktuelle Reihen, also originale, hinzugefügt und geänderte, nicht jedoch gelöschte
Deleted	Gelöschte Reihen
ModifiedCurrent	Aktuelle Version einer veränderten Reihe
ModifiedOriginal	Originale (ursprüngliche) Version einer veränderten Reihe
None	Keine Optionen (Standardeinstellung)

Tabelle 9.6: Statuswerte der Aufzählung `RowStateFilter`

1 Tatsächlich handelt es sich um die Funktionen des alten VBA-Moduls. Basic-Cracks werden damit wenig Probleme haben.

Wert	Bedeutung
OriginalRows	Die ursprüngliche Reihe ohne Beachtung veränderter oder gelöschter Reihen
Unchanged	Alle unveränderten Reihen

Tabelle 9.6: Statuswerte der Aufzählung `RowStateFilter` (Forts.)

Die Optionen sind Bitfelder und können deshalb mit dem `Or`-Operator verknüpft werden, wenn dies sinnvoll ist.

Aktualisieren einer Datenbank mit Datenansichten

Wenn Sie innerhalb einer `DataView` Änderungen vornehmen, werden diese in der Datenbank repräsentiert. Die Programmierung ist meist einfacher als mit den originalen Daten, wenngleich auch hier geringfügige Einschränkungen gegenüber Transact-SQL akzeptiert werden müssen. Auch deshalb sollten Sie T-SQL nicht aus den Augen verlieren. In vielen Fällen ist der von `DataView` vorgegebene Weg jedoch einfacher.

Bevor Sie die ersten Schritte mit einer Aktualisierung gehen, sollten Sie sich die Möglichkeiten der vorlagenorientierten Programmierung mit `DataGrid` ansehen, die vieles vereinfacht. Genauer wird im Abschnitt 6.6, »Vorlagengebundene Daten-Steuerelemente« ab Seite 566 darauf eingegangen. Andererseits erfüllt das `DataGrid` nie alle Ansprüche, es ist deshalb gut zu wissen, wie Aktualisierungen direkter programmiert werden.



Das nächste Beispiel nutzt eine Datensicht, um alle Einträge einer Tabelle zu bearbeiten oder einen zu löschen bzw. hinzuzufügen. Es arbeitet ein wenig intuitiver als ein `DataGrid`, denn zum Editieren genügt ein Klick auf die Tabellenreihe. Die Beschreibung erfolgt im nächsten Abschnitt zur Klasse `CommandBuilder`, denn beide Techniken werden dort parallel eingesetzt.

Allerdings ist der interne Vorgang durch Manipulation des `DataView`-Objekts nicht alles. Schließlich müssen die Daten wieder in die Datenbank. Das geschieht natürlich mit SQL-Befehlen. Damit Sie sich nicht zu viel Gedanken darüber machen müssen, wie das bei einfachen Beispielen abläuft, hilft die Klasse `CommandBuilder` bei der Erstellung.

9.3.5 Aktualisieren einer Datenbank mit CommandBuilder

Im letzten Abschnitt wurde bereits angedeutet, dass es einen Weg gibt, die Änderungen, die über eine Datenansicht vorgenommen wurden, zurück in die Datenbank zu schreiben. Natürlich können Sie direkt SQL-Befehle erzeugen und diese an die Datenbank senden. Bei komplexen Abfragen ist dies auch unumgänglich – gute SQL-Kenntnisse sind weiterhin notwendig. Bei einfachen Aufgabenstellungen bietet ADO.NET jedoch einige Verbesserungen, unter anderem durch automatisiertes Erzeugen der Basisbefehle für Datenbanken: UPDATE, INSERT und DELETE).



Das folgende Programm ist relativ umfangreich. Es wird deshalb nicht erst vollständig und dann in Teilen gezeigt. Es wird aber zwischen den Erklärungen vollständig abgedruckt. Wenn Sie es abtippen, fügen Sie alle Codes in diesem Abschnitt zusammen. Das fertige Programm finden Sie auf CD oder im Web unter dem Namen DataViewEditor.aspx bzw. DataViewEditor.aspx.vb.

Die Vorlage

Die eigentliche Arbeit wird in der Code-Datei erledigt, entsprechend beschränkt sich der HTML-Teil auf wenige Steuerelemente. Außerdem soll diesmal keines der fertigen Daten-Steuerelemente verwendet werden.

```
<%@ Page language="vb" ⌵
    Codebehind="DataViewEditor.aspx.vb" ⌵
    AutoEventWireup="false" ⌵
    Inherits="Addison.VBNet.Data.DataViewEditor" %>
<html>
  <head>
    <title>DataViewEditor</title>
  </head>
  <body MS_POSITIONING="GridLayout">
    <h1>DataView</h1>
    <h2>Editieren von Daten mit DataView</h2>
    <form id="DataViewEditor" method="post" runat="server">
      <input type="hidden" runat="server" ⌵
        id="FeldNummer" value="-1"/>
      <asp:Table Runat="server" ID="ArtikelTabelle" ⌵
        Font-Name="Verdana"/>
      <asp:Button Runat="server" ID="ArtikelHinzu" ⌵
```

```
Text="Neuer Artikel" OnClick="Artikel_Add"/>
<br/>
<asp:Label Runat="server" ID="Fehler"
BackColor="Red" />
</form>
</body>
</html>
```

Listing 9.34: Vorlage für Experimente mit CommandBuilder (DataViewEditor.aspx)

Im Prinzip wird eine leere Tabelle mit `<asp:table>` definiert, die später die Datensätze aufnimmt. Die Schaltfläche *ArtikelHinzu* dient dem Anfügen von Datensätzen. In *Fehler* erscheinen gegebenenfalls Fehlermeldungen.

Aufgabe des Programms ist es, die *Artikel*-Tabelle zu bearbeiten. Dazu gehören allgemein drei Funktionen:

- ▶ Ändern
- ▶ Löschen
- ▶ Hinzufügen

Die Abbildung zeigt den Bearbeitungsmodus der Tabelle:

DataView

Editieren von Daten mit DataView

ID	Artikelname	Preis	Artikelnummer
1	Windows XP Pro2	199,99	1325
2	MSDN Abo Professional	2850	6668
3	Windows XP Home	199,99	1326
4	Big Color Keyboard	199,99	7738
5	SharpTemple	199,99	8837
6	Werbegag 1	9,99	9998
7	Werbegag 22	4,99	9999
 Werbegag 3		2,99	9997
10	Werbegag 4	1,99	9996
22	Muster4	9,99	1000
25	Muster5	9,99	1000

Neuer Artikel

Abbildung 9.25: Ausgabe der Tabelle; eine Zeile ist im Bearbeitungsmodus

Die Motivation für dieses Programm lag in zwei Aspekten: Zum einen sollte gezeigt werden, dass auch beim Verzicht auf *DataGrid* eine ansprechende Lösung mit wenigen Zeilen Code möglich ist. Zum anderen sollte eine Funktion realisiert werden, die *DataGrid* nicht bietet. Die Auswahl der Zeilen ist nämlich über das Java-

Motivation

Script-Ereignis `onMouseOver` möglich. Ein Klick in die Zeile führt zum Sprung in den Bearbeitungsmodus. Das ist intuitiver als eine ständig sichtbare Schaltfläche.

Die Code-Datei

DataViewEditor.aspx.vb

Die Code-Datei beginnt mit den zusätzlich nötigen *Imports*-Anweisungen. Anhand der Zeile `Imports System.Data.SqlClient` ist erkennbar, dass der SQL Server als Datenquelle dienen soll. Nun beginnt die Definition der Klasse *DataViewEditor*, abgeleitet von *Page* und mit einigen öffentlichen Feldern. Die Abfrage der Artikel-tabelle wird fest vorgegeben, alle anderen SQL-Befehle werden später von ADO.NET automatisch erzeugt.

Eingesetzt wird zum Datenzugriff der *SqlDataAdapter*, zum Speichern der Daten die Klasse *DataSet* und zur Steuerung des aktuellen Anzeigezustands *DataView*.

```
Public Class DataViewEditor
    Inherits System.Web.UI.Page

    Protected WithEvents ArtikelTabelle As Table
    Protected FeldNummer As HtmlInputHidden
    Protected Fehler As Label

    Private tr As TableRow
    Private tc As TableCell

    Dim query As String = "SELECT * FROM Artikel"
    Dim connection As String = "Server=localhost; ↵
                                Database=Shop; uid=sa; ↵
                                pwd=password"

    Private ds As DataSet
    Private dv As DataView
    Protected WithEvents ArtikelHinzu As Button
    Private da As SqlDataAdapter
```

AddHeader

Da in diesem Beispiel kein fertiges Steuerelement verwendet wird, muss die komplette Tabelle selbst erzeugt werden. Das ist Dank des Web Server-Steuerelements *Table* nicht besonders schwierig. Die Methode *AddHeader* fügt die Kopfreihe hinzu, wobei die Beschriftungstexte hier fest vergeben werden. Als Rückgabewert wird eine Reihe vom Typ *TableRow* übergeben.

```

Private Function AddHeader() As TableRow
    tr = New TableRow()
    tc = New TableCell()
    tc.Text = "ID"
    tr.Cells.Add(tc)
    tc = New TableCell()
    tc.Text = "Artikelname"
    tr.Cells.Add(tc)
    tc = New TableCell()
    tc.Text = "Preis"
    tc = New TableCell()
    tc.Text = "Artikelnummer"
    tr.Cells.Add(tc)
    tr.BackColor = Color.LightBlue
    Return tr
End Function

```

Nun muss die Methode *GenerateTable* programmiert werden. Hiermit wird die komplette Tabelle erzeugt. Die Methode erwartet ein *DataRow*-Objekt, dessen Inhalt dargestellt werden soll sowie die Nummer der Reihe, die bearbeitet werden soll. Ist diese Angabe nicht erforderlich, wird -1 übergeben:

GenerateTable

```

Private Sub GenerateTable(ByVal dv As DataRow, ByVal EditRow As
Integer)
    Dim ht As HtmlInputText
    Dim b As Button
    Dim r As DataRow

    ArtikelTabelle.Controls.Clear()
    ArtikelTabelle.GridLines = GridLines.Both
    ArtikelTabelle.Rows.Add(AddHeader())

```

Da die Tabelle mehrfach aufgebaut wird, werden die zuvor definierten Steuerelemente mit *Controls.Clear* gelöscht. Es folgen Gestaltungseigenschaften, wobei hier nur exemplarisch das Tabellengitter aktiviert wird. Dann wird *AddHeader* aufgerufen, um den Kopf zu erzeugen.

Im nächsten Schritt werden nun alle Zeilen der *DataRow dv* durchlaufen, jede Reihe *r* entnommen und untersucht. Es kann vorkommen, dass eine Reihe bereits als gelöscht gekennzeichnet ist, diese wird dann von der Darstellung ausgenommen. Solche Untersuchungen nutzen die Eigenschaft *RowState*. Der Bearbeitungsmodus ist in zwei Fällen notwendig: Entweder ist tatsächlich eine Reihe zum Bearbeiten ausgewählt worden, dann ist *EditRow* ungleich -1. Wenn dagegen eine Reihe eingefügt wurde, ist der Status

dieser Zeile `DataRowState.Added`. Gelöschte Reihen müssen ebenfalls ausgeblendet werden. Da solche Reihen auch mitten im Zyklus auftreten können, erfolgt ein etwas radikaler Abbruch mit `GoTo`:

```
For i = 0 To dv.Table.Rows.Count - 1
Restart:
    r = dv.Table.Rows(i)
    tr = New TableRow()
    If r.RowState = DataRowState.Deleted Then
        i = i + 1
        GoTo Restart
    End If
    If i = EditRow Or r.RowState = DataRowState.Added Then
```

Jede Reihe besteht aus Zellen. Diese werden auf Basis der `DataColumns`-Kollektion erstellt, die aus der Eigenschaft `Columns` entnommen wird. Eine Besonderheit stellt die Spalte *id* dar, in der statt des Eingabefeldes zum Ändern die Schaltflächen eingeblendet werden:

```
For Each c As DataColumn in r.Table.Columns
    tc = New TableCell()
    if c.ColumnName <> "id" Then
```

Normale Spalten (alles außer *id*) werden im Änderungsmodus mit einem Eingabefeld dargestellt, erzeugt aus `HtmlInputText`. Die Methode `Add` fügt das Feld in die Tabellenzelle ein:

```
For Each c In r.Table.Columns
    tc = New TableCell()
    If c.ColumnName <> "id" Then
        ht = New HtmlInputText("text")
        ht.ID = c.ColumnName + i.ToString()
        ht.Value = r(c).ToString()
        ht.EnableViewState = False
        tc.Controls.Add(ht)
    Else
```

Ereignishandler zuweisen

Die ID-Spalte soll im Bearbeitungsmodus die beiden Schaltflächen enthalten. Beide werden durch Ereignisbehandlungsmethoden bedient. Der Typ ist `Button`, ein Web Server-Steuerelement. Die Zuweisung der Ereignisbehandlungsmethode mit `AddHandler` erfolgt an den Ereignishandler `Command`. Übergeben werden dem Kommando außerdem ein Kommandoname (`CommandName`), der zwischen »Add« und »Edit« unterscheidet. In `CommandArgument` wird die Nummer der Reihe übergeben.


```

        b = New Button()
        b.Text = "OK"
        b.BackColor = Color.Green
        AddHandler b.Command, AddressOf Artikel_Edit
        b.CommandName = IIf(r.RowState = DataRowState.Added, "Edit",
"Add")
        b.CommandArgument = i.ToString()
        tc.Controls.Add(b)
        b = New Button()
        b.Text = "Del"
        b.BackColor = Color.Red
        AddHandler b.Command, AddressOf Artikel_Delete
        b.CommandName = "FeldNummer"
        b.CommandArgument = i.ToString()
        tc.Controls.Add(b)
    End If

```

Nun werden die »normalen« Zellen erzeugt, also alle anderen, die sich nicht im Bearbeitungsmodus befinden. Damit die Reaktion auf `onMouseOver` erfolgt, ist JavaScript erforderlich. Dies wird der Tabellenreihe `tr` als Attribut-Parameter übergeben. Die einzige Aufgabe besteht darin, das Formular sich selbst aufrufen zu lassen und die Nummer der Reihe zu übergeben. Die Übergabe der Nummer erfolgt mit einem versteckten Feld, das in der HTML-Seite fest definiert wurde. Es steht als HTML Server-Steuerelement `FeldNummer` zur Verfügung. Das Senden des Formulars beschränkt sich auf den Aufruf der `submit()`-Methode in JavaScript; das korrekte Ziel hat ASP.NET eingetragen, weil das Formular mit `runat="server"` gekennzeichnet ist. Die übrigen Attribute dienen ebenfalls der Maussteuerung:

Auf Maus reagieren

```

        tr.Cells.Add(tc)
        Next
    Else
        tr.Attributes("onClick") ←
        =
        String.Format("document.DataViewEditor.FeldNummer.value='{0}'; ←
            document.DataViewEditor.submit ();", i.ToString())
        tr.Attributes("onMouseOver") ←
        = "this.style.cursor='hand'; ←
            this.style.backgroundColor='red';"
        tr.Attributes("onMouseOut") ←
        = "this.style.backgroundColor='white'"
    End If

```

Nach der Programmierung der interaktiven Steuerung der Reihe werden die Tabellenzellen erzeugt. Hier wird einfach der Inhalt

Die Reihen hinzufügen

der Datentabelle den Zellen als Text zugewiesen. Die fertige Reihe wird dann an die Tabelle angefügt:

```
Dim c As DataColumn
For Each c In r.Table.Columns
    tc = New TableCell()
    tc.Text = r(c).ToString()
    tr.Cells.Add(tc)
Next
End If
ArtikelTabelle.Rows.Add(tr)
Next
End Sub
```

Page_Load Nach den Hilfsmethoden zum Aufbau der Anzeige beginnt das eigentliche Programm, wie üblich in `Page_Load`. Beim Start der Applikation wird versucht, den Datenadapter aus einer Applikationsvariablen auszulesen. Falls er nicht existiert (da `Is Nothing`), wird ein neuer erzeugt und zugewiesen. Der Trick daran: Es existiert nur ein Datenadapter für alle Benutzer und der erste Benutzer, der die Applikation startet, erzeugt ihn. Die Steuerung der Anzeige erfolgt dann individuell in jeder einzelnen Sitzung mit Sitzungsvariablen und der Datenansicht. Diese Strategie reduziert die Datenbankzugriffe auf das unbedingt notwendige Minimum. Ganz unkritisch ist sie dennoch nicht, aber auf mögliche Probleme und deren Lösung wird später eingegangen.

Datenadapter Vorerst ist ein Blick auf den Datenadapter hilfreich. Nach dem Anlegen des Adapters, basierend auf der Standardabfrage `SELECT * FROM Artikel` wird der Parameter `AddWithKey` gesetzt. Die automatische Verwaltung der Änderungen in der Datenansicht funktioniert nur, wenn ein Primärschlüssel existiert. Dieser Parameter übernimmt den Schlüssel aus der Datenbank. Dann wird ein `SqlCommandBuilder`-Objekt erzeugt. Der Konstruktor fügt dem Datenadapter *da* automatisch generierte SQL-Befehle hinzu. Damit reicht später der Zugriff auf die abgeleitete Datenansicht.

```
Private Sub Page_Load(ByVal sender As Object, 
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    da = CType(Application("ArtikelDa"), SqlDataAdapter)
    Page.EnableViewState = False
    If da Is Nothing Then
        da = New SqlDataAdapter(query, connection)
        da.MissingSchemaAction = MissingSchemaAction.AddWithKey
```

```
Dim sc As SqlCommandBuilder = New SqlCommandBuilder(da)
Application("ArtikelDa") = da
```

Das *DataSet*, das die Daten der Abfrage aufnimmt, ist natürlich für jede Sitzung individuell – die Synchronisierung passiert erst später. Es wird beim ersten Abruf der Seite erzeugt, wenn die Eigenschaft *IsPostBack* gleich *False* ist. Gleichzeitig wird auch die Datenansicht erstellt – hier einmal ganz einfach ohne Sortierung oder Filterung. Sie können dies aber leicht ergänzen. Mit diesen Daten wird dann *GenerateTable* aufgerufen.

DataSet

```
End If
If Not Page.IsPostBack Then
    ds = New DataSet()
    da.Fill(ds, "Artikel")
    Session("ArtikelDs") = ds
    dv = New DataView(ds.Tables("Artikel"))
    GenerateTable(dv, -1)
Else
```

In allen anderen Fällen wird versucht, das *DataSet* aus der Sitzungsvariablen *ArtikelDs* zu entnehmen. Es ist außerdem beim Zurücksenden des Formulars festzustellen, welcher Wert im versteckten Feld *FeldNummer* steht. Dies zeigt die zu bearbeitende Zeile an. Falls die Auswahl erfolgte, obwohl noch kein *DataSet* existiert, wird dieses erzeugt. Das passiert, wenn jemand das Formular einer früheren Sitzung speichert.

```
ds = CType(Session("ArtikelDs"), DataSet)
If Not ds Is Nothing Then
    dv = New DataView(ds.Tables("Artikel"))
    GenerateTable(dv, Int32.Parse(FeldNummer.Value))
Else
    ds = New DataSet()
    da.Fill(ds, "Artikel")
    Session("ArtikelDs") = ds
    dv = New DataView(ds.Tables("Artikel"))
    GenerateTable(dv, -1)
End If
End If
End Sub
```

Wenn mehrere Benutzer auf dieselbe Tabelle zugreifen, kann es zu Problemen beim Löschen oder Schreiben von Daten kommen. Das passiert immer dann, wenn ein Datensatz verändert wird, der sich bereits im Bearbeitungszustand befindet. Auf das konkrete Problem wird noch eingegangen. Hier wird erst die Lösung prä-

SynchAnchor

sentiert: Tritt der Fehler auf, wird dem Benutzer ein Link angezeigt, der die Tabelle mit der Datenbank synchronisiert. Die folgende Funktion erzeugt das entsprechende Steuerelement:

```
Private Function SynchAnchor() As HtmlAnchor
    Dim a As HtmlAnchor = New HtmlAnchor()
    a.InnerText = "Zugriffsfehler: Neu ↩
                  synchronisieren erforderlich"
    a.HRef = Request.RawUrl
    Return a
End Function
```

Artikel_Add Nun beginnen die Ereignisbehandlungsmethoden. Zuerst *Artikel_Add*, mit der der Tabelle eine neue Reihe hinzugefügt wird. Im Kern der Funktion wird die Datenansicht *dv* erzeugt und mit *AllowNew* die Erlaubnis erteilt, neue Reihen anzufügen. Mit *AddNew* wird dann eine neue Reihe erzeugt und anschließend mit Standardwerten gefüllt. *EndEdit* beendet den Bearbeitungszustand. Zu diesem Zeitpunkt existiert die neue Reihe also schon. Wenn nun die Tabelle erzeugt wird, erkennt die Methode *GenerateTable*, die hinzugefügte Reihe und bietet sie zur Bearbeitung an.

```
Public Sub Artikel_Add(ByVal sender As Object, ↩
                      ByVal e As EventArgs)
    ds = CType(Session("ArtikelDs"), DataSet)
    da = CType(Application("ArtikelDa"), SqlDataAdapter)
    If Not ds Is Nothing Then
        dv = New DataView(ds.Tables("Artikel"))
        dv.AllowNew = True
        Dim dvr As DataRowView = dv.AddNew()
        dvr("name") = ""
        dvr("preis") = 9.99
        dvr("nummer") = 1000
        dvr.EndEdit()
        GenerateTable(dv, -1)
    End If
End Sub
```

Artikel_Delete Vergleichbar arbeitet *Artikel_Delete* beim Löschen von Zeilen. Hier können nun erstmals Synchronisationsprobleme auftreten. Es könnte passieren, dass in einer Sitzung ein Datensatz gelöscht wurde, während er in einer anderen bearbeitet wird. ADO.NET erzeugt dann eine Ausnahme vom Typ *DBConcurrencyException*. Dies passiert beim Aufruf der Methode *Update*, wenn die Anweisung ausgeführt wird. Um die Transaktion gültig zu machen, wird danach noch *AcceptChanges* aufgerufen. Das funktioniert aber nur, wenn kein Feh-

ler auftrat. Im Programm wird beim Auftreten der Ausnahme der Link zum Synchronisieren aufgerufen. Das ist sicher nicht praxistauglich, sondern soll der Demonstration dienen. Sie können so den Fehler bewusst auslösen und so lernen, wann er auftritt und gegebenenfalls eigene Lösungsstrategien entwerfen. In diesem Zusammenhang lohnt auch ein Blick auf die Eigenschaft `RowStateFilter` des `DataRowView`-Objekts, mit dem eine Filterung nach den Stati der Reihen möglich ist. Sie könnten so für eine Sicherheitsabfrage die zu löschenden oder zu ändernden Reihen nochmals getrennt ausgeben und damit dem Benutzer ein möglicherweise unbeabsichtigtes Verändern der Originaldaten anzeigen.

```
Public Sub Artikel_Delete(ByVal sender As Object, 
                        ByVal e As CommandEventArgs)
    ds = CType(Session("ArtikelDs"), DataSet)
    da = CType(Application("ArtikelDa"), SqlDataAdapter)
    Dim currentRow As Integer = Convert.ToInt32(e.CommandArgument)
    If Not ds Is Nothing Then
        dv = New DataView(ds.Tables("Artikel"))
        dv.AllowDelete = True
        dv.Delete(currentRow)
        Try
            da.Update(ds, "Artikel")
            ds.AcceptChanges()
        Catch
            Fehler.Controls.Add(SynchAnchor())
            ds.RejectChanges()
        Finally
            Session("ArtikelDs") = ds
        End Try
        GenerateTable(dv, -1)
    End If
End Sub
```

Die Methode *Artikel_Edit* dient der Bearbeitung der Daten einer Reihe. Auch hier wird lediglich das `DataView`-Objekt geändert. Mit dem Aufruf von `Update` wird dann der passende SQL-Befehl dem `DataAdapter` entnommen (das dort von `CommandBuilder` erzeugt wurde) und an die Datenbank gesendet. Auch hier kann die beschriebene Situation auftreten, dass ein Datensatz geändert wird, der sich gerade bei einem anderen Benutzer in Bearbeitung befand. Mangels Datensatzverriegelung – die Daten sind im `DataSet` losgelöst – ist auch hier eine Ausnahme vom Typ `DBConcurrencyException` zu erwarten.

Artikel_Edit

Die Methode *Artikel_Edit* unterscheidet sich auch an anderer Stelle etwas von der vorhergehend beschriebenen. Die Übernahme der Feldwerte erfolgt – ganz klassisch – mit `Request.Form` direkt aus den POST-Daten des Formulars. Die Felder, die dynamisch erzeugt wurden, werden nämlich nicht Bestandteil der Kollektion, denn sie werden erst zur Laufzeit hinzugefügt. Das Objektmodell der HTML-Seite entsteht aber schon vorher. Man kann zwar theoretisch die gesamte Statusverwaltung nachprogrammieren, aber dann hat man praktisch ein Steuerelement geschrieben. Das ist jedoch nicht die Aufgabe dieses Kapitels. Die Vereinfachung ist durchaus zulässig und praktikabel.

```
Public Sub Artikel_Edit(ByVal sender As Object, 
                        ByVal e As CommandEventArgs)
    ds = CType(Session("ArtikelDs"), DataSet)
    da = CType(Application("ArtikelDa"), SqlDataAdapter)
    Dim currentRow As Integer = Convert.ToInt32(e.CommandArgument)
    If Not ds Is Nothing Then
        If e.CommandName = "Add" Then
            da.Update(ds, "Artikel")
            ds.AcceptChanges()
        End If
        dv = New DataView(ds.Tables("Artikel"))
        dv.AllowEdit = True
        dv(currentRow)("name") 
            = Request.Form("name" + currentRow.ToString())
        dv(currentRow)("preis") 
            = Request.Form("preis" + currentRow.ToString())
        dv(currentRow)("nummer") 
            = Request.Form("nummer" + currentRow.ToString())
        Try
            da.Update(ds, "Artikel")
            ds.AcceptChanges()
        Catch
            Fehler.Controls.Add(SynchAnchor())
            ds.RejectChanges()
        Finally
            Session("ArtikelDs") = ds
        End Try
        GenerateTable(dv, -1)
    End If
End Sub
```

Falls sie etwas vermissen: Die in einer derartigen Anwendung typischen SQL-Befehle wie `UPDATE` oder `INSERT` werden tatsächlich vom `CommandBuilder` erzeugt.

Weitere Informationen zu Command und CommandBuilder

In der Praxis wird es natürlich vorkommen, dass solche einfachen automatisch erzeugten SQL-Befehle nicht ausreichen. Generell ist es immer eine gute Idee, einiges an komplexer Datenbankarbeit in gespeicherte Prozeduren zu verlagern. Sie können diese ebenso wie einfache SQL-Befehle dann als Kommandos vorgeben und so mit einem Einfügevorgang auch mehrere Tabellen bedienen. Voraussetzung sind natürlich gute Kenntnisse in Transact-SQL und im besonderen über gespeicherte Prozeduren. Informationen dazu finden Sie im Abschnitt 5.5, »Transact-SQL – SQL mit dem SQL Server 2000 lernen« ab Seite 376. Gespeicherte Prozeduren haben mehrere Vorteile:

- ▶ Jeglicher SQL-Code wird aus der Applikation in die Datenbank verlagert.
- ▶ Durch die Kompilation in der Datenbank ist die Ausführung schneller.
- ▶ Die Anwendungs- und die Datenbankentwicklung kann parallel stattfinden.

Die eigentlichen Kommandos werden vom `CommandBuilder` nur erzeugt. Die Ablage erfolgt in den entsprechenden Eigenschaften des `Datenadapters`: `InsertCommand` zum Einfügen, `UpdateCommand` zum Ändern, `DeleteCommand` zum Löschen und natürlich `SelectCommand` zur Abfrage. Um diese Eigenschaften zu füllen, muss ein `Command`-Objekt zugewiesen werden. Die Besonderheit: Sie können mit Parametern arbeiten, die erst im Augenblick der Ausführung gefüllt werden. Ein solches Kommando entsteht folgendermaßen:

SqlCommand

```
Dim sc As SqlCommand = New SqlCommand()
```

Dann wird der Text definiert, der die Basis des SQL-Befehls darstellt:

```
sc.CommandText = "BestellterArtikel"
```

Anzugeben ist noch der Typ, hier also eine gespeicherte Prozedur:

```
sc.CommandType = CommandType.StoredProcedure
```

Etwas aufwändiger ist die Definition der Parameter. Dazu wird jeweils ein Objekt vom Typ `SqlParameter` benötigt. Wie das genau funktioniert, zeigt das kleine Programm in Listing 9.35.



Wenn Sie statt dem SQL Server einen Datenbankzugriff über OleDb vornehmen, werden die adäquaten Klassen mit dem Präfix »OleDb« statt »Sql« verwendet. Einziger Unterschied: Bei den Parametern im Kommando erfolgt die Angabe nicht mit @name, sondern durch das Parametersymbol »?«. Die Zuweisung erfolgt fortlaufend in der Reihenfolge der Definition.

In dem folgenden Beispiel wird eine gespeicherte Prozedur verwendet, die schon früher eine Rolle spielte (siehe Listing 9.20): *BestellterArtikel*. Sie gibt Auskunft über die Anzahl der Bestellungen für einen bestimmten Artikel:

```
CREATE PROCEDURE BestellterArtikel
    @ArtikelNummer BIGINT
AS
SELECT b.[a-id],
       SUM(b.menge) AS menge,
       SUM(b.summe) AS summe,
       a.name AS name,
       a.preis AS einzelpreis
FROM Bestellungen b
JOIN Artikel a
ON a.id = b.[a-id]
AND b.[a-id] = @ArtikelNummer
GROUP BY b.[a-id], a.name, a.preis
```

Zwei Steuerelemente dienen der Bedienung. Ein *DropDownList*-Steuerelement bietet alle Artikelnamen an und nach der Auswahl durch den Benutzer sorgt ein *DataGrid* für die Darstellung:

```
<body>
<h1>DataAdapter, Command</h1>
<h2>Nutzung gespeicherter Prozeduren</h2>
Wählen Sie eine Artikelnummer, um die Bestellungen dazu zu sehen:
<form id="DataAdapterCommand" method="post" runat="server">
    <asp:DropDownList Runat="server" ID="ArtikelNummer"
        AutoPostBack="True"/>
</form>
<asp:DataGrid Runat="server" ID="Bestellungen"/>
</body>
```

Listing 9.35: Vorlage für das Abfrageprogramm (DataAdapterCommand.aspx)

Das führt dann je nach Artikel und Inhalt der Bestelltabelle zu etwa folgender Ausgabe:

DataAdapter, Command

Nutzung gespeicherter Prozeduren

Wählen Sie eine Artikelnummer, um die Bestellungen dazu zu sehen:

Windows XP Pro2

a-id	menge	summe	name	einzelpreis
1	5	748,99	Windows XP Pro2	199,99

Abbildung 9.26: Typische Ausgabe des Programms

Der gesamte Code steckt in der Page_Load-Methode:

```
Public Class DataAdapterCommand
    Inherits System.Web.UI.Page
    Protected ArtikelNummer As DropDownList
    Protected Bestellungen As DataGrid

    Private sconn As SqlConnection
    Private da As SqlDataAdapter
    Private ds As DataSet
    Private sc As SqlCommand
    Private sp As SqlParameter

    Private Sub Page_Load(ByVal sender As Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        If Not Page.IsPostBack Then
            sconn = New SqlConnection("Server=localhost;
                                     Database=Shop; uid=sa;
                                     pwd=password")

            sconn.Open()
            da = New SqlDataAdapter()
            ds = New DataSet("Bestellungen")
            sc = New SqlCommand()
            sc.CommandType = CommandType.Text
            sc.CommandText = "SELECT id, name FROM Artikel"
            sc.Connection = sconn
            Dim dr As SqlDataReader = sc.ExecuteReader()
            ArtikelNummer.DataSource = dr
            ArtikelNummer.DataTextField = "name"
            ArtikelNummer.DataValueField = "id"
            ArtikelNummer.DataBind()
            dr.Close()

            sc.CommandText = "BestellterArtikel" ' Stored Proc
            sc.CommandType = CommandType.StoredProcedure
            sp = New SqlParameter()
            sp.ParameterName = "@ArtikelNummer"
```

```

        sp.DbType = DbType.Int32
        sp.Direction = ParameterDirection.Input
        sc.Parameters.Add(sp)
        da.SelectCommand = sc
        Session("Adapter") = da
    Else
        da = CType(Session("Adapter"), SqlDataAdapter)
        If Not da Is Nothing Then
            ds = New DataSet()
            da.SelectCommand.Parameters("@ArtikelNummer").Value =
                = ArtikelNummer.SelectedItem.Value
            da.Fill(ds)
            Bestellungen.DataSource = ds
            Bestellungen.DataBind()
        End If
    End If
End Sub
End Class

```

Listing 9.36: Die Code-Datei (DataAdapterCommand.aspx.vb)

Wie es funktioniert

Der erste Teil nutzt einen schnellen *DataReader*, um die Auswahl-liste zu füllen. Das hier bereits verwendete *Command*-Objekt wird nun zur Nutzung der gespeicherten Prozedur eingesetzt. Als Kommandotext wird der Name der Prozedur genannt:

```
sc.CommandText = "BestellterArtikel"
```

Als Typ wird *StoredProcedure* festgelegt:

```
sc.CommandType = CommandType.StoredProcedure
```

Diesem Kommando wird nun ein Parameter zugewiesen:

```
sp = new SqlParameter()
```

Als Name wird der in der Prozedurdefinition verwendete angegeben:

```
sp.ParameterName = "@ArtikelNummer"
```

Datentyp

Nun ist der Datentyp in der Prozedur entsprechend dem SQL-Datentyp festzulegen:

```
sp.DbType = DbType.Int32
```

Richtung

Zuletzt ist noch die Richtung anzugeben, es handelt sich offensichtlich um einen Eingabeparameter:

```
sp.Direction = ParameterDirection.Input
```

Dieser Parameter wird nun die `Parameter`-Kollektion des Kommandos angehängt. Auf demselben Wege lassen sich so viele Kommandos definieren, wie die gespeicherte Prozedur erfordert.

```
sc.Parameters.Add(sp)
```

Nun wird das fertige Kommando der passenden Eigenschaft des Datenadapters zugewiesen. Da es sich um eine Abfrage handelt, wird es hier der Eigenschaft `SelectCommand` übergeben:

```
da.SelectCommand = sc
```

Der fertige Datenadapter wird nun als Sitzungsvariable abgelegt, sodass die spätere Verwendung schneller abläuft:

Datenadapter

```
Session("Adapter") = da
```

Damit ist der vorbereitende Teil abgeschlossen. Das `DropDownList`-Steuerelement wird angezeigt und wartet auf eine Änderung der Auswahl durch den Benutzer. Durch das `AutoPostBack`-Attribut wird das Formular sofort gesendet.

Beim Aufruf der Seite über das Formular wird der zweite Teil der `Page_Load`-Methode verarbeitet. Hier wird zuerst der Datenadapter aus der Sitzungsvariablen restauriert:

Page_Load

```
da = CType(Session("Adapter"), SqlDataAdapter)
```

Jetzt wird für die Anzeige der Daten ein neues `DataSet`-Objekt erstellt:

```
ds = New DataSet()
```

Bisher fehlte noch der eigentliche Wert, der dem Parameter übergeben wird. Ohne diesen Wert kann die eigentliche Abfrage nicht ausgeführt werden. Sie können den Parameterwert direkt ansprechen, indem auf die `SelectCommand`-Eigenschaft zugegriffen wird. Die Nummer wird der Auswahlliste entnommen. Parameter akzeptieren als Datentyp `Object`, sodass Sie hier den Wert als Zeichenkette übernehmen können. Es wird dann intern versucht, diesen Wert in den benötigten Datentyp zu übernehmen. In der Praxis ist es sicher sinnvoll, hier eine Typprüfung auf anderem Wege vorzunehmen oder eine Ausnahmebehandlung mit `Try-Catch` zu implementieren.

```
da.SelectCommand.Parameters("@ArtikelNummer").Value  
    = ArtikelNummer.SelectedItem.Value
```

Das fertige Kommando im Datenadapter wird nun verwendet, um die Abfrage auszuführen und die Daten in das `DataSet`-Objekt zu überführen:

```
da.Fill(ds)
```

Die so gewonnene Datenquelle wird dann zur Anzeige des `DataGrid`-Steuerelements verwendet:

```
Bestellungen.DataSource = ds
```

Die Ereignisse des Datenadapters

Die bisherige Programmierung basierte immer auf einem sehr direkten Weg, das heißt, die benötigten Aktionen wurden direkt durch Aufruf der entsprechenden Methode ausgelöst. Sie können aber auch auf bestimmte Ereignisse reagieren:

► RowUpdating

Dieses Ereignis wird ausgelöst, wenn der Aktualisierungsprozess beginnt.

► RowUpdated

Mit Hilfe dieses Ereignisses können Sie eigene Prozesse an das Ende des Aktualisierungsprozesses anhängen.

Beide Ereignisse verhalten sich damit ähnlich wie ein `ON UPDATE`-Trigger in T-SQL. Die Ereignisargumente der Ereignisbehandlungsmethoden lauten `SqlRowUpdatingEventArgs` bzw. `OleDbRowUpdatingEventArgs` und `SqlRowUpdatedEventArgs` bzw. `OleDbRowUpdatedEventArgs`. Der Parameter kennt folgende Eigenschaften:

Eigenschaft	Beschreibung
Command	Command-Objekt für die Aktualisierung
Errors	Fehler, die erzeugt wurden
Row	Die Reihe, die aktualisiert wird
StatementType	Typ des Kommandos
Status	Status; eine Aufzählung des Typs <code>UpdateStatus</code>
TableMapping	<code>DataTableMapping</code> -Objekt das verwendet wurde

Tabelle 9.7: Eigenschaften des Ereignisarguments der Ereignisbehandlungsmethode für `RowUpdating`- bzw. `RowUpdated`-Ereignisse

Ein Wort zu typisierten DataSets

Wenn Sie Datenstrukturen in einem *DataSet* neu aufbauen, können Sie dies mit einer strengen Typisierung machen. Damit nicht jeder Datentyp einzeln in den entsprechenden Eigenschaften programmiert werden muss, genügt die Angabe einer XSD-Datei (XML-Schema) mit den Strukturinformationen.

Der von .NET verarbeitete XSD-Dialekt umfasst zwar den Sprachumfang, der vom W3C standardisiert wurde, ist jedoch zusätzlich noch erweitert worden, um den Bedingungen gerecht zu werden, die eine Koexistenz mit dem SQL Server 2000 benötigt.

Letztlich genügt das Lesen eines Schemas aus einer XSD-Datei, um eine leere, aber vollständig beschriebene Struktur in einem *DataSet*-Objekt zur Verfügung zu haben. Trotz der Unterstützung in Visual Studio .NET sind allerdings gediegene XSD-Kenntnisse notwendig. Für einen ersten Eindruck über XSD informieren Sie sich bitte in Abschnitt »Eine sehr kurze Einführung in XSD« ab Seite 450.

9.4 Umgang mit Datenbanken in Visual Studio .NET

Visual Studio .NET unterstützt bereits in der Professional-Version den Programmierer beim Umgang mit Datenbanken. Dabei geht es nicht nur um die Assistenten, die auch unerfahrenen Entwicklern einen Zugang zu Datenbankabfragen erlauben, sondern um ganz praktische Hilfsmittel, die den Entwicklungszyklus verkürzen.

9.4.1 Zugriff auf einen Datenbankserver

Der einfachste Start zur Verwaltung einer Datenbankverbindung zum Entwurfszeitpunkt führt über den SERVER-EXPLORER. Dieses Werkzeug starten Sie in Visual Studio .NET über die Tastenkombination **[Strg]-[Alt]-[S]** oder über das Menü ANSICHT, Eintrag SERVER-EXPLORER. Sie finden dort eine Liste mit zwei Einträgen: DATENBANKVERBINDUNGEN und SERVER. Im Kontextmenü des Zweiges DATENBANKVERBINDUNGEN wählen Sie nun VERBINDUNG HINZUFÜGEN. Im folgenden Dialog tragen Sie den Namen des SQL Servers ein wählen die Datenbank aus.



Visual Studio .NET verwendet intern einen OleDb-Provider zum Zugriff. Möglicherweise verhält sich dieser Treiber bei komplexen Abfragen anders als der native SQL-Provider, den Sie in ADO.NET verwenden.



Abbildung 9.27: Hinzufügen einer Datenbankverbindung

Wenn die Datenbank bereits Tabellen oder andere Datenbankobjekte enthält, finden Sie diese im Zweig der neuen Verbindung.



Abbildung 9.28: Der Server-Explorer mit einer Datenbankverbindung

Zeitgleich mit dem Aufruf des Explorers erscheint eine neue Symbolleiste, die das Ausführen von SQL und die Generierung von Abfragen erlaubt. Wenn Sie diese Leiste nicht sehen, wählen Sie im Kontextmenü der Symbolleisten den Eintrag DATENBANKENTWURF.

9.4.2 Daten in Visual Studio .NET manipulieren

Wenn Sie Daten direkt in Visual Studio .NET manipulieren möchten, genügt die Auswahl der benötigten Tabelle und des Eintrags DATEN VON TABELLE ABRUFEN.



Abbildung 9.29: Ansicht des Server-Explorers mit einem Datenbankserver

Die Tabelle wird jetzt angezeigt. Wenn Sie Daten im Gitter ändern, wird dies sofort an die Datenbank übertragen. Neue Reihen fügen Sie hinzu, indem Sie diese in die Reihe mit dem Stern am Anfang schreiben.

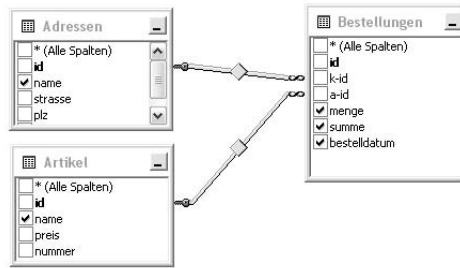
9.4.3 Hilfsmittel zum Generieren komplexer Abfragen

Es ist für diese Übung empfehlenswert, alle Ansichtsoptionen für den Datenbankentwurf einzuschalten. Dazu wählen Sie entweder die Tastenkombinationen `[Strg-1]`, `[Strg-2]`, `[Strg-3]` und `[Strg-4]` oder die entsprechenden Einträge im Menü ANSICHT | BEREICHE.

Eine SELECT-Abfrage generieren

Um eine SELECT-Abfrage zu generieren, gehen Sie folgendermaßen vor:

1. Ziehen Sie die benötigten Tabellen in den Bereich DIAGRAMM.
2. Klicken Sie die Spalten an, die Sie in der Ausgabe erhalten möchten.



3. Tragen Sie die Aliase im Bereich DATENBLATT in der Spalte ALIAS ein.

Spalte	Alias	Tabelle	Ausgabe	Sortierungsart	Sortierreihenfolge	Kriterien	Oder...
menge	Menge	Bestellungen	✓				
summe	Summe	Bestellungen	✓				
bestelldatum	[Bestellt am]	Bestellungen	✓				
name	Kundenname	Adressen	✓				
name	Artikelbezeichnung	Artikel	✓				

4. Der SELECT-Befehl entsteht nun auf Basis der Relationen im Bereich SQL.

```
SELECT Bestellungen.menge AS Menge, Bestellungen.summe AS Summe, Bestellungen.bestelldatum AS [Bestellt am], Adressen.name AS Kundenname,
       Artikel.name AS Artikelbezeichnung
FROM   Artikel INNER JOIN
       Bestellungen ON Artikel.id = Bestellungen.[a-id] INNER JOIN
       Adressen ON Bestellungen.[k-id] = Adressen.id
```

5. Wenn Sie in der Menüleiste nun auf Ausführen klicken, erscheint das Ergebnis im Bereich ERGEBNISSE.

	Menge	Summe	Bestellt am	Kundenname	Artikelbezeichnung
▶	1	99	9.7.2002 00:56:26	Lukas Werlich	Big Color Keyboard
	1	2850	4.8.2002 00:56:26	Jörg Krause	MSDN Abo Professional
	1	99	1.5.2002 00:56:26	Jörg Krause	Big Color Keyboard
	1	199,99	9.8.2002 00:56:26	Jörg Krause	Windows XP Pro
	3	450	8.8.2002 01:33:01	Jörg Krause	Windows XP Pro
	2	490	7.8.2002 01:33:32	Carolin Scholz	Windows XP Home
	1	99	6.8.2002 01:33:38	Carolin Scholz	Windows XP Pro
	1	2850	10.7.2002 01:33:5	Carolin Scholz	MSDN Abo Professional
	1	2850	15.7.2002 01:34:3	Carolin Scholz	MSDN Abo Professional
	1	2850	25.7.2002 01:34:3	Carolin Scholz	MSDN Abo Professional
	1	19,99	9.8.2002 15:50:39	Carolin Scholz	Sonderanfertigung
*					

Andere Abfragetypen generieren

Wenn Sie Abfragen für die Befehle UPDATE, DELETE oder INSERT erzeugen möchten, können Sie nur mit einer Tabelle arbeiten. Löschen Sie dazu die nicht benötigten Tabellen mit der **[Entf]**-Taste aus der Anzeige.

Klicken Sie nun in den Bereich SQL mit der rechten Maustaste. Im Kontextmenü wählen Sie TYP ÄNDERN und dort WERTE EINFÜGEN. Die Änderung des Typs ist auch über die Symbolleiste möglich. Aktivieren Sie in der Tabellenansicht die Spalten, die hinzugefügt werden sollen. Tragen Sie im Bereich DATENBLATT die einzufügenden Werte ein.

INSERT INTO

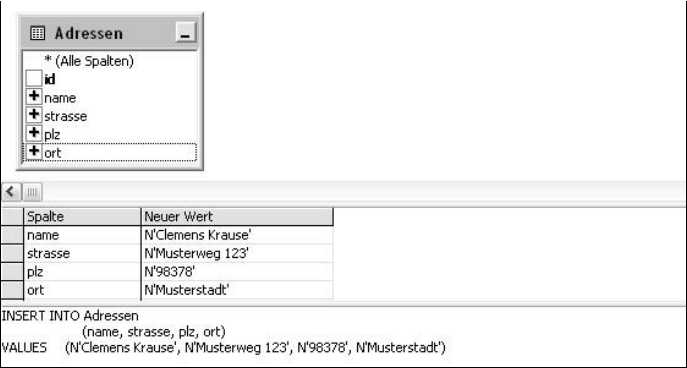


Abbildung 9.30: Erzeugen eines INSERT-Befehls

Zum Löschen von Daten wählen Sie im Kontextmenü TYP ÄNDERN und dort LÖSCHEN. Wählen Sie im Bereich DATENBLATT die Spalte aus, die als Löschkriterium herangezogen werden soll. In der Spalte Kriterien können Sie nun die WHERE-Bedingungen angeben. Innerhalb der Reihen werden mehrere Bedingungen OR-verknüpft, zwischen den Reihen mit AND.

DELETE FROM

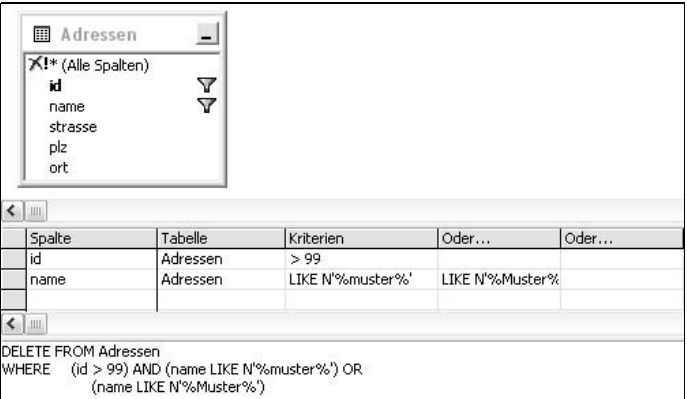


Abbildung 9.31: Erzeugen eines DELETE-Befehls

UPDATE Zum Erzeugen eines `UPDATE`-Befehls wählen Sie die Option `AKTUALISIEREN`. Auch hier kann ähnlich wie bei `DELETE` eine Reihe von Kriterien definiert werden, die in der `WHERE`-Bedingung genutzt werden.

Die noch nicht beschriebene Option `ERGEBNISSE EINFÜGEN` erzeugt eine Kombination aus `SELECT` und `INSERT`, wobei die Ergebnisse der `SELECT`-Abfrage in eine andere Tabelle mittels `INSERT` eingefügt werden.

9.5 Die Datenbindung an Steuerelemente

Fast immer ist die Nutzung einer Datenbank mit der Ausgabe der Daten verbunden. Kommt als Ziel eine Datei in Betracht, ist die Zahl der Möglichkeiten überschaubar. Bei der Programmierung von Webseiten geht es jedoch fast immer um den Benutzer und die Aufbereitung der Ausgabe für Menschen. So vielfältig wie die Bedürfnisse sind auch die Angebote an Steuerelementen in ASP.NET, die speziell der Ausgabe von Datensätzen dienen. Das schließt natürlich auch Methoden zum Bearbeiten ein. Dieser Abschnitt zeigt die bereits zuvor an einigen Stellen eingesetzten Steuerelemente detailliert.

Behandelt werden allgemein wiederholende Steuerelemente und die speziell für Datenbanken entworfenen Daten-Steuerelemente. Darüber hinaus wird die für den Abruf bestimmter Daten verwendete Datenbindungssyntax im Abschnitt 9.5.1, »Die Bindung von Datenquellen« ab Seite 927 betrachtet.

Table Auch wenn die Ausgabe von Daten in Tabellen mit dem Steuerelement `Table` möglich ist, wird dieses hier dennoch nicht betrachtet, denn es lässt nicht die direkte Bindung einer Datenquelle zu. Dies auch als Hinweis dazu, was hier unter »Daten-Steuerelement« verstanden wird.

Die gestalterischen Seiten der Daten-Steuerelemente selbst wurden bereits im Abschnitt 6.6, »Vorlagengebundene Daten-Steuerelemente« ab Seite 566 behandelt. In diesem Abschnitt geht es um die Art der Datenbindung aus Sicht einer Datenquelle.

9.5.1 Die Bindung von Datenquellen

Die mit ASP.NET eingeführte Datenbindungssyntax ist äußerst flexibel und vielseitig. Erfahrungsgemäß gelingt Einsteigern zwar sehr schnell eine einfache Bindung, in ihrer Vielfalt und Funktionalität stecken jedoch einige Tücken.

Grundsätzlich unterscheidet sich die Art der Auswertung der Bindungsausdrücke von der des eingebetteten Codes. Betrachten Sie folgenden einfachen Ausdruck:

```
Name: <%# Addison.VBNet.Data.Binding.Name %>
```

Hier wird im Namensraum `Addison.VBNet.Data`, der in diesem Kapitel verwendet wird, eine Klasse *Binding* und dort die Eigenschaft *Name* aufgerufen. Vergleichen Sie dazu folgenden Aufruf:

```
<% @Imports Namespace="Addison.VBNet.Data" %>
```

```
<% = Binding.Name %>
```

Dies ist die verkürzte Schreibweise für `Response.Redirect` mit eingebettetem Code.

Beide Verfahren führen letztlich zu demselben Ergebnis, verhalten sich aber verschieden. Der wichtigste Unterschied: Die Auswertung von eingebettetem Code erfolgt immer während der Verarbeitung der Seite. Sie können weder den Zeitpunkt ändern noch die Ausgabe an einer Stelle von einem anderen Programmteil aus unterdrücken. Es ist deshalb grundsätzlich nicht empfehlenswert, eingebettete Anweisungen in der Seite zu verwenden. Das Zusammenspiel mit hinterlegtem Code funktioniert kaum sinnvoll.

Ganz anders sieht es mit der Datenbindungssyntax aus. Hier erfolgt die Verarbeitung erst, wenn entweder für das betreffende Steuerelement oder für die gesamte Seite die Methode `DataBind` aufgerufen wird. Erst dann wird die Datenquelle abgefragt und der Inhalt des Steuerelements oder aller Elemente der Seite aktualisiert. Da die Methode allen Steuerelementen zur Verfügung steht, gibt es eine Ereigniskette, die die Bindung weiterreicht. Wenn Sie mit verschachtelten Steuerelementen arbeiten, führt der Aufruf von `DataBind` in einem der Elemente dazu, dass alle *untergeordneten* ebenfalls an ihre jeweilige Datenquelle gebunden werden.

Anforderungen an die Datenquelle

Prinzipiell muss eine Datenquelle lediglich einen Wert liefern. Ob die Erzeugung des Wertes durch eine selbst programmierte Eigenschaft, eine Methode mit Rückgabewert oder eine Eigenschaft oder Methode einer eingebauten Klasse geschieht, ist dabei ohne Belang.

Lediglich bei den Steuerelementen, die wiederholende Werte anzeigen können, muss die Quelle bestimmte Bedingungen erfüllen. Werden mehrere Werte übergeben und sollen diese durch die wiederholte Ausgabe des Steuerelements oder eines Teils davon erscheinen, muss es eine Technik geben, mit der Listen verarbeitet werden können. Das heißt, die Quelle muss auf Anforderung ein Element freigeben, dann zum nächsten Element springen und anzeigen, dass das Ende der Auflistung erreicht wurde. Damit dies funktioniert, muss die Quelle eine der Schnittstellen `IEnumerable`, `ICollection` oder `IDataSource` implementieren. Das ist bei vielen Klassen im Framework der Fall, beispielsweise lassen sich deshalb solche Typen wie `DataGridView`, `DataReader` oder auch `ArrayList` und `Hashtable` binden.

Wenn Sie eigene Datenquellen entwerfen, ist die Implementierung einer der Schnittstellen erforderlich.

Schnittstellen für den Entwurf von Datenquellen

IEnumerator Die einfachste Schnittstelle ist `IEnumerator`. Sie verfügt lediglich über eine einzige Methode: `GetEnumerator`. Wollen Sie eine eigene Klasse damit ausstatten, müssen Sie diese Methode implementieren. Sie muss ein Objekt vom Typ `IEnumerator` zurückgeben. Die spannende Arbeit passiert also in einer weiteren Implementierung. Die Schnittstelle `IEnumerator` stellt eine Eigenschaft und zwei Methoden zur Verfügung:

► **Current**

Diese Eigenschaft gibt das aktuelle Element aus der Auflistung zurück.

► **MoveNext**

Mit dieser Methode wird der Wechsel zum nächsten Element veranlasst. Wenn kein weiteres Element mehr verfügbar ist, wird `False` zurückgegeben.

 Reset

Diese Methode setzt den Zeiger auf die Elemente wieder an den Anfang zurück.

Es ist nicht die Aufgabe der Schnittstellendefinition festzulegen, wie die Elemente in die Auflistung hineingelangen. Sie können hier entweder einen Abruf aus einer anderen Quelle programmieren oder die klassische Methode `Add` verwenden. Wichtig ist, dass der Zeiger ungültig wird, wenn sich die Aufzählung ändert. Die Implementierung sollte deshalb immer so funktionieren, dass erst alle benötigten Werte gesammelt werden und dann das Durchlaufen der Aufzählung passiert. Das ist bei der Datenbindung idealerweise der Fall, weil Sie in den Übergabeprozess selbst nicht mehr eingreifen können.

Implementierung

Etwas anders funktioniert `ICollection`. Hier gibt es eine Eigenschaft `Count`, die über die Anzahl der Elemente informiert. Mit der Methode `CopyTo` lassen sich die Elemente ganz oder teilweise in ein Array kopieren. Aufzählungen dieser Art sind in .NET nicht threadsicher, das heißt, in einer Umgebung in der mehrere Threads auf die Applikation zugreifen, überschreiben sich Änderungen an der Aufzählung gegenseitig. Damit das nicht unbemerkt bleibt, wird eine Ausnahme ausgelöst, wenn das passiert. Damit die Aufzählung nun sicher arbeitet, kann der Zugriff für anderen Threads gesperrt werden. Ist die Sperre aktiv, sollte die Eigenschaft `IsSynchronized` implementiert sein und diesen Zustand anzeigen. Die Eigenschaft `SyncRoot` gibt dagegen ein Objekt zurück, das für die Synchronisierung verwendet wurde.

ICollection

Der Umgang mit `ICollection` funktioniert wiederum anders. Hier wird eine Methode `GetList` erwartet, die ein Objekt vom Typ `IList` zurückgibt. Es geht also darum, auch diese Schnittstelle zu implementieren. Außerdem gibt die Eigenschaft `ContainsListCollection` an, ob eine Auflistung momentan enthalten ist.

ICollection

Die Implementierung von `IList` verlangt drei Eigenschaften und sieben Methoden. Es ist die im Vergleich zu den anderen »ausgereifteste« Schnittstelle, die allerdings auch den geringsten Spielraum für eigene Erfindungen gestattet. Informationen über die Auflistung enthalten die folgenden Eigenschaften:

- ▶ `IsFixedSize`
Zeigt an, dass die Größe unveränderlich ist.
- ▶ `IsReadOnly`
Zeit an, dass der Inhalt schreibgeschützt ist.
- ▶ `Item`
Gibt ein Element zurück. In VB.NET ist dies auch die Standardmethode, weshalb der Zugriff auch ohne den Methodenamen mit der Arrayschreibweise erfolgen kann.

Die nachstehenden Methoden dienen dem Füllen mit Elementen und der Entnahme:

- ▶ `Add (object Wert)`
Fügt der Auflistung ein Element hinzu.
- ▶ `Clear`
Entfernt alle Elemente der Auflistung.
- ▶ `Contains (object Wert)`
Prüft, ob das Objekt in der Auflistung enthalten ist und gibt im Erfolgsfall `True` zurück.
- ▶ `IndexOf (object Wert)`
Gibt den numerischen Index des Objekts zurück.
- ▶ `Insert (int Index, object Wert)`
Fügt ein Objekt an der durch `Index` bestimmten Stelle ein.
- ▶ `Remove (object Wert)`
Entfernt das Objekt aus der Auflistung.
- ▶ `RemoveAt (int Index)`
Entfernt ein Objekt an der durch `Index` bestimmten Stelle.

Praktische Implementierung einer Schnittstelle

Im folgenden Beispiel wird die Schnittstelle `IEnumerable` implementiert. Dies dient als Demonstration für die Datenbindung. Umgesetzt wird eine Namensliste. Intern basiert diese auf `ArrayList`. Es ist bei größeren Projekten angebracht und stilistisch sauberer, wenn eine vorhandene Liste neu implementiert wird, wenn dadurch der Typ strenger gehandhabt werden kann. Im Beispiel kann die Liste nur Zeichenketten aufnehmen, während `ArrayList` jeden Typ annimmt.

Am Anfang steht wie üblich die Deklaration der Namensräume. Nun wird der »innere Teil«, von `IEnumerator` abgeleitet, implementiert. Praktisch basiert das auf zwei Feldern: `_al` ist der Speicherort der `ArrayList` und `_idx` ist eine Hilfsvariable zum Speichern des Zeigers beim Abruf der Kollektion.

Implementierung einer eigenen Datenquelle (DataSourceImplementation.aspx.vb)

Das Attribut `<Serializable()>` ist notwendig, weil die Instanz der Klasse später im Anzeigestatus (*ViewState*) gespeichert werden soll und dazu serialisiert werden muss. Um den Vorgang selbst müssen Sie sich nicht kümmern.

<Serializable()>

Implementiert werden dann die zwei Methoden `MoveNext` und `Reset`. Außerdem wird eine Eigenschaft `Current` realisiert, die den aktuellen Wert zurückgibt.

```
<Serializable()> _
Public Class NameListEnumerator
    Implements IEnumerator

    Public _al As ArrayList = Nothing
    Public _idx As Integer = 0

    Public ReadOnly Property Current() As Object _
        Implements IEnumerator.Current
        Get
            Return _al(_idx - 1)
        End Get
    End Property

    Public Function MoveNext() As Boolean _
        Implements IEnumerator.MoveNext
        If _al Is Nothing Then
            Return False
        End If
        If _idx < _al.Count Then
            _idx = _idx + 1
            Return True
        Else
            Return False
        End If
    End Function

    Public Sub Reset() _
        Implements IEnumerator.Reset
        _idx = 0
    End Sub
End Class
```

**IEnumerable
implementieren**

Nun wird die eigentliche Liste implementiert, die die Schnittstelle `IEnumerable` nutzt. Hier werden die fehlenden Methoden zur Benutzung der Liste programmiert. `Add` fügt Werte hinzu, `Clear` löscht alle Werte.

```
<Serializable(> _
Public Class NameList
    Implements IEnumerable
    Dim _nlg As NameListEnumerator

    Public Sub Clear()
        _nlg._idx = 0
        _nlg._al = Nothing
    End Sub
    Public Sub Add(ByVal val As String)
        If _nlg._al Is Nothing Then
            _nlg._al = New ArrayList()
        End If
        _nlg._al.Add(val)
    End Sub
```

Nun wird das wichtigste hier – `GetEnumerator` – implementiert; die Methode die letztlich für den Zugriff auf den Abrufmechanismus verantwortlich ist. Sie gibt eine Instanz der zuvor erstellten Klasse *NameListEnumerator* zurück. Beachten Sie, dass der Aufruf von `Reset` hier erfolgen muss; dies erledigt die Datenbindung später nicht.

```
    Public Sub New()
        _nlg = New NameListEnumerator()
        _nlg._al = New ArrayList()
    End Sub

    Public Function GetEnumerator() As IEnumerator _
    Implements IEnumerable.GetEnumerator
        _nlg.Reset()
        Return _nlg
    End Function
End Class
```

**Ereignisbehand-
lungsmethode**

Jetzt folgt die Erstellung der eigentlichen Hauptklasse. Hier werden die Ereignisbehandlungsmethoden für die beiden Schaltflächen zum Hinzufügen und Löschen geschrieben. Die fertige Liste wird außerdem im `ViewState`-Objekt gespeichert.

```
Public Class DataSourceImplement
    Inherits System.Web.UI.Page
    Protected Name As TextBox
```



```

Protected NamensListe As DataList
Protected Counter As Label
Protected WithEvents Send As Button
Protected WithEvents Clear As Button

Public n1 As NameList

Public Sub Eintragen_Click(ByVal sender As Object, ↵
                        ByVal e As EventArgs)
    n1 = CType(ViewState("n1"), NameList)
    If String.Compare(Name.Text, String.Empty) <> 0 Then
        n1.Add(Name.Text)
        NamensListe.DataBind()
    End If
    ViewState("n1") = n1
End Sub

Public Sub Loeschen_Click(ByVal sender As Object, ↵
                        ByVal e As EventArgs)
    n1.Clear()
    ViewState("n1") = n1
    NamensListe.DataBind()
End Sub

Private Sub OnBinding(ByVal sender As Object, ↵
                    ByVal e As EventArgs)
    Counter.Text = ((CType(sender, ↵
                        DataList)).Items.Count + 1).ToString()
End Sub

Private Sub Page_Load(ByVal sender As Object, ↵
                    ByVal e As System.EventArgs) _
Handles MyBase.Load
    If Not Page.IsPostBack Then
        n1 = New NameList()
        ViewState("n1") = n1
    Else

```

Am Ende folgt die Bindung der Datenquelle in der üblichen Form. Die Art der Implementierung war so gewählt, dass das Steuerelement mit den Informationen umgehen kann.

Die Bindung an die Datenquelle

```

        n1 = CType(ViewState("n1"), NameList)
        NamensListe.DataSource = n1
        AddHandler NamensListe.DataBinding, AddressOf OnBinding
    End If
End Sub
End Class

```

Mit dem Debugger untersuchen

Wenn Sie den Debugger starten und das Programm schrittweise durchgehen, können Sie feststellen, wie nach dem Aufruf der Methode `DataBind` als Nächstes die Methode `MoveNext` in `NameListEnumerator` aufgerufen wird, danach `Current` usw. – für jedes enthaltene Element. Die Datenbindung »bedient« sich quasi an Ihren Methoden. Auf diesem Wege können Sie leicht komplexe Abfragen realisieren und trotzdem die vorgefertigten Steuerelemente zur Darstellung nutzen.

Als Letztes soll ein Blick auf den HTML-Text zeigen, wie die Datenquelle genutzt wird:

```
<body>
<h1>Eigene Datenquelle implementieren</h1>
<h2>IEnumerator</h2>
<form id="DataSourceImplement" method="post" ⚡
  runat="server">
  <asp:TextBox Runat="server" ID="Name"/>
  <asp:Button Runat="server" ID="Send" Text="Eintragen" ⚡
    OnClick="Eintragen_Click"/>
  <asp:Button Runat="server" ID="Clear" Text="Löschen" ⚡
    OnClick="Loeschen_Click"/>
</form>
<asp:DataList Runat="server" ID="NamensListe">
  <ItemTemplate>
    <%# Container.DataItem %>
  </ItemTemplate>
</asp:DataList>
</body>
```

Listing 9.37: Die Nutzung der selbst erstellten Datenquelle (DataSourceImplement.aspx)

Hier wird ein `DataList`-Steuerelement eingesetzt. In der Vorlage wird lediglich `Container.DataItem` geschrieben. Die Zuordnung der Elemente erfolgt über die `IEnumerable`-Schnittstelle.

IEnumerator

4

Wert

Test

Datenquelle

123456

Abbildung 9.32: Die Ausgabe des Programms nach einigen Versuchen

9.5.2 Hintergrund der Datenbindungssyntax

Bei der Nutzung der Datenbindung im Zusammenhang mit eigenem Code können Sie mehrere Formen der Datenbindungssyntax verwenden.

Das vorhergehende Beispiel nutzte den direktesten Weg:

```
Container.DataItem
```

Das funktioniert, weil die Datenquelle nur einen einzigen Wert pro Abruf zurückgibt – die Eigenschaft `Current` hat das erledigt. Wenn nun eine Klasse verwendet wird, die mehrere Werte pro Datensatz enthält, muss eine Auswahl getroffen werden. Verzeichnisse wie `Hashtable` implementieren Schlüssel/Werte-Paare. Die Eigenschaften, die den Zugriff erlauben, heißen `Key` und `Value`. Zur Bindung der Werte schreiben Sie dann:

```
Container.DataItem.Key
```

```
Container.DataItem.Value
```

All diese Bindungen passieren am Ende des Seitenaufbaus, aber noch vor dem Ende von `Page_Load`. Meist ist dies auch erwünscht und ausreichend.

**Bindung von
Listen**

Formatierung und Konvertierung von Datenquellen

Im Buch wurde schon an einigen Stellen die Methode `Eval` verwendet, wenn es um den Zugriff auf Daten ging. An dieser Stelle sollen einige Hintergrundinformationen gegeben werden, um den Einsatzfall besser abschätzen zu können.

Generell handelt es sich um eine so genannte späte Bindung. Das heißt, der Vorgang der Datenbindung wird vollständig abgeschlossen, die Seite wird aufgebaut und dann erst – unmittelbar vor der Auslieferung – werden die `Eval` übergebenen Datenwerte analysiert und bearbeitet. ASP.NET nutzt intern die Reflektion genannte Technik, mit der zur Laufzeit Informationen über Klassen und Methoden ermittelt werden. Das ist zwar sehr leistungsfähig, greift aber praktisch in eine bereits vorkompilierte Seite ein. Der Nachteil dabei: Die Methode `Eval` ist langsamer. Eingesetzt wird sie vor allem, um Daten vielfältig für die Ausgabe zu formatieren.

Späte Bindung

Die Methode ist statisch und wird in `DataBinder` definiert:

```
<%# DataBinder.Eval(Container.DataItem, "Field", "Format") %>
```

Auf die Formatierung und die weiteren Teile der Syntax wurde bereits in Abschnitt »Eine Einführung in die Datenbindung in Vorlagen« ab Seite 571 eingegangen.

9.5.3 Ereignisbehandlung

Sie können den Zeitpunkt der Bindung der Daten nutzen, um davon abhängende Aktionen zu starten. Dazu enthalten alle daten-gebundenen Steuerelemente ein Ereignis `DataBinding`. Der folgende Code, der ergänzend im letzten Beispiel platziert wurde, zeigt die Vorgehensweise.

Zuerst wird dem Ereignis eine Ereignisbehandlungsmethode (der Delegat) zugewiesen:

```
AddHandler NamensListe.DataBinding, AddressOf OnBinding
```

Die Implementierung der Methode `OnBinding` (der Name ist willkürlich) ist sehr einfach:

```
Private Sub OnBinding(ByVal sender As Object,   
                      ByVal e As EventArgs)   
    Counter.Text = ((CType(sender,   
                          DataList)).Items.Count + 1).ToString()   
End Sub
```

Wie es funktioniert

Hier wird dem `Label`-Steuerelement `Counter` die Anzahl der Datensätze zugewiesen, die derzeit in dem Daten-Steuerelement sind, das das Datenbindungsereignis auslöste. Im vorhergehenden Beispiel fehlte eine explizite Methode »Count« zur Ermittlung der Anzahl der Elemente. Die hier gezeigte Vorgehensweise kompensiert diesen Mangel. Man muss aber eingestehen, dass es nicht zwingend erforderlich ist, dazu Ereignisse zu verwenden. Die Verwendung von Ereignissen ist oft kompakter und eleganter, aber nur selten besser lesbar.

9.6 Indexdienst programmieren

Der Indexdienst, früher bekannt als Index Server, gehört zum Lieferumfang von Windows 2000/XP und .NET-Server. Zusammen mit ASP.NET lässt sich eine sehr leistungsfähige Suchmaschine für die Homepage bauen.

9.6.1 Motivation und Hintergründe

Eine Volltextsuche gehört für größere Websites zur Standardausstattung. Läuft auf dem Webserver ASP.NET und der Indexdienst, lässt sich ein Suchformular besonders einfach realisieren. Neben DHML werden auf Wunsch auch Word- und Excel-Dateien, PDF und vieles mehr durchsucht. Die Tücken liegen allerdings im Detail, denn Suchanfragen können den Server stark belasten. Der Abschnitt zeigt den ressourcensparenden Einsatz der Klasse `DataSet` für die Speicherung der Suchergebnisse. Außerdem wird die seitenweise Ausgabe dem Daten-Steuerelement `DataGrid` überlassen.

Volltextsuche

Der Indexdienst

Der Indexdienst gehört für viele Webentwickler offensichtlich zu den großen Unbekannten, denn nur selten kommt er auf Servern zusammen mit programmierten Abfragen zum Einsatz. Das mag auch daran liegen, dass Microsoft vor langer Zeit eine eigene Abfragesprache propagiert hat, die man zu Recht als kryptisch bezeichnen kann. Die in allen Windows-Versionen mit NT-Kern befindliche Version kann jedoch auch über SQL-Anweisungen abgefragt werden. Es sind zusätzlich proprietäre Erweiterungen hinzugefügt worden, die auf die Bedürfnisse der Katalogabfrage zugeschnitten sind. Außerdem beschränken sich die Möglichkeiten auf den Einsatz von `SELECT`. Die erste wichtige Kenntnis ist also die des SQL-Dialekts des Indexdienstes.

SQL nutzen

Die zweite wichtige Kenntnis ist der Name des OleDb-Providers für den Indexdienst: »MSIDX« (Microsoft Indexing Service). Da ADO.NET neben dem SQL Server standardmäßig Klassen zum Zugriff auf OleDb-Quellen bietet, kann die Verwendung in einer ASP.NET-Anwendung kein großes Problem darstellen.

OleDb-Provider

Der Katalog Als Nächstes gilt es nun einen Katalog vorzubereiten. Wenn Sie den Indexdienst noch nicht verwenden, gehen Sie dazu folgendermaßen vor:

1. Öffnen Sie in der Systemsteuerung das Programm COMPUTER-VERWALTUNG.
2. Wählen Sie in der MMC den Zweig DIENSTE UND ANWENDUNGEN und darunter INDEXDIENST.
3. Falls noch keine Kataloge eingerichtet sind, legen Sie einen an. Im Beispiel heißt der Katalog »Web« und dient der Indizierung der virtuellen Verzeichnisse des Webservers.
4. Starten Sie den Indexdienst durch Anklicken des schwarzen Pfeils in der Symbolleiste oder über die Option STARTEN des Kontextmenüs des Eintrags INDEXDIENST.
5. Der Indexdienst beginnt die Indizierung. Abfragen können parallel dazu bereits erstellt werden, allerdings richtet sich das Suchergebnis nach der Zahl der bereits indizierten Dateien.

Diverse Feineinstellungen der Kataloge können im Katalogmanager erfolgen.

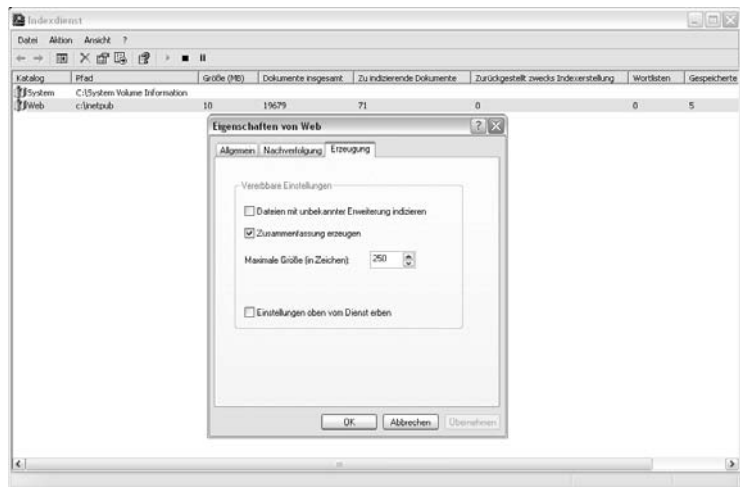


Abbildung 9.33: Der Katalogmanager des Indexdienstes

Details des Indexdienstes

Die Syntax der SQL-Abfrage basiert auf folgendem Muster (|-Zeichen geben Alternativen an):

```
SELECT Columns|Viewname|*
      From_Clause
      [WHERE_Clause]
      [Order_Clause]
```

Für From_Clause kann Folgendes eingesetzt werden:

```
[Servername.Catalog] ↵
  SCOPE([SHALLOW | DEEP TRAVERSAL OF ↵
        ("Path" [, "Path"]]))
```

Die Option hinter SCOPE kann entfallen. Wenn sie nicht gesetzt ist, werden alle Verzeichnisse im Katalog durchsucht, beginnend mit dem Stammverzeichnis. Wird die Option SHALLOW gesetzt, werden nur die angegebenen Pfade durchsucht; mit DEEP werden auch untergeordnete einbezogen. Für WHERE_Clause können so genannte Prädikate zusammen mit Booleschen Operatoren (AND, OR, NOT) eingesetzt werden:

```
WHERE Prädikat AND|OR|AND NOT Prädikat ....
```

Alternativ kann mit Vergleichsoperatoren und Konstanten gearbeitet werden:

```
WHERE Spalten_Name <|>|<=|>=|!= 'Konstante'
```

Die Prädikate bestimmen die Art der Suche. Durchsucht wird die angegebene Spalte. Suchwörter können mit AND, OR oder AND NOT kombiniert werden:

```
CONTAINS (Spalten_Name, 'Suchwort'|'"Wort"
          AND "Wort"
          OR "Wort3"
          AND NOT "Wort4"')
```

Die Spalte wird mit Hilfe einer natürlichsprachigen Frage durchsucht:

```
FREETEXT (Spalten_Name, 'Suchphrase')
```

Reguläre Ausdrücke definieren Suchmuster. Beachten Sie, dass die Syntax im Unterschied zu Perl den senkrechten Strich | als Sonderzeichenmarkierung verwendet:

MATCHES (Spalten_Name, 'RXSuchmuster')

Es gibt auch ein SQL-konformes Suchmuster für eine Suche nach Wortteilen:

Spalten_Name [NOT] LIKE 'Suchmuster'

Folgende Schreibweise gilt für Platzhalter:

- ▶ »_« – ein beliebiges Zeichen
- ▶ »%« – null bis beliebig viele Zeichen
- ▶ »[abcde]« – eine Liste spezifischer Zeichen

Nun muss natürlich noch bekannt sein, welche Spalten überhaupt angezeigt werden können. Eine Auswahl der wichtigsten entnehmen Sie der folgenden Tabelle:

Spalte	Bedeutung
Contents	Dokumentinhalt
Characterization	Zusammenfassung einer Fundstelle. Die Anzahl der Zeichen kann in den Eigenschaften des Katalogs bestimmt werden (siehe Dialogfenster des Katalogmanagers)
Filename	Dateiname
Size	Dateigröße in Bytes
Path	Pfad und Dateiname
Vpath	Virtueller Pfad und Dateiname
HitCount	Anzahl der Fundstellen des Suchwortes im Dokument
Rank	Rang (Wert zwischen 0 und 1.000)
Create	Zeitpunkt, wann die Datei erzeugt wurde
Write	Zeitpunkt, wann die Datei zuletzt beschrieben wurde
DocTitle	Titel des Dokuments (wenn vorhanden)
DocSubject	Betreff (trifft beispielsweise für Word zu)
DocPageCount	Anzahl der Seiten (trifft beispielsweise für Word zu)
DocAuthor	Autor (trifft beispielsweise für Word zu)
DocKeywords	Schlüsselwörter (trifft beispielsweise für Word und HTML zu)
DocComments	Kommentare (trifft beispielsweise für Word zu)

Weitere Detailfragen zum Projekt

Bevor Sie ein solches Projekt anfangen, sind einige Überlegungen zur Umsetzung angebracht. Suchergebnisse kann man in zwei Kategorien teilen: Kein Ergebnis und zu viele Ergebnisse. Die Behandlung des ersten Falles bereitet sicher keine Schwierigkeiten. Der zweite Fall sollte zum Nachdenken anregen. Zur Einschränkung der Suchanfrage stellt der Indexdienst ein reiches Spektrum an Operatoren bereit. Die Aufbereitung kann man für die Benutzer etwas vereinfachen, sodass auch Besucher, die mit AND und OR nichts anfangen können, gute Ergebnisse erzielen.

Fallen immer noch sehr viele Resultate an, muss die Ausgabe gesteuert werden. Zum einen sollte eine seitenweise Ausgabe ermöglicht werden. Zum anderen ist sicherzustellen, dass die einmal abgefragten Ergebnisse beim Wechsel der Seite nicht erneut abgerufen werden. Leider zeigen viele Demonstrationen der ADO.NET-Klassen Derartiges. Das funktioniert zwar auf einem Entwicklungssystem; im Web unter hoher Last jedoch nicht mehr.

**Ausgabe-
aufbereitung**

9.6.2 Der Entwurf eines Suchformulars

Das Formular wird zuerst entworfen. Es besteht aus zwei Teilen:

- Eingabe der Suchwörter
- Anzeige der Suchergebnisse

Listing 9.38 zeigt den ersten Teil des Formulars. Die Besonderheit hier ist die Verknüpfung der Schaltfläche mit einem Klickereignis. In der hinterlegten Code-Klasse, die die Anforderung verarbeitet, wird auf diese Schaltfläche gezielt reagiert. Damit kann für die gesamte Seite nur ein `<form>`-Tag und die eingebaute Formularverwaltung verwendet werden.

```
<form id="Form1" method="post" runat="server">
<table width="100%" border="0"
      style="font-family:Verdana; color:#6600ff">
  <tr>
    <td>Suchbegriff</td>
    <td>
      <asp:TextBox Runat="server" ID="searchWords"
        Width="200px" />
      <asp:RequiredFieldValidator Runat="server"
        ID="searchWordsValidator"
        ControlToValidate="searchWords">
```

```

                ErrorMessage="Sie müssen ein Suchbegriff
                               eingeben"/>
            </td>
        </tr>
        <tr>
            <td>Optionen</td>
            <td>
                <asp:DropDownList Runat="server" ID="searchOptions"
                                   Width="200px">
                    <asp:ListItem Value="all">Alle Wörter</asp:ListItem>
                    <asp:ListItem Value="any">Beliebige Wörter
                </asp:ListItem>
                    <asp:ListItem Value="natural">Natürliche Sprache
                </asp:ListItem>
                </asp:DropDownList>
            </td>
        </tr>
        <tr>
            <td></td>
            <td>
                <asp:Button Runat="server" ID="send1"
                           OnClick="startSearch" Text="Abfrage starten" />
            </td>
        </tr>
    </table>
</form>

```

Listing 9.38: Das Suchformular zur Abfrage des Indexdienstes (IndexSearchForm.aspx)

Wie es funktioniert

Der zweite Teil stellt die Suchergebnisse dar. Damit die vorbereitete, leere Tabelle nicht stört, wird der gesamte Teil in ein Panel-Steuerelement eingebaut, das anfangs unsichtbar ist:

```
<asp:Panel Runat="server" ID="searchResults" Visible="false">
```

Das eigentlich spannende Steuerelement folgt nach der Anzeige der Anzahl der Ergebnisse mit dem `DataGrid` (mehr zur Datenausgabe finden Sie in Listing 9.39). Interessant sind hier mehrere Abschnitte. Zum einen werden im Kopf des Elementes verschiedene Attribute gesetzt. Für die Anwendung wichtig ist `PageSize`, hiermit wird die Anzahl der Ergebnisse pro Seite bestimmt. Außerdem wird mit `AllowPaging="true"` die seitenweise Ausgabe ermöglicht. Dann werden die Formatierungen für die Abschnitte verwendet: `<PagerStyle>` bestimmt die Anzeige der Elemente zum Blättern zwischen den Seiten. Mit `<HeaderStyle>` wird der Kopf der Tabelle formatiert. Die eigentlichen Elemente können in `<ItemStyle>` formatiert werden. Damit allein ist freilich noch keine Anzeige möglich. Dem `DataGrid` muss erst mitgeteilt werden, welche Spalten

der Datenquelle angezeigt werden sollen. Die Datenquelle wird später mit den Ergebnissen der Abfrage des Indexdienstes gefüllt. Wenn Spalten direkt verwendet werden, ohne weitere Gestaltung, eignet sich das Steuerelement `<asp:boundcolumn>`, das nur innerhalb eines `DataGrid` auftreten darf. In der Anwendung erfolgt damit die Anzeige der Relevanz. Drei Attribute kommen zum Einsatz: `DataField` bestimmt die Spalte, `DataFormatString` die Formatierung und `HeaderText` den Spaltenkopf. Für die Spalte mit den Suchergebnissen ist `BoundColumn` zu primitiv. Hier wird eine komplette Vorlage im Tag `<ItemTemplate>` entworfen. Die Bindung an die Datenquelle erfolgt mit der üblichen Syntax in den `<%# ... %>`-Tags. Beachten Sie hier den Aufruf der Methode *Decode*, die eigens für die Abfrage von Suchergebnissen programmiert wurde. Diese Methode dekodiert HTML für die Ausgabe. Wenn Sie nämlich Webseiten durchsuchen und sich vom Indexdienst eine Zusammenfassung liefern lassen, werden HTML-Codes übernommen, meist Ausschnittweise. Das ist der sicherste Weg, das Layout des Formulars zu zerstören. Die Dekodierung verhindert solche lästigen Nebeneffekte.

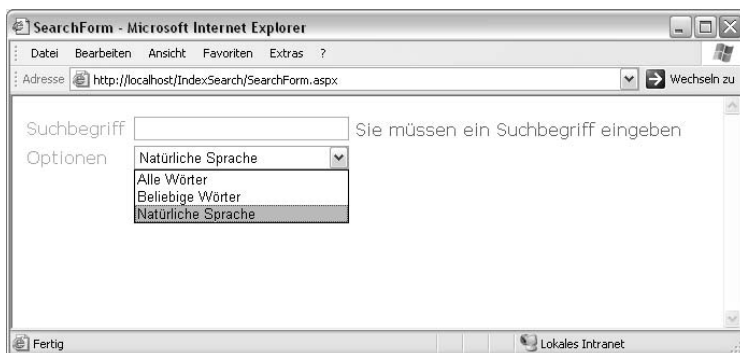


Abbildung 9.34: Das Suchformular in Aktion

9.6.3 Ausführung der Suche

Im zweiten Schritt geht es nun um die Ausführung der Suche. Zuerst finden Sie das Listing auf einen Blick, die wichtigsten Passagen werden nachfolgend erläutert.

```
Imports System.Data
Imports System.Data.OleDb
```



```

AND ",                ""_vti OR .xml""")
    Select Case sOptions
        Case "natural"
            query += String.Format("(FREETEXT (Contents, '{0}') ␣
                                OR FREETEXT (DocTitle, '{0}')) ", ␣
                                searchWords.Text)

        Case Else
            query += String.Format("(CONTAINS (Contents, '{0}') ␣
                                OR CONTAINS (DocTitle, '{0}')) ", ␣
                                searchWords.Text)

    End Select
    query += String.Format("ORDER BY {0} {1}", "Rank", "DESC")
    createQuery = query
End Function

Public Sub startSearch(ByVal sender As Object, ␣
                    ByVal e As EventArgs)
    Dim sCatalog As String = "Web"
    Dim query As String = createQuery()
    searchResults.Visible = True
    oDS = New DataSet()
    numberResults.Text = "keine"
    oConn = New OleDbConnection("Provider=MSIDXS; ␣
                                Data Source=" & sCatalog)

    Try
        oAD = New OleDbDataAdapter(New OleDbCommand(query,
oConn))
        oAD.Fill(oDS, "IndexResult")
        Dim iResults As Integer =
oDS.Tables("IndexResult").Rows.Count
        If iResults > 0 Then
            numberResults.Text = iResults.ToString()
            resultTable.CurrentPageIndex = 0
            resultTable.DataSource = oDS
            resultTable.DataBind()
        Else
            oDS = Nothing
        End If
        Session("custDS") = oDS
        Catch oe As OleDbException
            errorText.Text = "FEHLER beim Datenzugriff: " + oe.Message
    End Try
    oConn.Close()
End Sub

```

```

        Public Sub changeIndex(ByVal sender As Object, 
                                ByVal e As
                                EventArgs)
            oDS = CType(Session("custDS"), DataSet)
            resultTable.CurrentPageIndex = e.NewPageIndex
            resultTable.DataSource = oDS
            resultTable.DataBind()
        End Sub
    End Class

```

Listing 9.39: Das vollständige Suchprogramm (IndexSearchForm.aspx.vb)

Die einzelnen Methoden der Klasse werden in den nachfolgenden Abschnitten ausführlich vorgestellt.

Ereignisablauf

Das Erstellen von Formularen ist eher lästig. Es ist nichtsdestotrotz wichtig, hierbei Sorgfalt walten zu lassen, denn der Benutzer wird nur mit der von Ihnen entworfenen Oberfläche konfrontiert. Natürlich muss der Code dahinter auch perfekt sein. Ein erster Ansatz soll hier entwickelt werden. Die Ablage der Klasse erfolgt als Code Behind-Datei im Namespace *Addison.VBNet.Data* und dem Klassennamen *SearchForm*. Sie erbt – wie alle Klassen zur Seitensteuerung – von *Page*:

```

Public Class SearchForm
    Inherits System.Web.UI.Page

```

Für die Steuerung der Suchergebnisse wird eine Sitzungsvariable verwendet. Damit wird sichergestellt, dass die einmal erfolgten Abfragen während der Blättervorgänge im Speicher erhalten bleiben. So können wertvolle Ressourcen gespart werden, auch wenn das direkte und wiederholte Abfragen des Indexdienstes etwas einfacher möglich wäre. Der Eintritt in den Code erfolgt an zwei Stellen. Wenn eine neue Suchanfrage gestartet wird, beginnt die Ereignisverarbeitung mit der Methode *startSearch*. Wird dagegen im Ergebnis geblättert, erledigt *changeIndex* den Auftrag. Der naive Ansatz für derartige Datenbankprobleme nutzt fast immer die Methode *Page_Load*. Im vorliegenden Fall ist das jedoch nicht sinnvoll, denn *Page_Load* wird vor den Ereignisbehandlungsmethoden ausgeführt. Wenn die Daten einmal feststehen, kann ein folgendes

Ereignis daran wenig ändern. Andererseits werden die entscheidenden Schritte erst möglich, wenn ein spezifisches Ereignis ausgelöst wurde. Deshalb kann hier auf `Page_Load` verzichtet werden.

Suchen im Katalog

Der Code setzt voraus, dass im Indexdienst ein Katalog mit dem Namen »Web« eingerichtet wurde und durchsuchbar ist. Der Katalog durchsucht üblicherweise den Pfad ab `%systemroot%\inetpub` abwärts. Wenn Sie mehrere Domänen hosten, sollten Sie daran denken, jedem Stammpfad einen eigenen Katalog zu spendieren, sonst durchsuchen sich die Präsenzen gegenseitig. Die Auswahl des Kataloges erfolgt in der Verbindungszeichenfolge des OleDb-Providers. Der vom Programm für die Abfrage erzeugte SQL-Code bezieht sich dann auf diesen Katalog.

**Den Katalog
verwenden**

Bevor die Feinheiten der SQL-Abfrage diskutiert werden, sollte das allgemeine Prinzip des Datenzugriffs auf den Indexdienst klar sein. Verantwortlich ist die Methode `startSearch` (Listing 9.39). Dort wird zuerst die Abfrage mit der Methode `createQuery` ermittelt. Dann wird der Ergebnisbereich sichtbar gemacht, in dem der Eigenschaft `Visible` des Panel-Steuerelements `True` zugewiesen wird. Nun wird ein neues `DataSet`-Objekt erzeugt:

```
Dim query As String = createQuery()
searchResults.Visible = True
oDS = New DataSet()
```

Jetzt ist es an der Zeit, die Verbindung zum Indexdienst zu öffnen. Der Name des Providers steht fest; als Datenquelle wählen Sie einen der Kataloge aus, die Sie durchsuchen möchten:

```
oConn = New OleDbConnection("Provider=MSIDXS; ➤
                             Data Source=" & sCatalog)
```

Dann wird ein Datenadapter erzeugt, der mit einem Kommando-Objekt verknüpft wird und die Abfragedaten enthält:

```
oAD = New OleDbDataAdapter (New OleDbCommand (query, oConn))
```

Im nächsten Schritt müssen die Daten dem `DataSet` als Tabelle hinzugefügt werden. Die Abfrage wird hier willkürlich mit »Index-Result« bezeichnet. Von Vorteil ist die Art der Datenhaltung. Die Abfrageergebnisse werden im lokalen Speicher des Webserverns vorgehalten und nachfolgende Zugriffe darauf erfordern keine er-

neute Verbindung zur Datenbank. `DataSet`-Objekte sparen also Ressourcen. Ein weiterer wichtiger Vorteil ist die einfache Ausgabe der Daten des Inhalts eines `DataSet` in einem `DataGrid`-Steuerelement. Ganze zwei Zeilen sind letztendlich dafür verantwortlich:

```
resultTable.DataSource = oDS  
resultTable.DataBind()
```

Vorbereiten der Abfrage

Bislang kam die eigentliche Abfrage in der Variablen *query* aus der geheimnisvollen Methode *createQuery*. Diese erledigt mehrere Aufgaben. Zum einen wird hier die SQL-Abfrage in der korrekten Syntax erzeugt, abhängig von der Auswahl der Optionsliste (siehe Abbildung 9.34). Zum anderen wird mit Hilfe der Methode *searchCondition* aus den eingetippten Suchwörtern ein den Regeln von SQL entsprechender Ausdruck geformt. Das Listing 9.39 zeigt die beiden Methoden. In *createQuery* erfolgt auch der Zugriff auf die Auswahlliste. Je nach dem, ob hier die Option »any« oder »all« übermittelt wird, werden die einzelnen Suchbegriffe mit OR oder AND verknüpft.

In *startSearch* wird dann die SQL-Anweisung zusammengesetzt. Es werden hier zwei Prädikate verwendet: `CONTAINS` und `FREETEXT`. Beide führen eine Volltextsuche aus, wobei `FREETEXT` einen natürlichsprachigen Satz akzeptiert und versucht, daraus relevante Daten zu extrahieren. Zusammengebaut wird die Abfrage aus einer Liste interessanter Spalten (`SELECT`) im gesamten Bereich des Katalogs (`SCOPE()`) und unter Ausschluss von Pfaden, die *_vti* oder *.xml* enthalten. Die *_vti*-Dateien sind Hilfsdateien für FrontPage-Erweiterungen, die keine relevanten Daten enthalten und das Suchergebnis nur unnötig aufblähen würden.

Durchsucht werden die Spalten `Content` (gesamten Inhalt eines Dokuments) und `DocTitle` (Titel, wenn vorhanden).

Speichern des DataSet

Umgang mit mehrseitiger Ausgabe

Damit beim Blättern kein erneuter Zugriff erfolgt, wird das Abfrageergebnis im Speicher gehalten. Hierzu sind Sessionvariablen erforderlich. Das *SearchForm*-Objekt wird normalerweise am Ende der Seite zerstört, lokale Variableninhalte und damit das `DataSet`-

Objekt gehen verloren. Statische Felder behalten die Daten zwar im Kontext der Klasse, unterscheiden dann aber nicht zwischen Benutzern bzw. Sessions. Dieser Weg ist eher für globale Daten geeignet. Ein interessanter Aspekt übrigens. Denn Sie könnten trickreich Standardabfragen im Speicher vorhalten. Vorerst fehlt noch der Schritt, in dem die Daten tatsächlich Teil der Session werden. Dank ASP.NET ist dies extrem einfach:

```
Session("custDS") = oDS
```

Diese einfache Zeile verbirgt viele der inneren Werte. So stammt die Methode aus der Klasse `HttpSessionState`, die über `Page` automatisch verfügbar ist. Außerdem ist der Zugriff auf die Elemente als Standardeigenschaft definiert, weshalb die kurze Schreibweise mit den runden Klammern funktioniert. Und zu guter Letzt ist der Datentyp der Eigenschaft `Object` – damit kann jedes Objekt gespeichert werden, natürlich auch `DataSet`. Das Zurückholen erfolgt übrigens in der Ereignisbehandlungsmethode `changeIndex`, denn nur bei Änderung der anzuzeigenden Seite ist ein erneuter Zugriff erforderlich. Hier wird dann auf das `Session`-Objekt zugegriffen; der Inhalt muss allerdings noch in den richtigen Datentyp gebracht werden:

```
oDS = CType(Session("custDS"), DataSet)
```

Der Rest der Methode `changeIndex` ist schnell behandelt. Zuerst wird der Seitenzeiger neu gesetzt, damit das `DataGrid` den richtigen Ausschnitt aus den Suchergebnissen anzeigt. Dann erfolgt wieder die Bindung der Daten und die Anzeige ist aktualisiert:

```
resultTable.CurrentPageIndex = e.NewPageIndex  
resultTable.DataSource = oDS  
resultTable.DataBind()
```

Die kleine Applikation zeigt, wie mit nur wenigen Zeilen Code eine komfortable Suchmaschine auf Basis des Indexdienstes entwickelt werden kann. Dabei wird einfach der `OleDb`-Provider verwendet, der mit Windows installiert wurde. Ein solcher Provider steht für viele andere Datenverarbeitungsprogramme ebenfalls zur Verfügung, sodass der Einsatz zusammen mit ASP.NET sehr leicht möglich ist. Vor allem die Daten-Steuerelemente wie `DataGrid` lassen die Ausgabe dann mit einfachen Mitteln möglich werden.

Fazit

Suchbegriff	Wer waren Pink Floyd?																						
Optionen	Natürliche Sprache																						
Abfrage starten																							
Es wurden 87 Ergebnisse gefunden																							
1 2 3 4 5 6 7 8 9 <table border="1"> <thead> <tr> <th>Relevanz</th> <th>Kurzinfo</th> </tr> </thead> <tbody> <tr> <td>31.0%</td> <td>Q Magazine, november 1992 Zurück. Q Magazine - Waters Story, 1992. How did it go again? "We dahn nee nur edercayshun. Yes, that was it! "We dahn nee nur fort comtrawel!" It's good to know that in these days of silly disposable pop rubbish, there remains one man brave and brilliant enough to address the Really Big Questions. Questions, sugge</td> </tr> <tr> <td>27.1%</td> <td>Pink Floyd - Über den Autor</td> </tr> <tr> <td>25.8%</td> <td>Stranger Than Fiction Stranger Than Fiction The Ultimate Early Pink Floyd Collection (1968-1972). Images. Album Notes. Track Number. Title. Venue. Date Recorded. Time. Notes. 1. Corrosion. Royal Albert Hall. 6.2.1971. 8:32. 2. Pink Blues. Stadthalle Offenbach. 26.2.1971. 5:45. 3. Careful With That Axe Eugene. 28.6.70. 10:49.</td> </tr> <tr> <td>24.8%</td> <td>Syd Barrett Story from RQIQ-LP Zurück. SYD BARRETT. THERE IS a story that exists pertaining to an incident which occurred during one of Syd Barrett's last gigs with the Pink Floyd. After a lengthy interval, the band decided to take the stage (there is a certain amount of dispute as to which venue this all took place at) - all</td> </tr> <tr> <td>21.0%</td> <td>Interview taken from Q #48 September 90, with David Gilmour Zurück. THE RIGHTFUL HEIR?. Twenty-five years ago he was just the hired hand. Then he became Syd Barrett's full-time replacement. By 1985, following group leader Roger Waters' traumatic exit, David Gilmour had emerged as their unofficial supremo. But the fight for the Pink Floyd legacy still rages. "We</td> </tr> <tr> <td>21.0%</td> <td>Billboard Magazine, July 1992 Zurück. Roger Waters' 'Death' & Rebirth. It is the poet's responsibility to foresee the future, and it is his neighbor's duty to prevent the worst of it from taking place. With 'Amused To Death, surely one of the most provocative and musically dazzling records of the decade, Roger Waters has fulfilled his part of t</td> </tr> <tr> <td>19.4%</td> <td>Excerpts from an interview with Snowy White Zurück. Excerpts from an interview with Snowy White. Note.: T is the interviewer S is Snowy White K is Kumar Harada who plays in Snowy White's Blues Agency. T : Right. You toured with Pink Floyd on the Animals Tour and on the Wall Tour. How did you get that? How did that come about?S : They phoned up. The M</td> </tr> <tr> <td>17.4%</td> <td>Syd Barrett. Roger Keith Barrett - geboren am 6. Januar 1946 - verlebte eine völlig normale Kindheit in Cambridge. Er wuchs in einem großen Haus an der Hill Road auf der schönsten Straße von Cherry Hinton, in der Obhut warmherziger, liebender Eltern. Er war ein beliebter und guter Schüler. Seine Interessen reich</td> </tr> <tr> <td>16.9%</td> <td>Roger Waters Interview w/Chris Salewicz, June 1987 Zurück. Roger Waters Interview w/Chris Salewicz, June 1987. On June 15th, 1987 Roger Waters released his second solo album entitled "Radio KAOS. I believe this interview took place short after its release and before the Radio KAOS Tour kicked off at Rhode Island, on august 14 1987. Roger Waters (RW) was interv</td> </tr> <tr> <td>16.4%</td> <td>PH-2 Autor. Zur Person. Jahrgang 1964 Deutsch Verheiratet, 1 Kind (11 Jahre) Ausbildung an der Humboldt Universität Berlin, Studienfach: Technische Informatik; ohne Abschluss ; EDV-Erfahrung seit 1980 (ZX81) Solides Basiswissen: Chiptechnik, ElektrJörg Krause. Einsatz und Eignung. Ich biete Ihnen Software</td> </tr> </tbody> </table>		Relevanz	Kurzinfo	31.0%	Q Magazine, november 1992 Zurück. Q Magazine - Waters Story, 1992. How did it go again? "We dahn nee nur edercayshun. Yes, that was it! "We dahn nee nur fort comtrawel!" It's good to know that in these days of silly disposable pop rubbish, there remains one man brave and brilliant enough to address the Really Big Questions. Questions, sugge	27.1%	Pink Floyd - Über den Autor	25.8%	Stranger Than Fiction Stranger Than Fiction The Ultimate Early Pink Floyd Collection (1968-1972). Images. Album Notes. Track Number. Title. Venue. Date Recorded. Time. Notes. 1. Corrosion. Royal Albert Hall. 6.2.1971. 8:32. 2. Pink Blues. Stadthalle Offenbach. 26.2.1971. 5:45. 3. Careful With That Axe Eugene. 28.6.70. 10:49.	24.8%	Syd Barrett Story from RQIQ-LP Zurück. SYD BARRETT. THERE IS a story that exists pertaining to an incident which occurred during one of Syd Barrett's last gigs with the Pink Floyd. After a lengthy interval, the band decided to take the stage (there is a certain amount of dispute as to which venue this all took place at) - all	21.0%	Interview taken from Q #48 September 90, with David Gilmour Zurück. THE RIGHTFUL HEIR?. Twenty-five years ago he was just the hired hand. Then he became Syd Barrett's full-time replacement. By 1985, following group leader Roger Waters' traumatic exit, David Gilmour had emerged as their unofficial supremo. But the fight for the Pink Floyd legacy still rages. "We	21.0%	Billboard Magazine, July 1992 Zurück. Roger Waters' 'Death' & Rebirth. It is the poet's responsibility to foresee the future, and it is his neighbor's duty to prevent the worst of it from taking place. With 'Amused To Death, surely one of the most provocative and musically dazzling records of the decade, Roger Waters has fulfilled his part of t	19.4%	Excerpts from an interview with Snowy White Zurück. Excerpts from an interview with Snowy White. Note.: T is the interviewer S is Snowy White K is Kumar Harada who plays in Snowy White's Blues Agency. T : Right. You toured with Pink Floyd on the Animals Tour and on the Wall Tour. How did you get that? How did that come about?S : They phoned up. The M	17.4%	Syd Barrett. Roger Keith Barrett - geboren am 6. Januar 1946 - verlebte eine völlig normale Kindheit in Cambridge. Er wuchs in einem großen Haus an der Hill Road auf der schönsten Straße von Cherry Hinton, in der Obhut warmherziger, liebender Eltern. Er war ein beliebter und guter Schüler. Seine Interessen reich	16.9%	Roger Waters Interview w/Chris Salewicz, June 1987 Zurück. Roger Waters Interview w/Chris Salewicz, June 1987. On June 15th, 1987 Roger Waters released his second solo album entitled "Radio KAOS. I believe this interview took place short after its release and before the Radio KAOS Tour kicked off at Rhode Island, on august 14 1987. Roger Waters (RW) was interv	16.4%	PH-2 Autor. Zur Person. Jahrgang 1964 Deutsch Verheiratet, 1 Kind (11 Jahre) Ausbildung an der Humboldt Universität Berlin, Studienfach: Technische Informatik; ohne Abschluss ; EDV-Erfahrung seit 1980 (ZX81) Solides Basiswissen: Chiptechnik, ElektrJörg Krause. Einsatz und Eignung. Ich biete Ihnen Software
Relevanz	Kurzinfo																						
31.0%	Q Magazine, november 1992 Zurück. Q Magazine - Waters Story, 1992. How did it go again? "We dahn nee nur edercayshun. Yes, that was it! "We dahn nee nur fort comtrawel!" It's good to know that in these days of silly disposable pop rubbish, there remains one man brave and brilliant enough to address the Really Big Questions. Questions, sugge																						
27.1%	Pink Floyd - Über den Autor																						
25.8%	Stranger Than Fiction Stranger Than Fiction The Ultimate Early Pink Floyd Collection (1968-1972). Images. Album Notes. Track Number. Title. Venue. Date Recorded. Time. Notes. 1. Corrosion. Royal Albert Hall. 6.2.1971. 8:32. 2. Pink Blues. Stadthalle Offenbach. 26.2.1971. 5:45. 3. Careful With That Axe Eugene. 28.6.70. 10:49.																						
24.8%	Syd Barrett Story from RQIQ-LP Zurück. SYD BARRETT. THERE IS a story that exists pertaining to an incident which occurred during one of Syd Barrett's last gigs with the Pink Floyd. After a lengthy interval, the band decided to take the stage (there is a certain amount of dispute as to which venue this all took place at) - all																						
21.0%	Interview taken from Q #48 September 90, with David Gilmour Zurück. THE RIGHTFUL HEIR?. Twenty-five years ago he was just the hired hand. Then he became Syd Barrett's full-time replacement. By 1985, following group leader Roger Waters' traumatic exit, David Gilmour had emerged as their unofficial supremo. But the fight for the Pink Floyd legacy still rages. "We																						
21.0%	Billboard Magazine, July 1992 Zurück. Roger Waters' 'Death' & Rebirth. It is the poet's responsibility to foresee the future, and it is his neighbor's duty to prevent the worst of it from taking place. With 'Amused To Death, surely one of the most provocative and musically dazzling records of the decade, Roger Waters has fulfilled his part of t																						
19.4%	Excerpts from an interview with Snowy White Zurück. Excerpts from an interview with Snowy White. Note.: T is the interviewer S is Snowy White K is Kumar Harada who plays in Snowy White's Blues Agency. T : Right. You toured with Pink Floyd on the Animals Tour and on the Wall Tour. How did you get that? How did that come about?S : They phoned up. The M																						
17.4%	Syd Barrett. Roger Keith Barrett - geboren am 6. Januar 1946 - verlebte eine völlig normale Kindheit in Cambridge. Er wuchs in einem großen Haus an der Hill Road auf der schönsten Straße von Cherry Hinton, in der Obhut warmherziger, liebender Eltern. Er war ein beliebter und guter Schüler. Seine Interessen reich																						
16.9%	Roger Waters Interview w/Chris Salewicz, June 1987 Zurück. Roger Waters Interview w/Chris Salewicz, June 1987. On June 15th, 1987 Roger Waters released his second solo album entitled "Radio KAOS. I believe this interview took place short after its release and before the Radio KAOS Tour kicked off at Rhode Island, on august 14 1987. Roger Waters (RW) was interv																						
16.4%	PH-2 Autor. Zur Person. Jahrgang 1964 Deutsch Verheiratet, 1 Kind (11 Jahre) Ausbildung an der Humboldt Universität Berlin, Studienfach: Technische Informatik; ohne Abschluss ; EDV-Erfahrung seit 1980 (ZX81) Solides Basiswissen: Chiptechnik, ElektrJörg Krause. Einsatz und Eignung. Ich biete Ihnen Software																						
1 2 3 4 5 6 7 8 9																							

Abbildung 9.35: Erfolgreiche Abfrage des Indexdienstes mit ASP.NET und seitenweiser Ausgabe der Ergebnisse

C Professionelle Techniken in ASP.NET

10 Konfiguration, Optimierung und Sicherheit

Die Themen Konfiguration, Optimierung und Sicherheit spielen bei der täglichen Programmierung keine direkte Rolle, denn es handelt sich meist um einmalige Aufgaben. Das ändert nichts an der Bedeutung, die sie für stabile Applikationen haben. Dieses Kapitel gibt Anregungen, welchen Methoden es in ASP.NET gibt und welche elementaren Schritte erforderlich sind.

10.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

10.1.1 Über dieses Kapitel

In diesem Kapitel erfahren Sie mehr über die Konfiguration von Seiten, Applikationen und Servern. ASP.NET basiert auf einem dreistufigen Konfigurationsmodell. Durch so genannte Direktiven kann das Verhalten einer *aspx*-Seite gesteuert werden. Für Benutzer-Steuerelemente und Web-Services gibt es spezielle Direktiven. Abschnitt 10.2, »Konfiguration einzelner Seiten: Die Direktiven« ab Seite 954 behandelt dieses Thema.

Die Konfiguration einer Applikation basiert – ebenso wie die der gesamten Maschine – auf Konfigurationsdateien. Diese liegen im XML-Format vor und lassen sich sehr leicht bearbeiten. Einen Überblick finden Sie in Abschnitt 10.3, »Konfiguration von Applikationen: web.config« ab Seite 962. Weitere Details sind jedoch in

den Abschnitten des Buches zu finden, in denen bestimmte Teilfunktionen konfiguriert werden müssen.

Die Sicherheit von Webanwendungen und Servern spielt heute eine herausragende Rolle. ASP.NET und das Framework sind hier gut gewappnet. Abschnitt 10.5, »Sicherheit« ab Seite 973 führt in die Techniken der Absicherung von Seiten und Applikationen ein. Sie finden dort auch Informationen über die Programmierung von Anmeldeformularen, die Nutzung der integrierten Sicherheit von Windows 2000/.NET-Server und Methoden der Wiedererkennung von Benutzern.

10.2 Konfiguration einzelner Seiten: Die Direktiven

Viele grundlegende Operationen lassen sich über Seitendirektiven steuern. Diese stehen am Anfang der Seite und gelten für Übersetzung oder Ausführung. Die Direktiven sollten immer an den Anfang der Seite gesetzt werden, auch wenn dies nicht in allen Fällen zwingend notwendig ist.

10.2.1 Übersicht

Folgende Direktiven sind verfügbar:

► @Page

Hiermit werden Vorgänge gesteuert, die unmittelbar mit der Ausführung der Seite zu tun haben.

► @Control

Mit dieser Direktive werden Benutzer-Steuerelemente kontrolliert. Die Attribute sind mit @Page vergleichbar.

► @Import

Mit dieser Direktive werden Namensräume aus dem .NET-Framework importiert. Dies ist notwendig, wenn Code Behind nicht verwendet wird, da in ASP.NET-Vorlagen die Anweisung `Imports` nicht eingesetzt werden kann, mit der üblicherweise Namensräume importiert werden.

► @Implements

Diese Direktive importiert eine Schnittstellenbeschreibung (Interface) aus dem .NET-Framework.

► @Register

Hiermit werden Benutzer-Steuerelemente (User Controls) angemeldet, sodass der Compiler diese finden und einbinden kann.

► @Assembly

Die Nachfolger der DLLs in Windows sind in .NET Assemblies. Sie enthalten praktisch bereits übersetzten Code in der Intermediate Language. Mit dieser Direktive können Sie Assemblies direkt nutzen.

► @OutputCache

Jede Seite, die auf dem Server erzeugt wird, kann zwischengespeichert werden. Dies erhöht die Systemleistung und verringert die Prozessorlast. Für den korrekten Aufbau der Seite im Browser ist die Kontrolle dieses Vorgangs wichtig. Die Direktive steuert dies.

► @Reference

Dieses Element stellt für die Verbindung von Seiten bei der Steuerungsweitergabe mit `Server.Transfer` die Referenz her.

Die wichtigsten Direktiven werden nachfolgend vorgestellt – in dem Rahmen, in dem sie in diesem Buch Verwendung finden. In einigen Fällen erfolgte eine Erläuterung im Zusammenhang mit der Anwendung. Dann wird auf die entsprechenden Abschnitte verwiesen.

10.2.2 Die Seitendirektive @Page

Die Seitendirektive @Page definiert Bedingungen zur Abarbeitung einer Seite. Sie steht immer am Anfang einer *aspx*-Seite:

```
<%@Page Attribut="Wert" ... %>
```

Die wichtigsten Attribute finden Sie in der folgenden Tabelle:

Attribut	Parameter	Bedeutung
AutoEventWireUp	True, False	Leitet Ereignisse automatisch weiter, siehe Text nach der Tabelle.
Buffer	True, False	Steuert den Ausgabepuffer ein. True ist der Standardwert.

Tabelle 10.1: Die wichtigsten Attribute der Direktive @Page

Attribut	Parameter	Bedeutung
ClassName	Name	Name einer Klasse, die für diese Seite übersetzt wird, wenn mehrere Klassen zu Auswahl stehen.
ContentType	z. B. image/gif	Der Name eines MIME-Typs, mit dem der Inhalt gesendet werden soll.
Debug	True, False	Schaltet serverseitiges Debuggen ein. Der Standardwert ist False (aus).
Trace	True, False	Schaltet serverseitiges Ablaufverfolgung ein. Der Standardwert ist False (aus).
Description	Text	Infotext zur Seite, der von ASP.NET ignoriert wird.
Inherits	Name	Name einer Klasse einer hinterlegten Code-Datei (Code Behind).
Src	Name, URL	Quelle einer Code-Datei (Code Behind)
Language	VB, C#, JScript.NET	Name der Sprache, die im Inline-Code (<%>-Tags) erwartet wird.
Culture	Name	Name des Sprachpaketes, z. B. »de-DE«, zur Einstellung der Währung, Zahlenformate usw.

Tabelle 10.1: Die wichtigsten Attribute der Direktive @Page (Forts.)

AutoEventWireUp

Die Aktivierung der automatischen Ereignisverarbeitung ist notwendig, wenn Sie mit der Technik arbeiten, bei der Ereignisbehandlungsmethoden lediglich als `Public` deklariert und dann automatisch vom Ereignis eines Steuerelements gefunden werden. Der Visual Studio .NET-Designer, der im Buch zum Schreiben der Beispiele oft verwendet wurde, arbeitet etwas anders. Hier werden die Anweisung `Handles` und der Modifizierer `WithEvents` verwendet.

Einsatz der Seitendirektive zur Fehlersuche

Debug

Die Attribute `Debug` und `Trace` helfen bei der Fehlersuche. Mit `Debug` wird der Ablauf des Programms auf der Serverseite überwacht. Tritt ein Fehler auf, kann ASP.NET detaillierte Informationen über die Ursache bereitstellen. In Produktionsumgebungen wird das Attribut entfernt, weil die Überwachung Leistung kostet. Wenn Sie in manchen Situationen unzureichende Informationen über den Fehler finden oder die Angabe der Zeile mit der Ursache fehlt, fügen Sie der Seitendirektive `debug="True"` hinzu.

Server Error in '/aspdotnet' Application.

Could not find a part of the path "C:\WINNT\system32\xx".

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.IO.DirectoryNotFoundException: Could not find a part of the path "C:\WINNT\system32\xx".

Source Error:

```
Line 23:         ausgabe.Text += "<br/>Erzeugt am : " + dinfo.CreationTime.ToShortDateString();
Line 24:         ausgabe.Text += "<br/><u>Unterverzeichnisse</u>";
Line 25:         foreach (DirectoryInfo subdir in dinfo.GetDirectories())
Line 26:         {
Line 27:             ausgabe.Text += "<br/>" + subdir.FullName;
```

Source File: C:\inetpub\wwwroot\aspdotnet\readdir.cs Line: 25

Stack Trace:

```
[DirectoryNotFoundException: Could not find a part of the path "C:\WINNT\system32\xx".]
System.IO._Error.WinIOError(Int32 errorCode, String str) +287
System.IO.Directory.InternalGetFileDirectoryNames(String fullPath, Boolean file) +251
System.IO.DirectoryInfo.GetDirectories(String searchPattern) +382
System.IO.DirectoryInfo.GetDirectories() +13
dir.Page_Load() in C:\inetpub\wwwroot\dotnet\readdir.cs:25
System.Web.Util.ArglessEventHandlerDelegateProxy.Callback(Object sender, EventArgs e) +10
System.Web.UI.Control.OnLoad(EventArgs e) +67
System.Web.UI.Control.LoadRecursive() +29
System.Web.UI.Page.ProcessRequestMain() +724
```

Version Information: Microsoft .NET Framework Version:1.0.3705.0; ASP.NET Version:1.0.3705.0

Session Id:	ztulde45fze5ixmkbfiaywt45	Request Type:	POST
Time of Request:	21.4.2002 19:28:07	Status Code:	500
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

Category	Message	From First (s)	From Last (s)
aspx:page	Begin Init		
aspx:page	End Init	0,000090	0,000090
aspx:page	Begin LoadViewState	0,000528	0,000437
aspx:page	End LoadViewState	0,001009	0,000481
aspx:page	Begin ProcessPostData	0,001080	0,000071
aspx:page	End ProcessPostData	0,001150	0,000071
Unhandled Execution Error	Could not find a part of the path "C:\WINNT\system32\xx". at System.IO._Error.WinIOError(Int32 errorCode, String str) at System.IO.Directory.InternalGetFileDirectoryNames(String fullPath, Boolean file) at System.IO.DirectoryInfo.GetDirectories(String searchPattern) at System.IO.DirectoryInfo.GetDirectories() at dir.Page_Load() in C:\inetpub\wwwroot\dotnet\readdir.cs:line 25 at System.Web.Util.ArglessEventHandlerDelegateProxy.Callback(Object sender, EventArgs e) at System.Web.UI.Control.OnLoad(EventArgs e) at System.Web.UI.Control.LoadRecursive() at System.Web.UI.Page.ProcessRequestMain()	0,001811	0,000660

Control Id	Type	Render Size Bytes (including children)	Viewstate Size Bytes (excluding children)

Application Key	Type	Value
HitCounter	System.Int32	1
Version	System.String	0.9.3

Name	Value	Size
ASP.NET_SessionId	ztulde45fze5ixmkbfiaywt45	42

Name	Value
Cache-Control	no-cache
Connection	Keep-Alive
Content-Length	185
Content-Type	application/x-www-form-urlencoded
Accept	*//*
Accept-Encoding	gzip, deflate
Accept-Language	de,en-us;q=0.8,en-au;q=0.5,en-gb;q=0.3
Authorization	Basic QWRtaW5pc3RyYXVvcjpbGVTZW5z
Cookie	ASP.NET_SessionId=ztulde45fze5ixmkbfiaywt45
Host	localhost
Referer	http://localhost/aspdotnet/file_readdir.aspx
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)

Abbildung 10.1: Ausgabe eines Laufzeitfehlers mit trace="True"

Name	Value
__VIEWSTATE	dDwxMDgwMDESzY4O3Q8O2w8aT.wzPjs+O2w8dDxwPHA8bDxUZxh0Oz47bDxHZWJlb8TaWUgZWlufZlcnplaWNk
dirname	xx
senden	Anzeigen

Name	Value
ALL_HTTP	HTTP_CACHE_CONTROL: no-cache HTTP_CONNECTION: Keep-Alive HTTP_CONTENT_LENGTH: 185 HTTP_CONTENT_TYPE: application/x-www-form-urlencoded HTTP_ACCEPT: */* HTTP_ACCEPT_ENCODING: gzip, deflate HTTP_ACCEPT_LANGUAGE: de,en-us;q=0.8,en-au;q=0.5,en-gb;q=0.3 HTTP_AUTHORIZATION: Basic QWRtaW5pc3RyYXRvcjpbGtZW5z HTTP_COOKIE: ASP.NET_SessionId=ztlude45fze5ixmkbfaywt45 HTTP_HOST: localhost HTTP_REFERER: http://localhost/aspdotnet/file_readdir.aspx HTTP_USER_AGENT: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)
ALL_RAW	Cache-Control: no-cache Connection: Keep-Alive Content-Length: 185 Content-Type: application/x-www-form-urlencoded Accept: */* Accept-Encoding: gzip, deflate Accept-Language: de,en-us;q=0.8,en-au;q=0.5,en-gb;q=0.3 Authorization: Basic QWRtaW5pc3RyYXRvcjpbGtZW5z Cookie: ASP.NET_SessionId=ztlude45fze5ixmkbfaywt45 Host: localhost Referer: http://localhost/aspdotnet/file_readdir.aspx User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)
APPL_MD_PATH	/LM/W3SVC/1/Root/aspdotnet
APPL_PHYSICAL_PATH	C:\inetpub\wwwroot\dotnet\
AUTH_TYPE	Basic
AUTH_USER	W2KWS\Administrator
AUTH_PASSWORD	clemens
LOGON_USER	W2KWS\Administrator
REMOTE_USER	W2KWS\Administrator
CERT_COOKIE	
CERT_FLAGS	
CERT_ISSUER	
CERT_KEYSIZE	
CERT_SECRETKEYSIZE	
CERT_SERIALNUMBER	
CERT_SERVER_ISSUER	
CERT_SERVER_SUBJECT	
CERT_SUBJECT	
CONTENT_LENGTH	185
CONTENT_TYPE	application/x-www-form-urlencoded
GATEWAY_INTERFACE	CGI/1.1
HTTPS	off
HTTPS_KEYSIZE	
HTTPS_SECRETKEYSIZE	
HTTPS_SERVER_ISSUER	
HTTPS_SERVER_SUBJECT	
INSTANCE_ID	1
INSTANCE_META_PATH	/LM/W3SVC/1
LOCAL_ADDR	127.0.0.1
PATH_INFO	/aspdotnet/file_readdir.aspx
PATH_TRANSLATED	C:\inetpub\wwwroot\dotnet\file_readdir.aspx
QUERY_STRING	
REMOTE_ADDR	127.0.0.1
REMOTE_HOST	127.0.0.1
REQUEST_METHOD	POST
SCRIPT_NAME	/aspdotnet/file_readdir.aspx
SERVER_NAME	localhost
SERVER_PORT	80
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.1
SERVER_SOFTWARE	Microsoft-IIS/5.0
URL	/aspdotnet/file_readdir.aspx
HTTP_CACHE_CONTROL	no-cache
HTTP_CONNECTION	Keep-Alive
HTTP_CONTENT_LENGTH	185
HTTP_CONTENT_TYPE	application/x-www-form-urlencoded
HTTP_ACCEPT	*/*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	de,en-us;q=0.8,en-au;q=0.5,en-gb;q=0.3
HTTP_AUTHORIZATION	Basic QWRtaW5pc3RyYXRvcjpbGtZW5z
HTTP_COOKIE	ASP.NET_SessionId=ztlude45fze5ixmkbfaywt45
HTTP_HOST	localhost
HTTP_REFERER	http://localhost/aspdotnet/file_readdir.aspx
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)

Abbildung 10.1: Ausgabe eines Laufzeitfehlers mit `trace="True"` (Forts.)

Trace Wenn Sie noch mehr Informationen über den Gesamtzustand des Systems benötigen, um die Ursache eines Laufzeitfehlers zu finden, ergänzen Sie das Attribut `trace="True"`.

10.2.3 Die Direktive @Import

Mit dieser Direktive werden weitere Namensräume eingebunden. Sie kann so oft angewendet werden, wie Namensräume notwendig sind:

```
<%@Import namespace="System.Data" %>
```

ASP.NET importiert einige Namensräume automatisch, sodass die Anwendung nicht immer notwendig ist. Dies gilt nur für die Seite selbst, nicht für hinterlegten Code.

Folgende Namensräume werden eingebunden:

- ▶ System
- ▶ System.Collections
- ▶ System.Collections.Specialized
- ▶ System.Configuration
- ▶ System.IO
- ▶ System.Text
- ▶ System.Text.RegularExpressions
- ▶ System.Web
- ▶ System.Web.Caching
- ▶ System.Web.Security
- ▶ System.Web.SessionState
- ▶ System.Web.UI
- ▶ System.Web.UI.HtmlControls
- ▶ System.Web.UI.WebControls

Wenn @Import erforderlich ist, wird es in diesem Buch explizit erwähnt.

10.2.4 Die Direktive @Register

Mit dieser Direktive werden Benutzer-Steuerelemente (User Controls) der Seite bekannt gemacht:

```
<%@Register tagprefix="tagprefix" Tagname="tagname" Src="file" %>
```

Festgelegt wird das Präfix, den das Element auf der Seite verwendet sowie der Name und der Pfad, wo die *ascx*-Datei gespeichert wurde. Eine Anwendung finden Sie unter anderem im Abschnitt 7.2, »Modularer Code mit Benutzer-Steuerelementen« ab Seite 647.

Attribute und Parameter	Beschreibung
TagPrefix	Präfix des Steuerelements, der als Alias für den Namensraum gilt
TagName	Name des Steuerelements
Namespace	Namensraum, aus dem das Element stammt. Die Angabe ist optional.
Assembly	Name einer Assembly, die die Definition enthält (Angabe ohne den Suffix ».dll«). Dieses Attribut kann nicht gleichzeitig mit Path verwendet werden.
Path	Pfad, unter dem das Element definiert wurde (Angabe vollständig mit dem Suffix ».ascx«). Dieses Attribut kann nicht gleichzeitig mit Assembly verwendet werden.

10.2.5 Die Direktive @OutputCache

Die Direktive `@OutputCache` steuert das Verhalten des Ausgabezwischenspeichers. Der Zugriff auf ganz oder teilweise statischen Inhalt kann damit beschleunigt werden. Eine nähere Erläuterung der Verfahrensweise finden Sie in Abschnitt 8.7.1, »Caching von Seiten und Steuerelementen« ab Seite 809.

Die folgende Tabelle enthält die zulässigen Attribute und deren Bedeutung.

Attribute und Parameter	Beschreibung
Duration	Dauer, während der die Seite oder das Steuerelement im Cache verbleibt. Angabe in Sekunden.
Location	Optional, nicht für Steuerelemente verwendbar. Bestimmt, welcher physische Speicherort verwendet wird: ► Any Jeder verfügbare Speicher wird verwendet. ► Client Der Speicher des Browsers wird verwendet ► DownStream Der Speicher des Browsers oder ein HTTP 1.1 fähiger Proxy wird verwendet

Tabelle 10.2: Die Attribute der Direktive `@OutputCache`

Attribute und Parameter	Beschreibung
	► None Die Zwischenspeicherung ist deaktiviert ► Server Der Ausgabespeicher befindet sich auf dem Webserver
VaryByCustom	Optional. Entweder die Zeichenkette »browser«, dann variiert der Cache nach dem Client, d. h., jeder Browser oder eine beliebige Zeichenkette, dann ist eine Änderung der Standard-Cachemethoden erforderlich.
VaryByHeader	Optional. Liste von HTTP-Kopfzeilen, die benutzerdefinierte Anforderungen an das Speichergerät enthalten
VaryByParam	Erforderlich. Liste von Zeichenfolgen, die POST- oder GET-Parameter beschreiben, die die Änderung des Cache steuern. Sonderfälle sind: ► none keine Berücksichtigung von Parametern ► * Alle gültigen Parameter werden unterschieden
VaryByControl	Name des Steuerelements, auf das sich der Cache bezieht. Nur zulässig im Steuerelemente-Caching.

Tabelle 10.2: Die Attribute der Direktive *@OutputCache* (Forts.)

Wenn Listen in den Parametern verwendet werden dürfen, sind die Listenelemente durch Kommata zu trennen, soweit nichts anderes in der Beschreibung genannt wird.

10.2.6 Die Direktive @Control

Wenn Sie Benutzer-Steuerelemente entwerfen, erhalten auch diese eine mit @Page vergleichbare Direktive: @Control. Die folgende Tabelle zeigt alle zulässigen Attribute und deren Bedeutung:

Attribute und Parameter	Beschreibung
AutoEvent-WireUp="true false"	Aktiviert die automatische Ereignisverarbeitung
ClassName="Name"	Name der Klasse, in die das Steuerelement kompiliert werden soll

Tabelle 10.3: Die Attribute der Direktive *@Control*

Attribute und Parameter	Beschreibung
CompilerOptions=" "	Compileroptionen
Debug="true false"	Schaltet Debugging ein
Description="Beschreibung"	Beschreibung des Steuerelements
EnableViewState="True False"	Schaltet den Anzeigestatus ein oder aus.
Inherits="KlassenName"	Name der zu verwendenden Klasse aus der hinterlegten Code-Datei
Src="CodeDatei"	Name der Code-Datei
Language="vb"	Sprache des eingebetteten Codes
WarningLevel="0 1 2 4"	Reaktionsstufen des Compilers
Explicit="True False"	Wenn True, müssen alle Variablen deklariert werden. Der Standardwert ist hier False.
Strict="True False"	Wenn True, müssen alle Variablentypen exakt definiert werden. Der Standardwert ist hier False. Entspricht Option Strict in VB.NET-Code.

Tabelle 10.3: Die Attribute der Direktive `@Control` (Forts.)

AutoEventWireUp

Die Aktivierung der automatischen Ereignisverarbeitung ist notwendig, wenn Sie mit der Technik arbeiten, bei der Ereignisbehandlungsmethoden lediglich als `Public` deklariert und dann automatisch vom Ereignis eines Steuerelements gefunden werden. Der Visual Studio .NET-Designer, der im Buch zum Schreiben der Beispiele oft verwendet wurde, arbeitet etwas anders. Hier werden die Anweisungen `Handles` und der Modifizierer `With Events` verwendet.

10.3 Konfiguration von Applikationen: web.config

Um eine Applikation zu konfigurieren, wird mit ASP.NET ein ebenso einfaches wie leistungsfähiges Prinzip eingeführt. Neben einer zentralen Konfigurationsdatei, *machine.config*, die im nächsten Abschnitt beschrieben wird, kann der Betreiber einer Applikation in jedem Verzeichnis eine Datei *web.config* ablegen. Diese XML-Datei enthält eine einfache Struktur von Tags, mit denen vielfältige Konfigurationen möglich sind. Um die Aktivierung

müssen Sie sich nicht kümmern. Es wird weder ein Neustart der Applikation noch des IIS gefordert. Tatsächlich lädt ASP.NET die Datei beim ersten Aufruf und stellt den Inhalt dann aus dem Speicher zur Verfügung. Nur wenn sich Änderungen ergeben, wird sie neu geladen. Insofern ist die ständige Nutzung nicht mit Festplattenzugriffen verbunden, Änderungen wirken sich aber dennoch sofort aus.

Das vollständige Schema der Konfigurationsdatei finden Sie nachfolgend:

```
<configuration>
  <system.web>
    <authentication>
      <forms>
        <credentials>
        <passport>
      <authorization>
        <allow>
        <deny>
      <browserCaps>
        <result>
        <use>
        <filter>
          <case>
        <clientTarget>
          <add>
          <remove>
          <clear>
        <compilation>
          <compilers>
            <compiler>
          <assemblies>
            <add>
            <remove>
            <clear>
        <customErrors>
          <error>
        <globalization>
        <httpHandlers>
          <add>
          <remove>
          <clear>
        <httpModules>
          <add>
          <remove>
          <clear>
        <httpRuntime>
```

```

<identity>
  <machineKey>
  <pages>
  <processModel>
  <securityPolicy>
    <trustLevel>
  <sessionState>
  <trace>
  <trust>
  <webServices>
    <protocols>
      <add>
      <remove>
      <clear>
    <serviceDescriptionFormatExtensionTypes>
      <add>
      <remove>
      <clear>
    <soapExtensionTypes>
      <add>
    <soapExtensionReflectorTypes>
      <add>
    <soapExtensionImporterTypes>
      <add>
    <Wsd1HelpGenerator>
  </webServices>
</system.web>
</configuration>

```

Die wichtigsten Abschnitte werden nachfolgend vorgestellt. Einige Teile wirken jedoch auf ganz bestimmte Programmfunktionen und werden in den entsprechenden Abschnitten beschrieben:

- Das Element `<authentication>` wird in Abschnitt 10.5, »Sicherheit« ab Seite 973 beschrieben.
- Information zu `<webServices>` werden dagegen in Kapitel 13, »Webservices« ab Seite 1043 vorgestellt.
- Die Erstellung eigener HTTP-Handler und die Aktivierung im Abschnitt `<httpHandlers>` beschreibt Abschnitt 12.2, »HTTP-Handler« ab Seite 1028.
- Hinweise zur Konfiguration der Internationalisierung und Globalisierung finden Sie in Abschnitt 4.10.3, »Konfiguration in *web.config*« ab Seite 328. Verantwortlich dafür ist der Abschnitt `<globalization>`.

- Web Services dienen der Maschine-zu-Maschine-Kommunikation. Der Abschnitt `<webServices>` dient der Konfiguration. Erläuterungen dazu finden Sie im Kapitel 13, »Webservices« ab Seite 1043.

Alle anderen grundlegenden Einstellungen finden Sie nachfolgend.

10.3.1 Prinzipieller Umgang mit Konfigurationsdateien

Generell gelten die Einstellungen in der *web.config* für das Verzeichnis, in dem sie liegt. Das ist möglicherweise umständlich, wenn verschiedene Verzeichnisse einer Applikation unterschiedlich konfiguriert werden sollen. Es wäre sicher angenehm, alle Einstellungen in einer Datei zu verwalten.

Normalerweise stehen die applikationsspezifischen Parameter im Zweig `<system.web>`:

```
<configuration>
  <system.web>
    <!-- Hier stehen die Tags -->
  </system.web>
</configuration>
```

Einstellungen in web.config für ein bestimmtes Verzeichnis werden immer automatisch an alle tiefer liegenden vererbt. Durch die Option `<location>` können Sie dieses Verhalten modifizieren.



Diese Einstellungen gelten für den Ort, an dem die Konfigurationsdatei selbst liegt.

Sie können der Datei nun weitere Abschnitte hinzufügen, `<location>` genannt:

```
<configuration>
  <system.web>
    <!-- Hier stehen die Tags -->
  </system.web>
  <location path="data/special">
    <system.web>
      <!-- Hier stehen die Tags -->
    </system.web>
  </location>
</configuration>
```

Ausgehend vom aktuellen Verzeichnis kann mit der Angabe `path` bestimmt werden, für welches Verzeichnis die enthaltenen Konfigurationsangaben gelten.

Die Einstellungen lassen sich natürlich auch für die gesamte Maschine vornehmen. Dies erfolgt dann in der Datei *machine.config*.

10.4 Konfiguration des gesamten Systems machine.config

Die Konfiguration eines Servers übernimmt die Datei *machine.config*. Fast alle Einstellungen, die hier erfolgen, lassen sich in *web.config* wieder überschreiben. Sie finden *machine.config* in folgendem Ordner:

`%systemroot%\Microsoft.NET\Framework\v1.0.3705\CONFIG`

Die Versionsnummer, unterstrichen dargestellt, muss gegebenenfalls an Ihre Installation angepasst werden.

10.4.1 Optionen des Compilers

ASP.NET-Codes werden mit einem Compiler übersetzt. Das passiert, wenn Sie das Framework SDK installiert haben, beim Aufruf der Seite automatisch. Wenn Sie Visual Studio .NET verwenden, kümmert sich die Entwicklungsumgebung um die nötigen Schalter und Einstellungen. Abweichungen lassen sich in den Eigenschaften eines Projekts einstellen.

Bei der automatischen Übersetzung bestehen solche Eingriffsmöglichkeiten über die *web.config*, basierend auf den in *machine.config* befindlichen Standardeinstellungen:

```
<compilation debug="false" explicit="true" defaultLanguage="vb">
  <compilers>
    <compiler language="c#;cs;csharp"
      extension=".cs"
      type="Microsoft.CSharp.CSharpCodeProvider,
        System, Version=1.0.3300.0,
        Culture=neutral,
        PublicKeyToken=b77a5c561934e089"
      warningLevel="1" compilerOptions="/nostdlib"/>
    <compiler language="vb;vbs;visualbasic;vbscript"
      extension=".vb"
```

```

        type="Microsoft.VisualBasic.VBCodeProvider, ↵
            System, Version=1.0.3300.0, ↵
            Culture=neutral, ↵
            PublicKeyToken=b77a5c561934e089"/>
    <compiler language="js;jscript;javascript"
        extension=".js"
        type="Microsoft.JScript.JScriptCodeProvider, ↵
            Microsoft.JScript, ↵
            Version=7.0.3300.0, ↵
            Culture=neutral, ↵
            PublicKeyToken=b03f5f7f11d50a3a"/>

</compilers>
</compilation>

```

Die Auflistung der bereit stehenden Compiler dürfte kaum Fragen aufwerfen. Anbieter von anderen Sprachen für .NET müssen ihre Software so gestalten, dass der Compiler und dessen Basis-klasse hier vermerkt werden.

Interessanter sind die Attribute des Zweiges <compilation>. Hier gibt es folgende Einstellmöglichkeiten:

► debug="true|false"

Legt fest, ob der Compiler `Debuggercode` erzeugen soll oder nicht. Debuggercode führt im Code Marken mit, mit denen die Position der Haltepunkte in der Quelle festgestellt werden kann. Das macht den Code langsamer und größer, ist aber in der Entwicklungsphase notwendig. Die Option kann durch ein Attribut der Seitendirektive @Page überschrieben werden.

► defaultLanguage="cs|vb|js"

Dies ist die Standardsprache, die in ASP.NET-Seiten für eingebetteten Code verwendet werden soll. Auch dieser Wert kann mit der Seitendirektive @Page überschrieben werden. Der Standardwert ist »vb« für Visual Basic .NET.

► tempDirectory

Die temporären Dateien werden normalerweise in folgendem Pfad abgelegt:

`%systemroot%\Microsoft.NET\Framework\v1.0.3705\Temporary ASP.NET files\`

Der unterstrichene Teil muss der installierten Version entsprechen. Temporäre Dateien entstehen, wenn der Compiler ASP.NET-Dateien übersetzt.

► `strict="True|False"`

Schaltet die Strict-Option (`Option Strict`) in VB.NET ein oder aus. Standard ist »ein« (`True`).

► `explicit="True|False"`

Schaltet die Explicit-Option (`Option Explicit`) in VB.NET ein oder aus. Standard ist »ein« (`True`).

► `batch="True|False"`

Kontrolliert die Verfügbarkeit der Übersetzung von Stapelverarbeitungsdateien. Diese Option ist standardmäßig aktiv.

► `batchTimeout="15"`

Wenn mehrere Übersetzungsvorgänge aus einer Stapelverarbeitungsdatei heraus erfolgen, begrenzt diese Option die Anzahl Sekunden, die der Vorgang in Anspruch nehmen darf. Der Standardwert beträgt 15 Sekunden.

► `maxBatchSize="1000"`

Kontrolliert die maximale Zahl an Datenquellen, die in einem Batchlauf verwendet werden können. Der Standardwert beträgt 1.000.

► `maxBatchGeneratedFileSize="3000"`

Bestimmt die maximale Dateigröße, die durch Übersetzung in einem Batchlauf entstehen darf. Der Standardwert beträgt 3.000 KByte.

► `numRecompilesBeforeAppRestart="15"`

Anzahl der Compilerläufe, nach der die Applikation recycelt wird. Dies sichert die Zuverlässigkeit des Compilers.

In einem zweiten Unterelement `<assemblies>` werden außerdem die standardmäßig verfügbaren Assemblies definiert, die der Compiler verwenden darf. Hier sind die Klassen der Basisklassenbibliothek und einiger ergänzender Assemblies zu finden. Weitere, beispielsweise von Drittanbietern, können hinzugefügt werden.

Die Referenzen lassen sich aber auch im Visual Studio .NET hinzufügen, sodass die Option nur für reine SDK-Umgebungen interessant ist.

10.4.2 Den ASP.NET-Worker Process konfigurieren

ASP.NET läuft unabhängig vom IIS und kann deshalb weitgehend konfiguriert werden, auch wenn kein Zugriff auf die Metabasis (IIS 5) oder Konfigurationsdateien (IIS 6) besteht. Die Basis der ASP.NET-Engine bildet der so genannte »Worker Process« (der, der die Arbeit macht). Realisiert ist dieses Programm als *aspnet_wp.exe*. Ob der Worker Process gestartet ist und wie viel Speicher er konsumiert, können Sie im Task-Manager sehen.

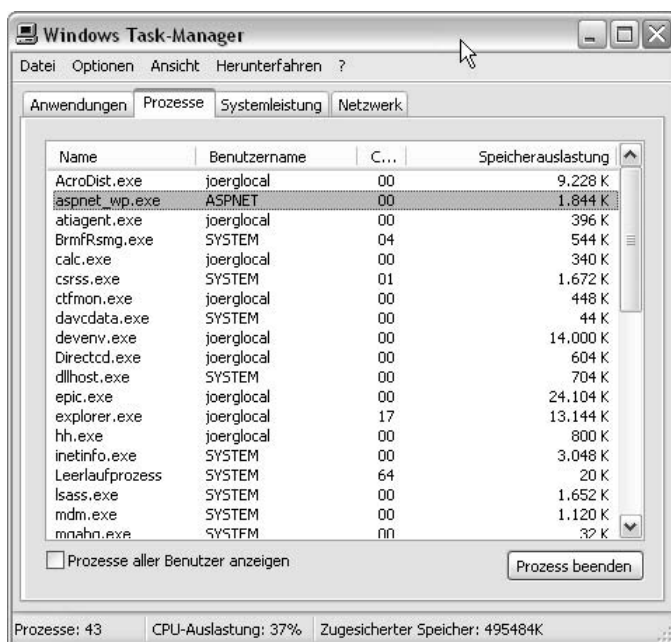


Abbildung 10.2: Der Worker Process im Task-Manager

Läuft der Worker Process, taucht er im Task-Manager auf. Im laufenden Betrieb eines Live-Servers kann so auch überwacht werden, welche Prozessorleistung allein die Verarbeitung der ASP.NET-Dateien in Anspruch nimmt. Es ist nahe liegend, dass hier Konfigurationsmöglichkeiten angebracht sind.

Die Konfigurationseinstellungen des Worker Process betreffen auch den IIS. Sie werden nicht durch ASP.NET überwacht, sondern von *aspnet_isapi.dll* beim Start des IIS gelesen. Nach allen Änderungen an dem nachfolgend beschriebenen Abschnitt `<processModel>` der Konfigurationsdatei *machine.config* muss der IIS neu gestartet werden.



Den Worker Process konfigurieren: <processModel>

Die Konfiguration erfolgt im Tag <processModel>. Die einzelnen Attribute werden nachfolgend vorgestellt.

**enable=
"true|false"**

Grundsätzlich kann der Worker Process als externes Programm ein- und ausgeschaltet werden. »Ein« (`enable="true"`) ist der Standardwert. Wenn Sie die Option deaktivieren, wird der Worker Process nicht extern, sondern im IIS als In-Process-Applikation gestartet. Dies ist zwar schneller, aber auch unsicherer, weil ein Absturz oder Hängen des Worker Process den IIS mitreißt. Um die Änderung wirksam werden zu lassen, muss der WWW-PUBLISHINGDIENST gestoppt und gestartet werden.

**timeout="Infinite-
|HH:MM:SS"**

Das nächste Attribut `timeout` bestimmt die Zeit, die der Worker Process maximal mit der Abarbeitung einer Anforderung beschäftigt sein darf. Der Standardwert ist `Infinite` (unendlich). Wird die Zeit überschritten und läuft das Programm noch immer, startet ein neuer Worker Process. Falls Applikationen laufen, die Sie nicht ändern können und die nach einer gewissen Zeit immer mehr Speicher verbrauchen und damit Systemleistung kosten, können Sie den Zeitraum so einstellen, dass vor Eintritt des Engpasses ein neuer Prozess startet. Folgender Parameter startet den Worker Process alle 24 Stunden neu:

```
timeout="24:00:00"
```

**idleTime=
"Infinite
|HH:MM:SS"**

Der Worker Process startet normalerweise bei der ersten eintreffenden Anforderung und bleibt dann in mindestens einer Instanz bestehen. Der Standardwert für das Attribut `idleTime` beträgt `Infinite`. Wird hier ein Zeitwert eingesetzt, beendet der Worker Process nach Ablauf der Zeit seine Arbeit und startet mit der nächsten Anforderung wieder. Der Neustart entspricht dem »Recyclen« des Prozesses. Auch hier ist die Anwendung gegeben, wenn es Probleme mit steigendem Ressourcenverbrauch gibt.

**shutDownTime-
out="HH:MM:SS"**

Der Worker Process hat normalerweise eine bestimmte Zeit zur Verfügung, um sich selbst zu beenden. Dieser Parameter – Standardwert fünf Sekunden – bestimmt diese Zeit. Möglicherweise laufen sehr anspruchsvolle Anwendungen, die längere Zeit brauchen, um Vorgänge abzuschließen und den Prozess später enden lassen. Dann kann die Zeit entsprechend erhöht werden. Wird der Zeitablauf erreicht, forciert ASP.NET das Ende des Prozesses ohne Rücksicht auf den Zustand laufender ASP.NET-Programme.

Der Neustart des Worker Processes spielt generell eine große Rolle, um hoch belastete Server ständig mit maximaler Leistung betreiben zu können. Vor allem Provider, die viele ihnen unbekannte Applikationen laufen haben, profitieren von dieser Option. So kann mit folgendem Parameter der Neustart nach 500 Anforderungen forciert werden:

```
requestLimit="500"
```

Der Standardwert ist *Infinite*, es wird also normalerweise kein recyceln des Prozesses stattfinden.

Ein anderen Fall tritt auch unter hoher Last ab und zu auf. Wenn zuviele Anforderungen anstehen, »stapelt« ASP.NET diese in einer Warteschlange. Nun kann das häufig passieren und darauf hindeuten, dass der Worker Process nicht mehr zufrieden stellend arbeitet. Standardmäßig dürfen 5000 Anforderungen gestapelt werden. Ist diese Zahl erreicht, wird der Prozess recycelt.

Wenn einige wenige Prozesse zu viel Speicher in Anspruch nehmen, sollte der Abbruch nach dem Speicherverbrauch erfolgen. Als Abbruchkriterium wird hier nicht ein absoluter Wert benötigt, sondern eine Prozentangabe (ohne Prozentzeichen), die den Wert in Abhängigkeit vom physischen Speicher angibt:

```
memoryLimit="50"
```

Normalerweise sollten echte Speicherlecks in verwaltetem (managed) Code nicht auftreten. Wenn jedoch alte COM-Komponenten eingebunden werden, die natürlich nicht verwalteten Code enthalten, sind derartige Fehler möglich.

Neu in ASP.NET ist auch die explizite Unterstützung von Mehrprozessorsystemen. Microsoft bezeichnet dies allgemein als »Web Garden«. Mit Hilfe der Attribute `webGarden` und `cpuMask` wird festgelegt, wie viele Prozessoren verwendet werden.



Der Worker Process läuft normalerweise unter dem Konto ASPNET. Sie können dies in Abbildung 10.2 in der Spalte Benutzername erkennen. Dieses Konto lässt sich ändern, indem die Attribute `userName` und `password` gesetzt werden.

Fehler, die während der Arbeit des Worker Process erkannt werden, erscheinen im Ereignisprotokoll »Anwendung«. Ob nur Fehler oder all Ereignisse aufgezeichnet werden, kann mit dem Attri-

requestLimit=
"Infinite|Zahl"

requestQueue-
Limit="Zahl"

memory-
Limit="Zahl"

userName
= "Name"
password="pw"

logLevel=
"All|None|Errors"

but `logLevel` entschieden werden. Die Möglichen Optionen sind `All`, `None` und `Errors`. `Errors` ist der Standardwert.

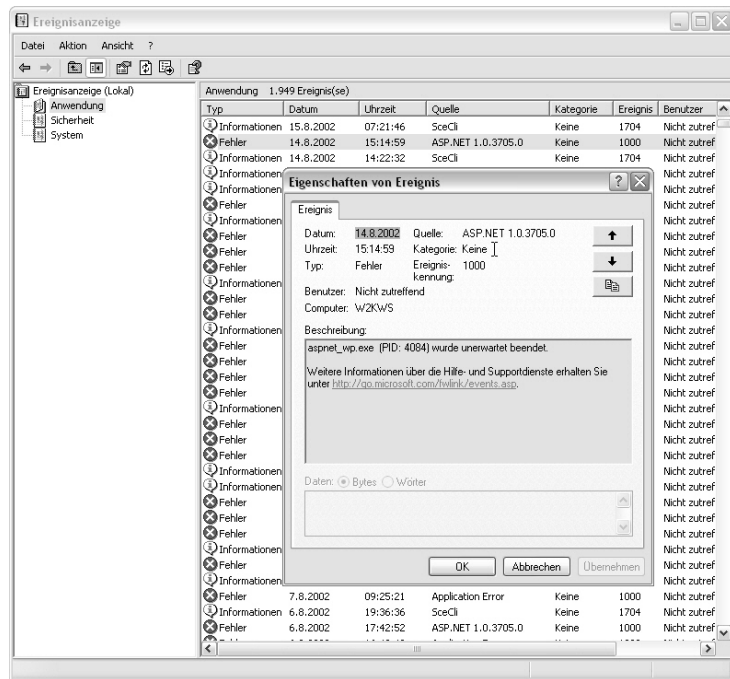


Abbildung 10.3: Fehler des Worker Process im Ereignisprotokoll

`clientConnected-
Check="Infinite-
|HH:MM:SS"`

Mit einem weiteren Attribut kann das Verhalten des Clients überwacht werden. Wenn die Verbindung zwischen Webserver und Browser langsam ist, kann es vorkommen, dass Benutzer die Seite erneut anfordern, obwohl die vorhergehende Anforderung noch nicht beendet wurde. Dann stapeln sich die Anforderungen in der Anforderungswarteschlange. Standardmäßig prüft ASP.NET alle fünf Sekunden, ob neue Anforderungen des Clients vorliegen. Wenn das der Fall ist, wird nur die neueste bearbeitet, sodass die Arbeit des Prozesses entlastet wird, weil sinnlose Anforderungen, auf die ohnehin niemand mehr wartet, unbeantwortet bleiben.

Weitere Attribute sind speziellen Anwendungen vorbehalten und sollen hier nur überblicksweise gezeigt werden:

- Die Sicherheit von COM- und DCOM-Komponenten wird mit `comAuthenticationLevel` und `comImpersonate` geprüft.

- ▶ Wenn der seltene Fall eintritt, dass sich zwei Threads gegenseitig verriegeln (Deadlock), hilft `responseDeadlockInterval`. Dieses Attribut legt fest, wann dies erkannt wird; der Standardwert sind 3 Minuten (»00:03:00«). Mit `responseRestartDeadlockInterval` wird festgelegt, wie lange nach einem Abbruch auf die nächste Prüfung gewartet wird (Standard: 9 Minuten (»00:09:00«)).
- ▶ Zur Kontrolle der Threads des Worker Process dienen `maxWorkerThreads` (maximale Anzahl, Standardwert 25) und `maxIoThreads` (Ein-/Ausgabe-Threads, Standardwert ebenfalls 25).

10.5 Sicherheit

Dieser Abschnitt behandelt Sicherheit aus dem speziellen Blickwinkel der Web-Applikation. Eine mehr allgemeine gehaltene Einführung in die Sicherheitsprinzipien des Frameworks finden Sie am Anfang des Buches im Abschnitt 1.6, »Allgemeine Sicherheit im Framework« ab Seite 83.

10.5.1 Das Sicherheitskonzept des IIS

ASP.NET nutzt den IIS als Schnittstelle zum Client. Der IIS mischt sich zwar nicht mehr in die eigentliche Verarbeitung von Seiten ein, ist aber als unterliegende Instanz Teil des Sicherheitskonzeptes. Er verfügt über einige Sicherheitskonzepte, die für die Arbeit mit ASP.NET von Interesse sind:

- ▶ **Verschlüsselung der Kommunikation mit SSL (Secure Socket Layer)**
Hiermit wird der Übertragungsweg geschützt. SSL verhindert wirkungsvoll, dass Dritte übertragene Daten lesen oder verändern können.
- ▶ **Beschränkung des Zugriffes auf bestimmte IP-Adressen oder Domänen**
Durch die generelle Beschränkung des Zugriffes kann der Zugriff von außen geregelt werden, sodass die Kommunikation mit unerwünschten Partnern gar nicht erst stattfindet. Dieses Verfahren ist für öffentliche Webserver kaum einsetzbar. Wenn Sie einen Webserver im Intranet installieren, ist der Einsatz sinnvoll.

► **Bereitstellung virtueller Verzeichnisse, die das Ausspähen der physischen Verzeichnisstruktur verhindern**

Generell stellt der IIS ein virtuelles Verzeichnis für Ihre Applikation bereit. Wenn Sie Projekte mit Visual Studio .NET entwickeln, werden diese auch in einem solchen Verzeichnis installiert.

► **Prüfung von Zugriffsberechtigungen auf Basis der NTFS-Sicherheit**

Über den IIS lassen sich mehrere Zugriffsmethoden einrichten, die eine Steuerung der Nutzung der Applikation auf Benutzerebenen erlauben. Damit haben Sie die genaueste Kontrolle, aber auch den höchsten Aufwand.

Wenn Sie ein Sicherheitskonzept für Ihre Site erarbeiten, können Sie diese vier Methoden einsetzen, um sich wirkungsvoll gegen Angriffe zu schützen.

Formen der Benutzerauthentifizierung

Der IIS lässt bestimmte Formen der Benutzerauthentifizierung zu, die je nach Einsatz und Sicherheitsbedarf verwendet werden können:

► **Anonymer Zugriff**

Diese Form erlaubt den Zugriff ohne weitere Einschränkungen. Der IIS gestattet dem Client die Nutzung des Systemkontos *IUSR_MachineName*. Das gilt aber nur bis zum Aufruf des ASP.NET-Worker Process. Alle .NET-Instanzen werden unter dem ASP.NET-Konto *ASPNET* ausgeführt.

► **Authentifizierung mit Basic Authentication**

Der Benutzer muss sich hier mit Benutzernamen und Kennwort identifizieren. Beide Angaben werden im Klartext übertragen – abgesehen von der üblichen Base64-Kodierung für binäre Inhalte. Das Verfahren muss verwendet werden, wenn der geschützte Zugriff beliebigen Clientprogrammen wie Netscape oder Opera gestattet werden soll.

► **Authentifizierung mit Digest Authentication**

Dieses Verfahren erfordert auch die Angaben von Benutzernamen und Kennwort. Es wird aber nur ein Hashwert (MD5-verschlüsselt) übertragen, sodass ein Abfangen der Kennwortdaten nicht ausreicht, um daraus die Daten zu re-

konstruieren. Auf Serverseite ist der Zugriff auf Active Directory erforderlich, als Client muss der Internet Explorer eingesetzt werden.

► **Integrierte Windows-Authentifizierung**

Dieses Verfahren nutzt die interne Anmeldeprozedur von Windows und ist damit auf Windows-Clients beschränkt. Auch hier wird ein Hashwert verwendet, sodass der Übertragungsweg sicher ist. Hashwerte sind hier Prüfsummen über Daten.

► **Sicherung über Clientzertifikate**

Wenn bestimmte Benutzer sicher identifiziert werden sollen, können Clientzertifikate verwendet werden. ASP.NET stellt den Zugriff darauf über entsprechende Klassen bereit. Allerdings ist der Aufwand auf Benutzerseite sehr hoch, denn neben der kostenpflichtigen Beschaffung des Zertifikats ist auch die Installation im Browser erforderlich.

10.5.2 ASP.NET-Sicherheitsfunktionen im IIS

Für die Feststellung, welche Rechte ein konkreter Prozess im Rahmen einer Anforderung von Ressourcen nun hat, sind einige weitere Überlegungen notwendig. Die erste Instanz, mit der der Browser Kontakt aufnimmt, ist der IIS. Dieser läuft – als Programm *inetinfo.exe* – unter dem Konto *SYSTEM*. Dieses Konto hat natürlich sehr weit reichende Privilegien, sodass weitere Sicherheitsebenen wie Schalen darüber liegen. Die erste Ebene ist die so genannte Impersonifizierung des Prozesses. Der Benutzer hat sich immer in der einen oder anderen Weise identifiziert – dazu gehört auch die sichere Erkennung des Wunsches, anonym zu arbeiten. In diesem Fall wird das Konto *IUSR_MachineName* verwendet. Erfolgt ein Zugriff auf Komponenten (COM-Objekte), wird als Zugriffskonto *IWAM_MachineName* (WAM = Windows Application Management) verwendet. Die ASP.NET-Laufzeitumgebung ist davon ausgeschlossen, sie läuft immer unter dem Konto *ASPNET*, das bei der Installation eingerichtet wurde.

Die Authentifizierung in ASP.NET

ASP.NET bietet eine gute Unterstützung für verschiedene Authentifizierungsarten an. Sie können je nach Bedarf entscheiden, welches System zum Einsatz kommt:

► **Windows-Authentifizierung**

Siehe Abschnitt 10.5.5, »Windows-Authentifizierung« ab Seite 985.

► **Forms-Authentifizierung**

Siehe Abschnitt 10.5.3, »Forms-Authentifizierung« ab Seite 977.

► **Passport**

Bevor Sie sich daran machen, die Anmeldung über Passport zu realisieren, sollten Sie sich mit dem Konzept auseinandersetzen. Passport ist ein Dienst der Microsoft .NET My Services-Palette und bietet Benutzern die Möglichkeit eines zentralen Login für alle teilnehmenden Webseiten. Das ist für Benutzer bequem und kostenlos. Für Unternehmen sieht das freilich anders aus. Sie müssen sich beim Passport-Dienst anmelden und sich den Schlüssel und Zugangscodes beschaffen, um daran teilnehmen zu können. Dies ist in der Praxis eine längere Prozedur. Nach der Anmeldung müssen Sie eine »Compliance«-Erklärung absenden und einem Tester bei Microsoft Zugang zu Ihrer Applikation verschaffen. Nach erfolgreichem Test wird Ihr Konto freigegeben und Sie nehmen nun mit Ihrer Site an Passport teil.

Das Ganze macht Microsoft nicht aus Nächstenliebe. Da der Dienst für Benutzer kostenlos ist, stellt der Unternehmer die Einnahmequelle dar. Zu bezahlen sind derzeit einmalig \$ 10.000 pro Unternehmen, auch wenn Sie mehrere Domänen anschließen. Weiterhin fallen \$ 1.500 für jeden Compliance-Test an, also am Anfang mindestens ein Mal. Danach behält sich Microsoft weitere Tests zu je \$ 1.500 vor, wobei man mit einem jährlichen Rhythmus rechnen kann.

Sie sollten sich deshalb genau überlegen, ob der Aufwand für Passport lohnt. Für eine Website mit hoher An- und Abmeldehäufigkeit, starkem Endkundenverkehr und möglicherweise weiteren kostenpflichtigen Diensten kann der Einsatz lohnenswert sein. Die Autoren haben Erfahrung in der Realisierung solcher Dienste und stehen interessierten Lesern hier gern im Rahmen einer Consultingleistung zur Verfügung.

**Der anonyme
Zugriff**

Daneben besteht natürlich immer die Möglichkeit, auf eine explizite Authentifizierung zu verzichten. Das ist im Übrigen auch die Standardeinstellung: Der anonyme Zugriff. Der Prozess läuft in diesem Fall unter dem Konto *ASPNET* ab, solange die Impersoni-

fizierung deaktiviert bleibt¹. Wenn die Impersonifizierung aktiviert wurde, übernimmt ASP.NET das Konto des IIS – im Falle des anonymen Zugriffs ist dies *IUSR_MachineName*.

Um nun die gewünschte Zugriffsart festzulegen, nutzen Sie den Abschnitt `<authentication>` der Datei *web.config*.

```
<configuration>
  <system.web>
    <authentication mode="Windows" />
    ...
  </system.web>
</configuration>
```

Für das Attribut `mode` können Sie folgende Werte wählen: *Windows*, *Passport*, *Forms*, *None*. Innerhalb des Zweiges `<system.web>` können dann feinere Einstellungen vorgenommen werden, die in den entsprechenden Abschnitten weiter unten beschrieben werden.

Die Impersonifizierung in ASP.NET

Wenn sich nun aber ein Benutzer nicht anonym anmeldet, muss die Laufzeitumgebung in der Lage sein, ihm die Rechte einzuräumen, die er erwartet und die seinem Konto entsprechen. Dieser Vorgang wird als »Impersonifizierung« bezeichnet. Das heißt, der aktuelle Prozess nimmt die »Person« des Aufrufers an, um mit seinen Rechten operieren zu können.

Standardmäßig ist die Impersonifizierung abgeschaltet. Das heißt, dass trotz der Anmeldung über den IIS einem Benutzer nicht mehr Rechte eingeräumt werden, als das Konto *ASPNET* hergibt. Sie können diesen Vorgang explizit für den gesamten Server freigeben. Dazu öffnen Sie die Datei *machine.config* und suchen den Eintrag `<identity>`.

10.5.3 Forms-Authentifizierung

Die Forms-Authentifizierung unterstützt die Entwicklung einer eigenen Anmeldeprozedur basierend auf HTML-Formularen und Cookies. Das Sicherheitsniveau ist relativ niedrig, außerdem muss der Benutzer dauerhaft Cookies akzeptieren. Dafür ist der Einrichtungsaufwand gering. Die Datei *web.config* konfigurieren Sie für diese Funktion folgendermaßen:

1 Der deaktivierte Zustand entspricht der Standardeinstellung.

```

<authentication mode="Forms" >
  <forms
    name="LogOnCookie"
    path="/"
    loginUrl="login.aspx"
    protection="All"
    timeout="25">
    <credentials passwordFormat="Clear">
      <user name="Joerg" password="haide" />
      <user name="Uwe" password="lukas" />
    </credentials>
  </forms>
</authentication>
<authorization>
  <deny users="?" />
</authorization>

```

Eine solche Konfiguration führt mehrere Schritte bei der Anforderung einer Ressource im betroffenen Verzeichnis aus. Zuerst wird geprüft, ob für den betreffenden Benutzer der Zugriff erlaubt oder verboten ist. Im Beispiel wird allen anonymen Benutzern der Zugriff untersagt:

```
<deny users="?" />
```



Sie können im Attribut `users` neben einer durch Kommata getrennten Liste von Namen auch ein Sternchen »« für »Alle« oder ein Fragezeichen »?« für »Anonym« angeben. Wenn Sie Domännennamen kontrollieren wollen, schreiben Sie »DOMAIN\Name«.*

Erst wenn ein solches Verbot greift, wird die Authentifizierungsprozedur gestartet. Erlauben Sie den Zugriff dagegen, wird auch keine Anmeldung angefordert.

Im Fall der Form-Authentifizierung wird ASP.NET nun Folgendes versuchen: Es wird ein Cookie erzeugt, basierend auf den Attributen des `<forms>`-Abschnittes. Dann wird die Ausführung an eine Anmeldeseite weitergeleitet, die im Beispiel den Namen `login.aspx` hat. Dies ist auch der Standardname, den ASP.NET verwendet, wenn Sie keinen angeben. Dort sollten Sie ein Formular programmieren, das die benötigten Daten erfasst. Anschließend vergleichen Sie mit einem entsprechenden Programm, ob die Anmeldedaten Ihren Erwartungen entsprechen. Dann geben Sie die Anwendung frei. ASP.NET kümmert sich nun während der Sitzung um die laufende Identifizierung des Benutzers, basierend auf dem Cookie.

Beachten Sie, dass für die Attributnamen Groß- und Kleinschreibung entscheidend ist.



Die möglichen Einstellungen sind sehr variabel. Die folgende Tabelle zeigt die Bedeutung der Attribute des `<forms>`-Elements:

Attribut	Bedeutung und Optionen
<code>name</code>	Name des Cookies
<code>loginUrl</code>	URL, unter der das Anmeldeformular zu finden ist
<code>protection</code>	Verschlüsselungsart des Cookies: ▶ All – Cookie wird verschlüsselt und Daten werden überprüft. Dies ist die Standardeinstellung ▶ None – Keine Prüfung und Verschlüsselung ▶ Encryption – nur verschlüsseln ▶ Validation – Nur überprüfen
<code>timeout</code>	Laufzeit des Cookies in Minuten. Der Standardwert ist 30.
<code>path</code>	Pfad zur Anwendung, der dem Cookie zugewiesen wird. Der Standard ist <code>»»</code> (Wurzel der Applikation)

Tabelle 10.4: Attribute für `<forms>`

Innerhalb des `<forms>`-Zweiges werden nun die speziellen Zugriffsbedingungen innerhalb des Zweiges `<credentials>` aufgeführt. Hierfür finden Sie nachfolgend das verfügbare Attribut:

Attribut	Bedeutung und Optionen
<code>passwordFormat</code>	Zeigt an, wie das Kennwort verschlüsselt ist: ▶ Clear – Keine Verschlüsselung ▶ MD5 – MD5-Hash ▶ SHA1 – SHA1-Hash Die Verschlüsselung zeigt an, dass der in der Datei befindliche Wert bereits in dieser Form kodiert ist.

Tabelle 10.5: Attribute für `<credentials>`

Bleibt zuletzt die Festlegung der Benutzer, für die die Authentifizierung gilt, mit dem Tag `<user>`. Zwei Attribute sind hier zu finden:

Attribut	Bedeutung und Optionen
name	Der Anmeldename
password	Das erwartete Kennwort

Tabelle 10.6: Attribute für <user>



Die Nennung des Kennworts im Klartext in web.config ist eine Sicherheitslücke, auch wenn ASP.NET den Zugriff auf diese Datei über das Web – egal in welchem Sicherheitskontext – niemals zulässt. Denn lokal besteht natürlich Zugriff darauf. Speichern Sie die Kennwörter daher nach Möglichkeit in der angegebenen Hash-Kodierung (MD5 oder SHA1).

Programmierung einer Authentifizierungslösung

Die typische Programmierung einer Authentifizierungslösung besteht aus folgenden Schritten:

1. Festlegen der Authentifizierungsart und Vorbereiten der »web.config«
2. Erstellen eines Eingabefelds
3. Überprüfen der Eingabewerte und Reaktion darauf



Das folgende Beispiel verwendet die zuvor vorgestellte Einstellung der Datei »web.config«, wobei es auf den Abschnitt <authentication> ankommt. Nutzen Sie dann Zugriffe von verschiedenen Computern, um anonyme und authentifizierte Anmeldeversuche testen zu können.

Die Reaktion auf die Eingabe kann wiederum auf zwei Aktionen hinauslaufen: Freigabe der ursprünglich angeforderten Datei oder Ausgabe eines Hinweises über die misslungene Anmeldung. Beides ist sehr einfach zu programmieren. Das Formular login.aspx könnte folgendermaßen aussehen:

```
<h1>Anmeldung erforderlich</h1>
<form runat="server">
<table>
  <tr>
    <td>Name: </td>
    <td>
      <asp:TextBox Runat="server" ID="logName" />
    </td>
  </tr>
</table>
```



```

<tr>
  <td>Kennwort: </td>
  <td>
    <asp:TextBox TextMode="Password"
      Runat="server" ID="logPassword" />
  </td>
</tr>
<tr>
  <td></td>
  <td>
    <asp:Button Runat="server" ID="Submit"
      Text="Anmelden"
      OnClick="Login_Click"/>
  </td>
</tr>
</table>
<asp:Label Runat="server" ID="LoginError"
  BackColor="Red" ForeColor="White"/>
</form>

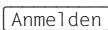
```

Listing 10.1: Ein Anmeldeformular (login.aspx)

Wenn nun eine geschützte Datei angefordert wird, leitet ASP.NET zu diesem Formular um. Sie erkennen dies durch die Veränderung des URL:

Ablauf der Authentifizierung

Adresse  <http://localhost/dasbuch/csharpsecurity/login.aspx?ReturnUrl=%2fdasbuch%2fcsharpsecurity%2fSecurityCheck.aspx>

Dort erhält der Benutzer die Möglichkeit, die Daten einzugeben. Wird auf die Schaltfläche  geklickt, wird die entsprechende Ereignisbehandlungsmethode ausgeführt.

Anmeldung erforderlich

Name:

Kennwort:

Fehler: Anmeldedaten falsch. Beachten Sie Groß- und Kleinschreibung

Abbildung 10.4: Das Anmeldeformular nach einer fehlerhaften Anmeldung

Der Aufwand ist offensichtlich überschaubar, wie das folgende Listing zeigt:

```
Imports System.Web.Security

Public Class login
    Inherits System.Web.UI.Page
    Protected logName As TextBox
    Protected logPassword As TextBox
    Protected WithEvents Submit Button
    Protected LoginError As Label

    Public Sub Login_Click(ByVal sender As Object,
                          ByVal e As EventArgs)
        If FormsAuthentication.Authenticate(logName.Text,
                                           logPassword.Text) Then
            FormsAuthentication.RedirectFromLoginPage(logName.Text,
            True)
        Else
            LoginError.Text = "Fehler: Anmeldedaten falsch.
                              Beachten Sie Groß- und Kleinschreibung"
        End If
    End Sub
End Class
```

Listing 10.2: Auswertung der Eingabe und Prüfen der Anmeldedaten (login.aspx.vb)

Wie es funktioniert

Um Zugriff auf die Klassen zur Auswertung der Anmeldeinformationen zu erlangen, müssen Sie noch den Namensraum `System.Web.Security` einbinden:

```
Imports System.Web.Security
```

Im eigentlichen Programm prüfen Sie nun, ob Name und Kennwort denen in der `web.config` entsprechen:

```
If (FormsAuthentication.Authenticate(logName.Text,
logPassword.Text)) Then
```

Wenn das der Fall ist, lösen Sie eine Weiterleitung auf die ursprünglich aufgerufene Seite aus:

```
FormsAuthentication.RedirectFromLoginPage(logName.Text, True)
```

Tipp: permanente Cookies

Diese statische Methode bietet als zweiten Parameter die Möglichkeit, das mit der Weiterleitung gesendete Authentifizierungs-Cookie permanent zu speichern. Diese Wahl können Sie – im Sinne einer guten Benutzerführung – auch dem Benutzer überlassen. Dazu könnten Sie ein `CheckBox`-Steuerelement auf der Anmeldeseite

te platzieren. Wird es aktiviert, wird der Parameter gleich `True` gesetzt. Dann wird das Ablaufdatum nicht auf die in der `web.config` definierten Minuten (im Beispiel waren es 25) gesetzt, sondern so weit in die Zukunft, dass es praktisch permanent wird.

Es ist schlechter Stil, permanente Cookies ohne Einverständnis des Benutzers zu speichern. Mit der offenen Bekundung der Verwendung eines Cookies und der Wahlmöglichkeit durch den Benutzer steigt die Nutzungsrate deutlich.



Der *Else*-Zweig behandelt nun noch den Fall einer fehlerhaften Anmeldung.

Weitere Methoden der Klasse `FormsAuthentication`

Verwendet wurden bereits die Methode `Authenticate` (Prüfung der Anmeldeparameter) und `RedirectFromLoginPage` (Setzen des Cookies und Ausführen der Weiterleitung). Sie können diesen Prozess auch feiner steuern, wenn Sie weitere Methoden kennen.

Ebenfalls statisch ist `GetRedirectUrl`. Beim Aufruf wird der URL zurückgegeben, der zur Weiterleitung verwendet wird. Sie können den Wert nicht ändern. Allerdings ist es natürlich möglich, eine eigene Weiterleitung mit `Response.Redirect` (extern) oder `Server.Transfer` (intern) auszuführen und auf die Dienste von `RedirectFromLoginPage` zu verzichten.

`GetRedirectUrl`

Allerdings müssten Sie dann auch das Cookie selbst setzen. Dazu kommt die statische Methode `GetAuthCookie` in Frage. Hiermit wird ein Cookie-Objekt vom Typ `HttpCookie` erzeugt. Als Parameter ist der Anmelde-name des Benutzers und das Ablaufverhalten anzugeben. Das Ablaufverhalten wird als zweiter Parameter übergeben – `True` für ein permanentes Cookie, `False` für eines mit der definierten Laufzeit. Sie können das Cookie nun verändern und dann mit `SetAuthCookie` wieder setzen, sodass es von `RedirectFromLoginPage` verarbeitet werden kann.

Mehr allgemeine Informationen zu Cookies finden Sie in Abschnitt 8.5, »Cookies« ab Seite 791.

Mehr zu Cookies

Bleibt als letzte Maßnahme einer vernünftigen Benutzeranmeldung noch eine Abmeldemöglichkeit. mit Hilfe der Methode `SignOut`, die keine Parameter verlangt, wird das Cookie zerstört und damit beim der nächsten Weiterleitung vom Browser ent-

**`SignOut`:
Abmelden**


```
Imports System.Security
Imports System.Security.Principal

Public Class Security
    Inherits System.Web.UI.Page
    Protected Footer As Label

    Public Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim user As IIdentity = HttpContext.Current.User.Identity

        If (user.IsAuthenticated) Then
            Footer.Text = "Willkommen, " + user.Name
        Else
            Footer.Text = "Oops. Noch unbekannt?"
        End If
    End Sub
End Class
```

*Listing 10.3: Zugriff auf den authentifizierten Benutzer
(Ausschnitt aus SecurityCheck.aspx.vb)*

Mit `IsAuthenticated` wird hier sicherheitshalber festgestellt, ob die Authentifizierung bereits erfolgt ist. Wenn Sie das letzte Beispiel mit der Forms-Authentifizierung aktiviert und verwendet haben, sollte das der Fall sein. Andernfalls erscheint die Fehlermeldung. Über die Eigenschaft `Name` können Sie auf den Anmeldenamen zugreifen. Es ist damit ein leichtes, persönliche Informationen aus einer Datenbank zu holen und die Anzeige der Seite dementsprechend zu modifizieren.

Wie es funktioniert

Statt mit Formularen können Sie Benutzer auch über ein reguläres Konto in der lokalen Benutzerverwaltung oder im Active Directory identifizieren. Dies dürfte jedoch nur im Intranet praktikabel sein. Lesen Sie im nächsten Abschnitt, wie es funktioniert.

10.5.5 Windows-Authentifizierung

Die Windows-Authentifizierung basiert auf der Prüfung der Anmeldung mit Hilfe der NT-Benutzerverwaltung bzw. des Active Directory. Zusätzlich zur Aktivierung der Windows-Authentifizierung selbst muss die Impersonifizierung aktiviert werden, denn nur so gelangt der Anmeldedatenstrom über den IIS bis zu ASP.NET vor.

Für den Benutzer gilt, dass bei dieser Form der Authentifizierung das Standarddialogfeld des Browsers verwendet wird (Abbildung 10.6). Gestalterische Freiheiten wie bei der Forms-Authentifizierung gibt es hier nicht.

Die Ressourcen müssen Sie, wenn Sie Windows die Kontrolle überlassen, natürlich auch über NTFS schützen. Dazu rufen Sie für den betreffenden Dateien und Verzeichnisse den Eigenschaften-Dialog auf und tragen dort auf der Registerkarte SICHERHEIT die Benutzer oder Gruppen ein, denen der Zugriff gewährt werden soll.

Die Windows-Authentifizierung einrichten

Zur Aktivierung modifizieren Sie die Datei *web.config* folgendermaßen:

```
<authentication mode="Windows" />
<identity impersonate="true" />
<authorization>
  <deny users="?" />
</authorization>
```

Dies verhindert den anonymen Zugriff und stellt zugleich sicher, dass das Programm künftig unter dem Konto des Anmelders ausgeführt wird. Wenn der Benutzer nun die Seite abrufen, erscheint folgender Dialog im Browser:



Abbildung 10.6: Der Anmeldedialog im Browser

War die Anmeldung erfolgreich, können Sie ähnlich wie bei der Form-Authentifizierung auf die Anmeldeinformationen zugreifen.

Die Anmeldeinformationen auswerten

Die Verwaltung von Windows-Benutzerkonten obliegt einigen Klassen im Namensraum `System.Security.Principal`, den Sie folgendermaßen einbinden:

Prinzipale

```
Imports System.Security.Principal
```

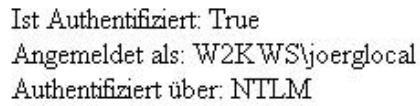
Den aktuell angemeldeten Benutzer können Sie wie schon bei der Form-Authentifizierung dem Objekt `User` entnehmen. Das folgende Beispiel zeigt das Prinzip:

```
Imports System.Text
Imports System.Security.Principal

Public Class WinPrincipal
    Inherits System.Web.UI.Page
    Protected Info As Label
    Private Sub Page_Load(ByVal sender As Object, _
                          ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim wp As IIdentity
        wp = HttpContext.Current.User.Identity
        Dim sb As StringBuilder = New StringBuilder()
        sb.AppendFormat("Ist Authentifiziert: {0}<br/>", _
                        wp.IsAuthenticated.ToString())
        sb.AppendFormat("Angemeldet als: {0}<br/>", _
                        wp.Name)
        sb.AppendFormat("Authentifiziert über: {0}<br/>", _
                        wp.AuthenticationType)
        Info.Text = sb.ToString()
    End Sub
End Class
```

Listing 10.4: Ermittlung von Informationen über den aktuell angemeldeten Benutzer (WinPrincipal.aspx.vb)

Die Ausgabe hängt nun von den Einstellungen des IIS, der Sicherheitsbedingungen des angeforderten Objekts und dem verwendeten Namen ab. Die Arbeitsstation des Autors hieß zu diesem Zeitpunkt »W2KWS«:

A screenshot of a user authentication status window. It contains three lines of text: 'Ist Authentifiziert: True', 'Angemeldet als: W2KWS\joerglocal', and 'Authentifiziert über: NTLM'. The text is in a monospaced font and is enclosed in a rectangular box with a thin border.

```
Ist Authentifiziert: True  
Angemeldet als: W2KWS\joerglocal  
Authentifiziert über: NTLM
```

Abbildung 10.7: Daten eines authentifizierten Benutzers

Es obliegt nun wieder den weiteren Erfordernissen, die gewonnenen Daten zweckentsprechend zu verwenden. Vor allem Kombinationen mit Datenbanken führen schnell zu effizienten und leistungsfähigen Benutzerverwaltungen, die auch mit einer größeren Benutzerzahl gut zurecht kommen.

II Besondere .NET-Klassen

Dem ASP.NET-Programmierer steht der gesamte Funktionsumfang des .NET-Frameworks zur Verfügung. Dieses Kapitel gibt ein paar Anregungen, welche Möglichkeiten es außerhalb der standardmäßig zu ASP.NET gehörenden Funktionen noch gibt.

II.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

II.1.1 Über dieses Kapitel

Mit GDI+ steht eine leistungsfähige Grafikbibliothek zur Verfügung. Auch wenn sie ursprünglich aus der klassischen Windows-Programmierung kommt – GDI steht für Graphics Device Interface – bieten sich dem ASP.NET-Entwickler damit neue Perspektiven. Denn nun können Bilder dynamisch erstellt und gesendet werden, was sich in anderen Sprachen bereits großer Beliebtheit erfreut. Lesen Sie mehr dazu im Abschnitt 11.2, »Bilder dynamisch erstellen« ab Seite 990.

Ebenfalls von großer Bedeutung ist das Versenden von E-Mail direkt über einen SMTP-Server. Auch hier bietet .NET ausreichende Unterstützung. Egal ob es sich um einfache Text-E-Mail oder aufwändiges HTML handelt, alles kann versendet werden. Abschnitt 11.3, »Versenden von E-Mail via SMTP« ab Seite 1006 geht darauf ein.

Zu den Aufgaben des Webprogrammierers gehört auch, Daten von externen Datenquellen zu beschaffen. Manchmal bietet es sich

an, hier direkt andere Webserver abzufragen und quasi programmtechnisch eine HTTP-Anforderung auszulösen. Auch dies ist in .NET einfach zu programmieren. Abschnitt 11.4, »Zugriff auf andere Server« ab Seite 1019 zeigt, wie es geht.

11.2 Bilder dynamisch erstellen

Bildfunktionen sind sehr beliebte und leistungsfähige Funktionen. Statt viele Bilder im Voraus mit aufwändigen Werkzeugen zu erzeugen, werden diese bei Bedarf in ASP.NET generiert.

11.2.1 Prinzip der Bilderzeugung

Wenn man sich den Vorgang der Übertragung einer Webseite vor Augen hält, erkennt man, dass alle Elemente der Seite auf einzelnen Anforderungen beruhen. Erst wird die aufgerufene Stammsite übertragen, dann wertet der Browser alle enthaltenen Elemente aus und ruft diese einzeln ab. Solange es Bandbreite und Systemleistung zulassen, erfolgt dieser Abruf parallel.

Durch dieses Verfahren kann das Ziel eines Bildaufrufes natürlich von der Adresse der ursprünglichen Seite abweichen. Insbesondere ist die Angabe der Bildquelle primär nicht an eine Bilddatei gebunden, sondern es kann ebenso auch ein ASP.NET-Programm sein. In HTML sieht ein Aufruf dann folgendermaßen aus:

```

```

Das Programm *ladebilder.aspx* ist dann dafür verantwortlich, dem Browser die erwarteten Daten zu liefern. Eine einfache Form wäre der Zugriff auf bereits vorhandene Bilddaten und die Auswahl und Versendung per Programm. Wenn Sie dem Programm Parameter übergeben, kann damit schon eine gewisse Dynamik erzielt werden. Als Aufruf käme folgende Version in Frage:

```

```

Aufbau des Programms zum Bildversand

Wenn ein ASP.NET-Programm Bilder versendet, müssen Sie daran denken, dass der Browser zwei Dinge erwartet:

1. Die Angabe des korrekten Headers im Kopf der HTTP-Nachricht:

Content-type: image/png

Die Angabe des Bildtypes hängt vom erzeugten Format ab.

2. Die binären Bilddaten ohne jedes weitere Zeichen im Körper der HTTP-Nachricht

Am einfachsten lässt sich dies erreichen, indem das entsprechende Attribut der Seitendirektive verwendet wird:

```
<% @Page ContentType="image/png" %>
```

Außerdem müssen Sie sicher stellen, dass wirklich nur Bilddaten übertragen werden – HTML oder auch nur ein übrig gebliebener Zeilenumbruch müssen unbedingt vermieden werden.

ContentType

Dynamische Bildauswahl

Als erstes Beispiel soll eine Gruppe von Bildern dynamisch ausgewählt werden. Ideal ist dies für einen grafischen Hitzähler, der mit vorgefertigten Bilddateien arbeitet. Für jede Ziffer wird eine Bilddatei abgelegt; die Namen lauten *0.gif*, *1.gif* usw.

Der Zähler selbst wird hier nur simuliert – mit einem Zufallsgenerator:

```
<html lang="de">
  <head>
    <title>Dynamische Bilder</title>
  </head>
  <body>
    <h1>Dynamische Bilder</h1>
    <div id="zaehler" runat="server">
      <asp:image id="d1" runat="server"/>
      <asp:image id="d2" runat="server"/>
      <asp:image id="d3" runat="server"/>
      <asp:image id="d4" runat="server"/>
      <asp:image id="d5" runat="server"/>
    </div>
  </body>
</html>
```

Listing 11.1: Anzeige eines grafischen Zählers (PixSelect.aspx)

```

Private Sub Page_Load(ByVal sender As System.Object,
                      ByVal e As System.EventArgs) _
Handles MyBase.Load
    Dim rnd As Random = New Random()
    Dim zahl As String = rnd.Next(10000, 99999).ToString()
    d1.ImageUrl = "PixSelectLade.aspx?nummer="+zahl.Substring(0, 1)
    d2.ImageUrl = "PixSelectLade.aspx?nummer="+zahl.Substring(1, 1)
    d3.ImageUrl = "PixSelectLade.aspx?nummer="+zahl.Substring(2, 1)
    d4.ImageUrl = "PixSelectLade.aspx?nummer="+zahl.Substring(3, 1)
    d5.ImageUrl = "PixSelectLade.aspx?nummer="+zahl.Substring(4, 1)
End Sub

```

Listing 11.2: Code-Datei für den grafischen Zähler (PixSelect.aspx.vb)

Wie es funktioniert

Die Bilder werden mit Web Server-Steuerelementen vom Typ Image angezeigt. Nach dem Laden der Seite wird zuerst eine Zufallszahl ermittelt, im Bereich von 10.000 bis 99.999. Damit später ziffernweise zugegriffen werden kann, wird die Zahl gleich in eine Zeichenkette umgewandelt:

```
Dim zahl As String = rnd.Next(10000, 99999).ToString()
```

Dann wird den fünf Steuerelementen jeweils ein URL zugewiesen. Als GET-Parameter *nummer* wird die passenden Ziffer übergeben:

```
d1.ImageUrl = "ladebild.aspx?nummer=" + zahl(0)
```

Die eigentliche Darstellung der Bilder erfolgt in *ladebild.aspx*. Im HTML-Code erfolgt der Aufruf folgendermaßen:

```

```

Die ganze Arbeit wird dann in folgendem Programm erledigt:

```

<%@ Page language="vb" ContentType="image/gif" %>
<% @Import namespace="System.IO" %>
<script runat="server" language="vb">
Public Sub Page_Load()
    Dim bildname As String = Server.MapPath("img/"
                                + Request.QueryString("nummer") + ".gif")

    Try
        Dim fs As FileStream = New
FileStream(bildname, FileMode.Open)
        Dim groesse As Integer = CType(fs.Length, Integer)
        Dim puffer() As Byte = New Byte(groesse) {}
        fs.Read(puffer, 0, groesse)
        fs.Close()
        Response.ClearContent()
        Response.BinaryWrite(puffer)
    End Try
End Sub

```

```

    Catch e As FileNotFoundException
        Response.ClearContent()
        Response.ContentType = "text/html"
        Response.Write ("Fehler: " + e.Message)
    End Try
    Response.End()
End Sub
</script>

```

Listing 11.3: Ausgabe der Bilddaten (PixSelectLade.aspx)

Das Programm beginnt mit der Festlegung des MIME-Typs der Übertragung. Dazu wird die Seitendirektive @Page verwendet:

Wie es funktioniert

```
<%@Page language="vb" ContentType="image/gif" %>
```

Da die Bilddaten von der Festplatte gelesen werden müssen, muss der Namensraum System.IO eingebunden werden:

```
<%@Import namespace="System.IO" %>
```

Innerhalb der Methode Page_Load wird zuerst der Pfad zur passenden Bilddatei ermittelt:

```
Dim bildname As String = Server.MapPath("img/" &
    & Request.QueryString("nummer") & ".gif")
```

Die eigentliche Verarbeitung wird innerhalb eines Try-Zweiges vorgenommen, damit Laufzeitfehler beim Dateizugriff nicht stören. Zum Dateizugriff wird ein FileStream-Objekt erzeugt:

Dateizugriff

```
Dim fs As FileStream = New FileStream(bildname, FileMode.Open)
```

Dann wird die Größe der Datei ermittelt und nach Integer konvertiert:

```
Dim groesse As Integer = CType(fs.Length, Integer)
```

Nun wird für diese Datei ein Pufferspeicher erzeugt. Da Bilder Binärdaten sind, eignet sich ein Byte-Array dafür:

```
Dim puffer() As Byte = New Byte(groesse) {}
```

Nun kann die Datei komplett in den Puffer gelesen werden:

Puffer füllen

```
fs.Read(puffer, 0, groesse)
```

Anschließend wird die Datei geschlossen:

```
fs.Close()
```

Jetzt können die Bilddaten an den Browser gesendet werden. Zuerst wird der eventuell bereits im Ausgabepuffer – der nichts mit dem Bildpuffer zu tun hat – befindliche Text gelöscht:

```
Response.ClearContent()
```

Daten senden Dann werden die binären Daten gesendet:

```
Response.BinaryWrite(puffer)
```

Damit ist die Übertragung beendet, das Bild erscheint. Am Ende des Skripts wird dies explizit angezeigt, damit nachfolgende Leerzeilen oder -zeichen den Prozess nicht stören:

```
Response.End()
```

Der Rest kümmert sich um die Ausnahmebehandlung, wenn erforderlich.



Abbildung 11.1: Ausgabe von Zahlenwerten mit dynamischer Bildzuordnung

11.2.2 Bilder direkt erzeugen

Der erste Abschnitt zeigte das Prinzip der dynamischen Bilderzeugung, bei dem die Bildquelle ein ASP.NET-Programm ist. Spannender ist die Möglichkeit, gleich die ganze Grafik dynamisch zu erzeugen. Im .NET-Framework gibt es eine große Zahl von Klassen, die der Bilderzeugung dienen. Durch ein einheitliches Schnittstellenkonzept können die erzeugten Bilddaten nicht nur Windows-Applikationen bereitgestellt, sondern direkt an Stream-Objekte übertragen werden. Einen solchen Datenstrom kann man speichern oder wie im vorangegangenen Beispiel mit `Response.BinaryWrite` an den Browser senden. Alternativ kann der Ausgabefunktion als Stream-Objekt auch `Response.OutputStream` übergeben werden, eine Repräsentation des Ausgabestroms zum Browser.

Das folgende Beispiel basiert wieder auf einer HTML-Vorlage. Dem Programm, das die Grafik erzeugt, werden zwei Parameter übergeben: `titel` und `text`.

```

<html lang="de">
  <head>
    <title>Dynamische Bilder</title>
  </head>
  <body>
    <h1>Dynamische Bilder</h1>
    <asp:image id="d5" runat="server"
      imageUrl="PixDynamicLade.aspx?titel=Bilder
%20erstellen&text=Hallo, ich bin ein dynamisches Bild"/>
  </body>
</html>

```

Listing 11.4: Aufruf des Grafikprogramms mit GET-Parametern (PixDynamic.aspx)

Angezeigt wird damit folgendes Banner:



Abbildung 11.2: Anzeige eines dynamisch erzeugten Bildes

Das eigentliche Geheimnis steckt im folgenden Programm:

```

<%@ Page language="vb" %>
<% @Import namespace="System.Drawing" %>
<% @Import namespace="System.Drawing.Imaging" %>
<script runat="server" language="vb">
Private Sub Page_Load()
    Try
        Dim text As String = Request.QueryString("text")
        Dim titel As String = Request.QueryString("titel")
        Dim ob As Bitmap = New Bitmap(460,80)
        Dim banner As Graphics = Graphics.FromImage(ob)
        Dim pinselGruen As SolidBrush =
            = New SolidBrush(Color.FromArgb(0,80,0))
        Dim pinselSchwarz As SolidBrush = New
SolidBrush(Color.Black)
        Dim pinselGelb As SolidBrush = New
SolidBrush(Color.Yellow)
        Dim fontTitel As Font = New Font("Verdana",20)
        Dim fontText As Font = New Font("Verdana",14)
        Dim titelGroesse As SizeF =
            = banner.MeasureString(titel,fontTitel)
        Dim textGroesse As SizeF =
            = banner.MeasureString(text,fontText)
        banner.FillRectangle (pinselGelb, 0, 0, 460, 80)

```

```

        banner.DrawString (titel, fontTitel, pinselSchwarz, ↵
                        230 - (titelGroesse.Width / 2), 8)
        banner.DrawString (text, fontText, pinselGruen, ↵
                        230 - (textGroesse.Width / 2), 40)
        Response.ClearContent()
        Response.ContentType = "image/jpeg"
        ob.Save(Response.OutputStream, ImageFormat.Jpeg)
        banner.Dispose()
        ob.Dispose()
    Catch e As Exception
        Response.ClearContent()
        Response.ContentType = "text/html"
        Response.Write ("Fehler: " + e.Message)
    End Try
    Response.End()
End Sub
</script>

```

Listing 11.5: Programm zum Erzeugen eines Banners (PixDynamicLade.aspx)

Wie es funktioniert

Das Programm ruft zuerst die GET-Parameter text und titel ab:

```

Dim text As String = Request.QueryString("text")
Dim titel As String = Request.QueryString("titel")

```

Bildgröße

Dann wird zuerst ein Bitmap-Objekt erzeugt, das die Basisgröße des Bildes festlegt:

```
Dim ob As Bitmap = New Bitmap(460,80)
```

Zeichenfläche

Auf der Grundlage dieser Zeichnungsfläche wird ein Graphics-Objekt erstellt, auf das nun Zeichenfunktionen angewendet werden können:

```
Dim banner As Graphics = Graphics.FromImage(ob)
```

Farben

Für die Ausgabe werden nun drei SolidBrush-Objekte erstellt, die zum Zeichnen durchgehender Flächen benötigt werden und einen Farbwert zugewiesen bekommen.

```

Dim pinselGruen As SolidBrush = New
SolidBrush(Color.FromArgb(0,80,0))
Dim pinselSchwarz As SolidBrush = New SolidBrush(Color.Black)
Dim pinselGelb As SolidBrush = New SolidBrush(Color.Yellow)

```

Fonts

Als Farbwerte können Farbnamen oder die Methode FromArgb für RGB-Farbwerte genutzt werden. Color ist eine Struktur, die vielfältige Farbumterstützung bietet, beispielsweise Konvertierungsmethoden. Nach den Farben müssen die Fonts ausgewählt wer-

den. Der Titel soll die Schriftart Verdana in einer Höhe von 20 Punkt bekommen:

```
Dim fontTitel As Font = New Font("Verdana",20)
```

Der Text wird nur 14 Punkt groß:

Schriftgröße

```
Dim fontText As Font = New Font("Verdana",14)
```

Da die Texte mehr oder weniger lang sein können, wird nun die tatsächliche Größe in Abhängigkeit von der gewählten Schriftart ermittelt:

```
Dim titelGroesse As SizeF = banner.MeasureString  
(titel,fontTitel)  
Dim textGroesse As SizeF = banner.MeasureString(text,fontText)
```

Nun wird der Hintergrund gezeichnet – ein gelbes Rechteck, das die gesamte Fläche ausfüllt:

Hintergrund

```
banner.FillRectangle (pinselGelb, 0, 0, 460, 80)
```

Die Methode DrawString zeichnet nun den Text auf die Fläche; angegeben werden die Zeichenkette, der Font, die Farbe und die Platzierung der linken oberen Ecke in einem XY-Koordinatensystem:

Zeichenmethoden

```
banner.DrawString  
(titel, fontTitel, pinselSchwarz, 230 - (titelGroesse.Width / 2), 8)  
banner.DrawString (text, fontText, pinselGruen, 230 - (textGroesse.Width / 2), 40)
```

Die Berechnung der X-Koordinate geht von der Hälfte des Banners aus (230 Pixel). Dann wird die Breite des Textes halbiert und abgezogen; damit steht der linke Rand fest. Die Höhe wird fest vorgegeben. Natürlich kann auch dieser Wert leicht in Abhängigkeit von der Schriftgröße berechnet werden, beispielsweise um den Text vertikal zu zentrieren.

Der Rest des Programms dient nur der Ausgabe. Zuerst wird der Ausgabepuffer geleert:

```
Response.ClearContent()
```

Dann wird das Bildobjekt mit der Methode Save direkt an den Ausgabedatenstrom übergeben. Die Methode erwartet außerdem die Angabe des Typs der Grafik. Im Beispiel wird JPEG genutzt:

**Ausgabe und
Pixelformat**

```
ob.Save(Response.OutputStream, ImageFormat.Jpeg)
```

Zuletzt werden die beiden Grafikobjekte (Bitmap und Graphics) explizit zerstört. Das erledigt zwar die Garbage Collection irgendwann automatisch, aber Grafiken sind außerordentlich speicherhungrig und die schnelle Entfernung aus dem Speicher entlastet das System deutlich. Ohne diese Maßnahmen wären Anwendungen mit dynamisch erstellten Bildern kaum unter Last¹ zu betreiben:

```
banner.Dispose()  
ob.Dispose()
```

Der übrige Teil des Programms dient nur der Fehlerbehandlung. Zuletzt wird die Ausgabe beendet, damit nachlaufende Leerzeilen die Ausgabe nicht stören und das Bild erscheint im Browser:

```
Response.End()
```

Cache An dieser Stelle sollten Sie übrigens auch über einen Cache für dynamische Grafiken nachdenken. Informationen dazu bietet der Abschnitt 8.7, »Optimierung des Datenverkehrs« ab Seite 809.

11.2.3 Eine kleine Grafikbibliothek

Die folgende Klasse zeigt, wie universelle Banner erzeugt werden können. Sie wird hier nur vorgestellt, um einen tieferen Einblick in die Grafikklassen zu geben. Eine Anwendung folgt im Abschnitt 12.2.3, »Programmierung eines eigenen Handlers« ab Seite 1031. Basis bildet die Übergabe von GET-Parametern, welche Sie der folgenden Tabelle entnehmen können:

Parameter	Bedeutung
w	Breite (Width)
h	Höhe (Height)
ff	Schriftartennamen (Font Face)
fs	Schrifthöhe in Punkt (Font Size)
c	Farbe (Color)
bgc	Hintergrundfarbe (Background Color)
bg	Hintergrundbild (Background Image)
b	Randbreite in Pixel (Border)
bc	Randfarbe (Border Color)

Tabelle 11.1: Die Parameter des Programms zum dynamischen Erzeugen von Bildern

1 Für hochfrequentierte Websites ist das sowieso keine besonders gute Idee.

Parameter	Bedeutung
a	Ausrichtung des Textes (Align)
t	Der Text selber
type	Das Bildformat (GIF; JPG etc.)

Tabelle 11.1: Die Parameter des Programms zum dynamischen Erzeugen von Bildern (Forts.)

Ausgestattet mit diesen Informationen kann die Klasse auch zum direkten Abruf aus einem -Tag heraus genutzt werden.

```
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging
Imports System.ComponentModel

Public Class PictureClass
    Private _width As Int16
    Private _height As Int16
    Private _fontface As String
    Private _fontsize As Double
    Private _rotate As Int16
    Private _color As String
    Private _bgcolor As String
    Private _background As String
    Private _border As Int16
    Private _bordercolor As String
    Private _align As String
    Private _text As String
    Private _type As String

    Private oSB1 As SolidBrush, oSB2 As SolidBrush
    Private oSP1 As Pen
    Private _context As HttpContext

    Friend Property Context() As HttpContext
        Get
            Return _context
        End Get
        Set(ByVal Value As HttpContext)
            _context = Value
        End Set
    End Property

    Public Sub ProcessPicture()
        Dim Request As HttpRequest = Context.Request
        Dim Response As HttpResponse = Context.Response
```

```
' extract params from querystring
If Not Request.QueryString("w") Is Nothing Then
    _width = Convert.ToInt16(Request.QueryString("w"))
Else
    _width = 480
End If
If Not Request.QueryString("h") Is Nothing Then
    _height = Convert.ToInt16(Request.QueryString("h"))
Else
    _height = 80
End If
If Not Request.QueryString("ff") Is Nothing Then
    _fontface = Request.QueryString("ff")
Else
    _fontface = "Verdana"
End If
If Not Request.QueryString("fs") Is Nothing Then
    _fontsize = 
    = Convert.ToSingle(Request.QueryString("fs"))
Else
    _fontsize = 12.0
End If
If Not Request.QueryString("c") Is Nothing Then
    _color = Request.QueryString("c")
Else
    _color = "black"
End If
If Not Request.QueryString("bgc") Is Nothing Then
    _bgcolor = Request.QueryString("bgc")
Else
    _bgcolor = "white"
End If
If Not Request.QueryString("bg") Is Nothing Then
    _background = Request.QueryString("bg")
Else
    _background = String.Empty
End If
If Not Request.QueryString("b") Is Nothing Then
    _border = Convert.ToInt16(Request.QueryString("b"))
Else
    _border = 1
End If
If Not Request.QueryString("bc") Is Nothing Then
    _bordercolor = Request.QueryString("bc")
Else
    _bordercolor = "blue"
End If
If Not Request.QueryString("a") Is Nothing Then
    _align = Request.QueryString("a").ToLower()
```

```

Else
    _align = "left"
End If
If Not Request.QueryString("t") Is Nothing Then
    _text = Request.QueryString("t")
Else
    _text = "No Text"
End If
If Not Request.QueryString("type") Is Nothing Then
    _type = Request.QueryString("type")
Else
    _type = "gif"
End If
If Not Request.QueryString("r") Is Nothing Then
    _rotate = Convert.ToInt16(Request.QueryString("r"))
Else
    _rotate = 0
End If
' build picture dynamically
' create bitmap
Dim oStringF As StringFormat = New StringFormat()
If (_rotate > 45 And _rotate < 135) ➤
    Or (_rotate > 225 And _rotate < 315) Then
        Dim _dummy As Int16 = _width
        _width = _height
        _height = _dummy
        oStringF.FormatFlags ➤
            = StringFormatFlags.DirectionVertical
    End If
Dim oBM As Bitmap = New Bitmap(_width, _height)
Dim oG As Graphics = Graphics.FromImage(oBM)
Dim cc As ColorConverter = New ColorConverter()
oSB1 = New
SolidBrush(CType(cc.ConvertFromString(_bgcolor), ➤
    System.Drawing.Color))
oSP1 = New Pen(CType(cc.ConvertFromString(_bordercolor), ➤
    System.Drawing.Color))
oSB2 = New SolidBrush(CType(cc.ConvertFromString(_color), ➤
    System.Drawing.Color))
Dim oF As Font = New Font(_fontface, CType(_fontsize,
Single))
oG.FillRectangle(oSB1, 0, 0, _width - _border, _height -
_border)
Dim oSF As SizeF = oG.MeasureString(_text, oF)
Dim oPF As PointF
Select Case _align
    Case "center"
        oStringF.Alignment = StringAlignment.Center
        oPF = New PointF(_width / 2 - (oSF.Width / 2), ➤

```

```

        _height / 2 - (oSF.Height / 2))
    Case "right"
        oStringF.Alignment = StringAlignment.Far
        oPF = New PointF(_width - oSF.Width, ␣
            _height / 2 - (oSF.Height / 2))
    Case Else
        oStringF.Alignment = StringAlignment.Near
        oPF = New PointF(0, _height / 2 - (oSF.Height / 2))
End Select
Dim oM As Matrix = New Matrix()
oM.Rotate(-_rotate, MatrixOrder.Prepend)
oG.Transform = oM
oG.SmoothingMode = SmoothingMode.AntiAlias
Dim oRF As RectangleF = New RectangleF(oPF, oSF)
oG.DrawString(_text, oF, oSB2, oRF, oStringF)
If _border > 0 Then
    oSP1.Width = Convert.ToSingle(_border)
    oG.DrawRectangle(oSP1, New Rectangle(0, 0, ␣
        _width, _height))
End If
' send picture
Response.ClearContent()
Select Case _type
    Case "jpg"
    Case "jpeg"
        Response.ContentType = "image/jpeg"
        oBM.Save(Response.OutputStream, ImageFormat.Jpeg)
    Case "png"
        Response.ContentType = "image/png"
        oBM.Save(Response.OutputStream, ImageFormat.Png)
    Case Else
        Response.ContentType = "image/gif"
        oBM.Save(Response.OutputStream, ImageFormat.Gif)
End Select
oBM.Dispose()
oG.Dispose()
Response.End()
End Sub
End Class

```

Listing 11.6: Eine universelle Klasse zum Erzeugen von Bannern (PictureClass.vb)

Die Grafik-funktionen

Die Bibliothek soll vor allem als Anregung für eigene Experimente dienen. Eine vollständige Erläuterung würde weit in die Tiefen von GDI+ führen und dies hat nicht besonders viel mit ASP.NET zu tun. Versuchen Sie nachzuvollziehen, wie die Ausgaben verarbeitet werden. Konzentrieren Sie sich bei der Suche in der Referenz auf die in der folgenden Tabelle aufgeführten Klassen, die die wichtigsten Abläufe beim Zeichnen von Bildern enthalten.

Klasse aus GDI+	Kurzbeschreibung
Graphics	Zeichnungsfläche eines Bildes
Bitmap	Abbildung der Pixelstruktur zur Ausgabe
Pen	Stift, wird zum Zeichnen von Linien und Konturen verwendet
Brush	Pinselfarbe, wird zum Zeichnen von Flächen benötigt
Font	Verwaltet Schriften
Color, ColorConverter	Verwaltung von Farben und Transformation

Tabelle 11.2: Die wichtigsten Klassen aus GDI+

Die eigentliche Zeichenarbeit erledigen Methoden der Klasse *Graphics*:

Methoden	Kurzbeschreibung
DrawLine	Linien
DrawPolygon	Polygone
DrawRectangle	Rechtecke
DrawString	Textausgabe
DrawImage	Einbinden von Bildern
DrawArc	Kreisbögen
DrawBezier	Kurven
DrawEllipse	Ellipsen und Kreise
DrawGrid	Gitter
DrawPie	Ausgefüllte Kreisform

Tabelle 11.3: Wichtige Zeichen-Methoden der Klasse *Graphics*

Die Bibliothek nimmt Werte an, die als QueryString übergeben wurden und erstellt daraus ein dynamisches Bild, das direkt an den Browser gesendet wird. Es ist Sache des aufrufenden Programms, dies in entsprechender Weise zu nutzen.

Zum Probieren wurde eine kleine Testapplikation entwickelt, die aus zwei Teilen besteht. Zuerst ein Formular, das die Einstellung diverser Parameter erlaubt. Sie finden es im Projekt *vbnetadvanced* unter dem Namen *PictureClassTest.aspx*.

Clientseitig wird JavaScript verwendet, um die Formularwerte in einen URL gemäß den Aufrufkonventionen in Tabelle 11.1 zu verwandeln. Aufgerufen wird das Programm *PictureClass.aspx*, das nur aus der folgenden Code Behind-Datei besteht:

Test der Klasse PictureClass

Dieses Programm testet die Klasse. Wählen Sie aus dem Formular verschiedene Parameter aus:

Breite	400	Punkt
Höhe	40	Punkt
Schriftart	Times	
Schriftgröße	12	Punkt
Farbe	Rot	
Hintergrundfarbe	Gelb	
Randbreite	☐ Keine ☒ 1 ☐ 2 ☐ 3 ☐ 5 Wert in Pixel	
Randfarbe	blau	
Ausrichtung des Textes	☒ Links ☐ Rechts	
Der zu schreibende Text	The quick brown fox...	
Bildformat	☒ GIF ☐ Jpeg ☐ PNG	

Abbildung 11.3: Das Formular zum Testen der Bilderzeugungsklasse

```
Public Class PictureClass1
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim pc As PictureClass = New PictureClass()
        pc.Context = System.Web.HttpContext.Current
        pc.ProcessPicture()
    End Sub

End Class
```

Listing 11.7: So werden die Parameter übernommen und an die Bildklasse weitergereicht (PictureClass.aspx.vb)

Hier wird nur eine Instanz der Klasse *PictureClass* erzeugt und der Kontext an die entsprechende Eigenschaft übergeben. Der Kontext enthält die POST- und GET-Parameter, auf die dadurch auch die nächste Klasse Zugriff hat. Mit dem Aufruf der Methode *ProcessPicture* erscheint das Bild in dem durch JavaScript erzeugten neuen Fenster.

Eine konkrete Anwendung der Klasse finden Sie in Abschnitt 12.2.3, »Programmierung eines eigenen Handlers« ab Seite 1031, wo auf die hier gemachten Ausführungen Bezug genommen wird.

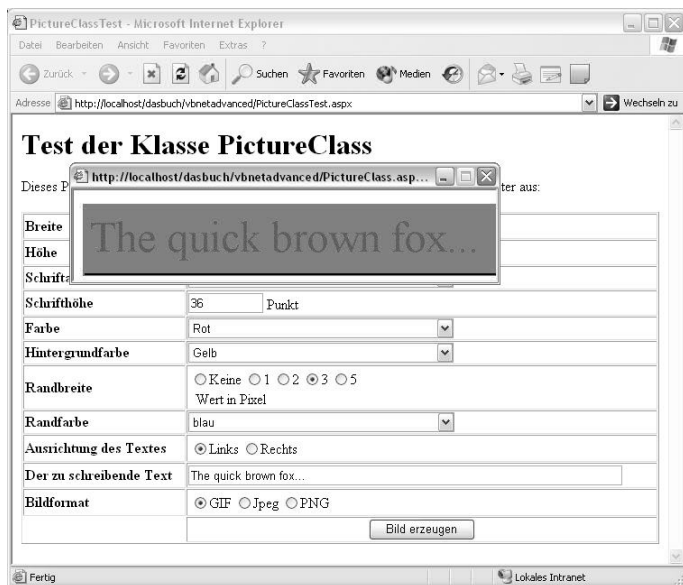


Abbildung 11.4: Das Testprogramm in Aktion

11.2.4 Hinweise zur Fehlersuche

Problematisch sind die Grafikfunktionen, weil die Ausgabe normalerweise nicht zu einer Fehlermeldung führt – das gilt auch für Compilerfehler. Der Browser erwartet durch den Header immer ein Bild. Er wird also im Zweifelsfall ein Ersatzbild anzeigen, statt der Fehlerausgabe aus dem *Catch*-Zweig.

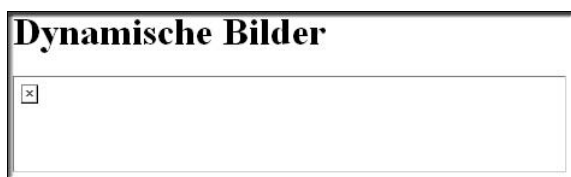


Abbildung 11.5: Offensichtlich funktioniert das Grafikprogramm nicht – dennoch erscheint kein Fehler.

In diesem Fall sollten Sie das Grafikprogramm direkt aufrufen, gegebenenfalls mit den notwendigen Parametern. Der Browser kann in jedem Fall ein Bild allein anzeigen, er benötigt das `<img>`-Tag nicht. Handelt es sich um Compilerfehler, werden diese in der üb-

lichen Form angezeigt. Andernfalls wird der Text der Fehlermeldung ausgegeben, wenn Sie die *Catch*-Anweisung entsprechend aufbauen.

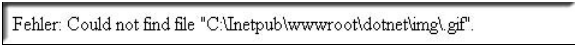
Tipps Falls die Gefahr von Laufzeitfehlern sehr groß ist, sollten Sie die Ausgabestrategie etwas ändern. Statt den MIME-Typ global festzulegen, können Sie diesen auch direkt als Eigenschaft `ContentType` des `Response`-Objekts angeben:

```
Response.ClearContent()
Response.ContentType = "image/gif"
ob.Save(Response.OutputStream, ImageFormat.Jpeg)
Response.End()
```

Die Ausgabe im *Catch*-Zweig sieht dann die Generierung von HTML vor. Es ist hier freilich eine Vereinfachung, sämtliche Fehler in einem Zweig abzufangen. Wenn Sie Datei- und Datenbankoperationen verwenden, sollten Sie weitere *Catch*-Abschnitte hinzufügen.

```
Catch e As Exception
    Response.ClearContent()
    Response.ContentType = "text/html"
    Response.Write ("Fehler: " + e.Message)
    Response.End()
End Try
```

Die Fehlermeldung erscheint, wenn der Browser HTML erkennt, in der Standardschrift Times. Wird dagegen Text erkannt, wird meist Courier verwendet. Sie können so prüfen, welcher *Content-type*-Header gesendet worden ist.



Fehler: Could not find file "C:\inetpub\wwwroot\dotnet\img\gif".

Abbildung 11.6: Fehlermeldung beim Aufruf eines Grafikprogramms

11.3 Versenden von E-Mail via SMTP

Kontaktformulare können Sie mit dem bisherigen Wissen bereits erstellen. Ebenso ist die Speicherung als Textdatei möglich. Eleganter wäre es natürlich, wenn der Server die Informationen gleich per E-Mail an Sie senden könnte – natürlich ohne das E-Mail-Programm des Benutzers zu verwenden. Die Benutzung des clientseitigen Programms ist eher unglücklich, weil viele Pro-

gramme drastische Warnungen beim Aufruf aus einer Webseite heraus aussprechen und die Existenz eines solchen Clients nicht sicher festgestellt werden kann. Lassen Sie den E-Mail-Versand serverseitig ablaufen, kann der Benutzer dies nicht beeinflussen und es stört in keiner Weise seine Privatsphäre.

11.3.1 Vorbereitung

Für den Versand von E-Mail sind einige vorbereitende Schritte notwendig. Zuerst müssen Sie wissen, dass der E-Mail-Verkehr im Internet über so genannte SMTP-Server (SMTP = Simple Mail Transfer Protocol) abgewickelt wird. Wenn ASP.NET eine E-Mail versenden soll, muss ein solcher Server zur Verfügung stehen. Wenn Sie derartige Programme auf einem lokalen Entwicklungssystem testen, müssen Sie auch daran denken, dass eine Verbindung zum Internet bestehen muss. SMTP basiert auf einer ständigen Erreichbarkeit des jeweiligen Servers. Im Gegensatz dazu arbeitet das Protokoll zum Abruf von E-Mail durch einen Client – POP3 – nur auf Anforderung. Dies wird oft verwechselt. Sie befinden sich nun auf der »Serverseite« der E-Mail und müssen sich mit den Gepflogenheiten bei SMTP auseinandersetzen.

Zum Glück macht es ASP.NET recht einfach, die nötigen Einstellungen vorzunehmen. Die Angabe des SMTP-Servers erfolgt einfach durch Setzen einer Eigenschaft des entsprechenden Objekts. Dennoch sollten Sie über einige Kenntnisse des Aufbaus von E-Mails verfügen, um auch speziellen Anforderungen der Programmierung gerecht werden zu können.

Die Kopfzeilen der E-Mail

RFC 822 definiert die in E-Mail-Nachrichten zulässigen Header. Die folgende Aufstellung nennt alle dort definierten Header. Die Reihenfolge ist nur »empfohlen«. Halten Sie sich trotzdem daran, einige Mail-Clients könnten sonst Probleme bekommen.

Richtig ist diese Reihenfolge:

- ▶ »Return-Path«
- ▶ »Received«

- ▶ »Date«
- ▶ »From«
- ▶ »Subject«
- ▶ »Sender«
- ▶ »To«
- ▶ »CC« usw.

Die folgende Aufstellung enthält alle Header: Der Punkt vor der Beschreibung gehört nicht zur Syntax. Angaben in eckigen Klammern [] sind optional:

- ▶ Return-path : *route-addr*
Pfad zurück zum Absender.
- ▶ Received : *further-parameters*
Für *further-parameters* setzen Sie einen oder mehrere der folgenden Parameter, jeweils zeilenweise getrennt, ein:
 - ▶ [from domain]
Name des sendenden Computers
 - ▶ [by domain]
Name des empfangenen Computers
 - ▶ [via atom]
Physischer Pfad zu E-Mail.
 - ▶ *(with atom)
Protokoll für die Übertragung.
 - ▶ [id msg-id]
Nachrichten-ID des Empfängers.
 - ▶ [for addr-spec] Formular.
 - ▶ ; *date-time*
date-time ist die Empfangszeit.
- ▶ [Reply-To : *address*]
address muss mindestens ein Mal angegeben werden und kann aus folgenden Elementen bestehen:
 - ▶ From : *mailbox*
Absenderadresse (Autor).

- ▶ Sender : *mailbox*
Sendeeinrichtung. Kann mit dem Autor identisch sein, kann aber auch seinen Senderserver bezeichnen und wird dann mit *From* kombiniert.
- ▶ [Resent-Reply-To : *address*]
Weiterleitungsinformation. Eine Angabe dazu kann aus zwei Elementen bestehen. Das Element ist optional, *address* besteht aus folgenden Elementen:
 - ▶ Resent-Sender : *mailbox*
 - ▶ Resent-From : *mailbox*
Weiterleitungsinformation mit Senderkennung und Absenderdaten.
- ▶ Date : *date-time*
Sendedatum der Nachricht.
- ▶ Resent-Date : *date-time*
Zeitpunkt einer Weiterleitung.
- ▶ To : *address*
Primäre Empfängeradresse. Mindestens eine Angabe muss hier stehen. Ergänzend können weitere Header folgen:
- ▶ Resent-To : *address*
Empfänger einer Weiterleitung.
- ▶ cc : *address*
Kopie an weiteren Empfänger (carbon copy)
- ▶ Resent-cc : *address*
Kopie einer Weiterleitung.
- ▶ bcc : *address*
Verdeckte Kopie (blind carbon copy)
- ▶ Resent-bcc : *address*
Verdeckte Kopie einer Weiterleitung.
- ▶ Message-ID : *msg-id*
Eindeutige ID des Empfängers.
- ▶ Resent-Message-ID : *msg-id*
Eindeutige ID einer Weiterleitung.

- ▶ **In-Reply-To** : **(phrase | msg-id)*
Bezugnahme einer Antwort auf eine andere E-Mail, die Zuordnung erfolgt durch eine Phrase (beispielsweise den Betreff) oder die Message-ID.
- ▶ **References** : **(phrase | msg-id)*
Referenz auf eine andere E-Mail, findet vor allem bei Mailinglisten Anwendung. Die Zuordnung erfolgt durch eine Phrase (beispielsweise den Betreff) oder die Message-ID.
- ▶ **Keywords** : *phrase*
Schlüsselworte; wird nur selten verwendet.
- ▶ **Subject** : *text*
Betreff der Nachricht, sollte unbedingt verwendet werden.
- ▶ **Comments** : **text*
Kommentare, wird von kaum einem Client unterstützt.
- ▶ **Encrypted** : *word*
Hinweis auf eine bestehende Verschlüsselung des Nachrichten-Körpers.
- ▶ *extension-field*
Erweiterung, nicht offiziell definiert.
- ▶ *user-defined-field*
Nutzerspezifische Erweiterungen, beispielsweise Hinweise auf Priorität. .
- ▶ *<addr-spec>*
Eindeutige ID der Nachricht (die spitzen Klammern gehören dazu).

Das folgende Beispiel zeigt, wie die Header einer E-Mail aussehen können. Die Mail entstammt einer Mailingliste. Die Header sind fett hervorgehoben, die optionalen Attribute kursiv. Wegen des begrenzten Platzes pro Zeile wurden einige Umbrüche eingefügt, diese sind mit ➤ gekennzeichnet. Generell beginnt auf jeder Zeile ein neuer Header, wenn der Text in der ersten Spalte startet. Gut zu erkennen sind die vielen zusätzlichen, nicht RFC 822-konformen Header und der komplexe Pfad, den diese E-Mail genommen hat (sicher auch provoziert durch eine Weiterleitung):

Return-path: <mailadmin@asp.net>
Envelope-to: joerg.krause.net@comzept.de
Delivery-date: Sat, 22 Apr 2000 00:31:15 +0200
Received: from [208.49.167.126] (helo=mb3.mailbank.com) ↵
by mx01.kundenserver.de with esmtp (Exim 2.12 #2) ↵
id 12ilw7-00007M-00 ↵
for joerg@krause.net; Sat, 22 Apr 2002 00:30:47
+02 ↵
Received: from info.asp.net (MAIL.ASP.NET) ↵
by mb3.mailbank.com (8.9.1/8.9.1) with SMTP id
PAA29717 ↵
for ; <joerg@krause.net> ↵
Fri, 21 Apr 2000 15:30:04 -0700
Received: (qmail 14450 invoked from network); 21 Apr 2000 ↵
22:26:17 -0000
Received: from solix.wiso.uni-koeln.de (daemon@134.95.183.82)
by solix.wiso.uni-koeln.de with SMTP; 21 Apr 2000 ↵
22:26:17 -0000
Received: from twdc.de (qmailr@[212.172.92.2])
by mail.asp.net (8.8.8/8.8.8) with SMTP id ↵
AAA14428 ↵
for <asp@asp.net>; Sat, 22 Apr 2000 00:26:10 +0200
Received: (qmail 20861 invoked from network); 21 Apr 2000 ↵
19:39:42 -0000
Received: from dialin126-nt.pg7.frankfurt.nikoma.de ↵
(HELO claudia) (213.54.38.126)
by 212.172.92.3 with SMTP; 21 Apr 2000 19:39:42
-0000
Message-ID: <001501bfabd2\$188c6840\$7e2636d5@claudia>
From: "Joerg Krause" <joerg@krause.net>
To: <mailing@mailserver.asp.net> ASP.NET-LIST
References: <004501bfabac\$e9c52800\$d47c353e@n4n8u9> ↵
<39009410.44C4B172@kawo2.rwth-aachen.de>
Subject:Re: [dotnet] Einführung in ASP.NET
Date: Fri, 21 Apr 2000 20:50:37 +0200
Organization: Comzept GmbH
MIME-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.00.2014.211
X-MimeOLE: Produced By Microsoft
MimeOLE V5.00.2014.211
Reply-To: mailing@mailserver.asp.net
Sender: mailing@mailserver.asp.net

```

Errors-To: admin@mailserver.asp.net
X-Mailman-Version: 1.0b8
Precedence: bulk
List-Id: deutschsprachige ASP.NET-Mailingliste  ↵
        <mailing@mailserver.asp.net>
X-BeenThere: <mailing@asp.net>
Content-Transfer-Encoding: quoted-printable
X-MIME-Autoconverted: from 8bit to quoted-printable by  ↵
        mb3.mailbank.com id PAA29717

```

Listing 11.8: Header einer »gewöhnlichen« E-Mail aus einer Mailingliste

11.3.2 Eine E-Mail-Anwendung programmieren

Das .NET-Framework stellt E-Mail-Funktionen für Webserver im Namensraum `System.Web.Mail` zur Verfügung. Die Bekanntgabe erfolgt entweder in der HTML-Seite mit folgender Direktive:

```
<%@Import namespace="System.Web.Mail" %>
```

Alternativ ergänzen Sie eine Datei mit hinterlegtem Code folgendermaßen:

```
Imports System.Web.Mail
```

Basis der E-Mail-Anwendung: Ein Kontaktformular

Damit es überhaupt etwas zu versenden gibt, beginnt die Entwicklung der E-Mail-Anwendung mit einem Kontaktformular. Dieses ist bewusst einfach gehalten. Mit den Informationen über Steuerelemente können Sie es leicht an Ihre Bedürfnisse anpassen und erweitern.

Der Code zum Versenden des Formularinhalts als E-Mail wird in einer Code-Datei hinterlegt (Code Behind).

```

<html lang="de">
  <head>
    <title>Kontaktformular</title>
  </head>
  <body>
    <h1>Kontaktformular</h1>
    <form runat="server">
      <table>
        <tr>

```



```
<td>Name</td>
<td>
  <asp:textbox runat="server" id="name"
    width="250px"/>
  <asp:requiredfieldvalidator runat="server"
    id="checkname"
    controltovalidate="name"
    display="dynamic">
    <div style="color:red">
      Geben Sie einen Namen ein</div>
    </asp:requiredfieldvalidator>
</td>
</tr>
<tr>
<td>Anschrift</td>
<td>
  <asp:textbox runat="server" id="anschrift"
    width="250px"/>
</td>
</tr>
<tr>
<td>Plz / Ort</td>
<td>
  <asp:textbox runat="server"
    id="plz" width="50px"/>
  <asp:textbox runat="server"
    id="ort" width="196px"/>
</td>
</tr>
<tr>
<td>Telefon</td>
<td>
  <asp:textbox runat="server" id="telefon"/>
  mit Vorwahl
</td>
</tr>
<tr>
<td>E-Mail</td>
<td>
  <asp:textbox runat="server"
    id="email" width="250px"/>
</td>
</tr>
<tr>
<td>Anschrift</td>
<td>
  <asp:textbox textmode="multiline"
    runat="server"
    id="nachricht" width="250px">
```

```

        height="100px"/>
    </td>
</tr>
<tr>
    <td colspan="2" align="right">
        <asp:button runat="server" id="send"
            text="Senden" onclick="SendMail"/>
    </td>
</tr>
</table>
<asp:label runat="server" id="ergebnis"/>
</form>
</body>
</html>

```

Listing 11.9: Kontaktformular, das zum Versenden per E-Mail verwendet wird (EmailForm.aspx)

Das Formular verfügt nur über eine einfache Prüfung des Namens. Die gesamte Arbeit erledigt die Klasse Mail.

E-Mail versenden

Das Versenden ist weniger aufwändig als das Erstellen des Formulars:

```

Imports System.Web.Mail

Public Class Mail
    Inherits Page
    Public name As TextBox, anschrift As TextBox, ↵
        plz As TextBox, ort As TextBox, telefon As TextBox, ↵
        email As TextBox, nachricht As TextBox
    Public ergebnis As Label
    Private eMailMessage As MailMessage
    Private eMailText As String

    Public Sub SendMail(ByVal sender As Object, ↵
        ByVal ea As EventArgs) _
        Handles send.Click
        Try
            eMailMessage = New MailMessage()
            eMailText = "Kontaktformular:" + vbCrLf
            eMailText += "Name      : " + name.Text + vbCrLf
            eMailText += "Anschrift: " + anschrift.Text + vbCrLf
            eMailText += "Plz/Ort  : " + plz.Text + " " ↵

```

```

                                + ort.Text + vbCr
eMailText += "E-Mail   : " + email.Text + vbCr
eMailText += "Telefon  : " + telefon.Text + vbCr
eMailText += "Nachricht: " + vbCr
eMailText += nachricht.Text + vbCr + vbCr
eMailMessage.From = email.Text
eMailMessage.To = "webmaster@comzept.de"
eMailMessage.Subject = "Kontaktformular ASP.NET"
eMailMessage.Body = eMailText
SmtpMail.SmtpServer = "smtp.kundenserver.de"
SmtpMail.Send(eMailMessage)
email = Nothing
ergebnis.Text = "Nachricht erfolgreich gesendet."
Catch e As Exception
    ergebnis.Text = "FEHLER: " + e.Message
End Try
End Sub
End Class

```

Listing 11.10: Programm zum Versenden von Formulardaten per E-Mail
(EmailForm.aspx.vb)

Das Programm versendet eine E-Mail in Textform. .NET unterstützt auch HTML-Mails und Anhänge, dazu folgen am Ende des Abschnitts noch Hinweise. Die gesamte Arbeit erledigt die Methode *SendMail*, die das *onClick*-Ereignis der Sendeschaltfläche verarbeitet. Auf *Page_Load* kann damit verzichtet werden:

Wie es funktioniert

```

Public Sub SendMail(ByVal sender As Object, _
                    ByVal ea As EventArgs) _
    Handles send.Click

```

Die Methode startet mit einem *Try*-Zweig. Das Versenden von E-Mail kann an vielen Problemen scheitern; beispielsweise kann der SMTP-Server nicht erreichbar sein, die Nachricht ablehnen, Netzwerkfehler können stören usw. Alle Fehler werden mit einem universellen *Catch* abgefangen.

Fehler abfangen

Zuerst wird nun die eigentliche Nachricht zusammengebaut. Basis ist ein Objekt der Klasse *MailMessage*:

Nachrichte aufbauen

```
eMailMessage = New MailMessage()
```

Jetzt wird der Text der Nachricht erstellt:

```
eMailText = "Kontaktformular:" + vbCr
```

Als Zeilenendezeichen wird die Konstante `vbCr` verwendet. In Text-E-Mails dürfen Sie natürlich nicht `<br/>` verwenden, dies ist HTML vorbehalten. Ohne Zeilenumbruch würde aber der gesamte Text in einer Zeile stehen. Die übrigen Zuweisungen lesen die Formularfelder aus.

Absender Dann sind Absender und Empfänger zu bestimmen. Zuerst der Absender, dessen Wert wird dem Formular entnommen – in der Hoffnung, der Benutzer gibt seine E-Mail-Adresse vernünftig an:

```
eMailMessage.From = email.Text
```

Empfänger Dann wird der Empfänger bestimmt. Tragen Sie hier die Adresse ein, an die das Formular gesendet werden soll:

```
eMailMessage.To = "webmaster@comzept.de"
```

Betreff Der Betreff kann auch festgelegt werden. Es ist sinnvoll, dies nicht dem Benutzer zu überlassen, damit Sie die E-Mails vom Kontaktformular leicht erkennen können:

```
eMailMessage.Subject = "Kontaktformular ASP.NET"
```

Text Zuletzt wird noch der Text an das Objekt übergeben:

```
eMailMessage.Body = eMailText
```

Damit der Versand auch funktioniert, ist nun der SMTP-Server zu bestimmen. Ist – unter Windows 2000 Pro, .NET-Server oder XP Pro – der SMTP-Server korrekt installiert, verwenden Sie »localhost«. Andernfalls tragen Sie den SMTP-Server Ihres Providers ein:

```
SmtpMail.SmtpServer = "smtp.meinserver.de"
```



Sollte der SMTP-Server des Providers mit einer Fehlermeldung der Art »Relaying prohibited« reagieren, wurden Sie nicht als autorisierter Benutzer erkannt. Meist kann dies umgangen werden, indem unmittelbar vorher eine autorisierte Abfrage des POP3-Servers erfolgt, also Post abgeholt wird.

Nachdem nun alles vorbereitet wurde, kann der E-Mail-Versand erfolgen:

```
SmtpMail.Send(eMailMessage)
```

Zuletzt wird das `Mail`-Objekt vernichtet und das Programm endet mit einer Erfolgsmeldung.

Kontaktformular

Name

Anschrift

Plz / Ort

Telefon mit Vorwahl

E-Mail

Anschrift

Dies ist eine Textnachricht, um zu prüfen, ob der E-Mail-Versand funktioniert.

Nachricht erfolgreich gesendet.

Abbildung 11.7: Viel Formular und wenig E-Mail – der Versand geht eher still vor sich.

Wenn Sie die ersten Versuche an sich selbst senden, sollten Sie im Postfach bald die entsprechenden Nachrichten vorfinden.



Abbildung 11.8: Nachricht in Outlook 2000, die mit dem Kontaktformular erzeugt wurde

Abschließend soll noch auf einen speziellen Fehler hingewiesen werden, der häufig auftritt:

Fehlerquellen

Could not access 'CDO.Message' object

Schalten Sie das ausführliche Debugging an und fangen den Laufzeitfehler nicht mit *Catch* ab, erhalten Sie einen besseren Hinweis der folgenden Art:

Es ist mindestens eines der Felder "Von" oder "Absender" erforderlich,
es wurde jedoch keines der Felder gefunden.

Meist wurde vergessen, die Eigenschaft `From` zu füllen. Bauen Sie hier einen `RequiredFieldValidator` ein, um den Fehler zu vermeiden.

11.3.3 Spezielle E-Mail-Funktionen

Massenpost

Das Versenden von E-Mail in der vorgestellten Form kann leicht erweitert werden. Wollen Sie Massenpost versenden, muss nur der Inhalt der Eigenschaft `To` ausgetauscht werden. Natürlich muss es nicht gleich Massenpost sein. Denken Sie beispielsweise daran, das Kontaktformular mit einer pauschalen Reaktion auch gleich an den Benutzer selbst zu senden. Er weiß dann sicher, dass der Versand funktioniert hat. Dies ist gute Benutzerführung und kundenfreundlich.



Beachten Sie, dass das Versenden von Massenpost an Personen, mit denen Sie nicht in einer laufenden Geschäftsbeziehung stehen, strafbar ist. Verwenden Sie Rundschreiben nur, wenn der Benutzer sich vorher damit einverstanden erklärt hat, beispielsweise durch Beitritt zu einer Mailingliste oder einem Newsletter.

Dateianhänge

Eine weitere Funktion betrifft Dateianhänge. Dazu wird die Klasse `MailAttachment` eingesetzt. Es ist sehr einfach, die Anforderung einer Datei per E-Mail mit ASP.NET zu programmieren:

```
Dim mailDatei As MailAttachment ←  
    = New MailAttachment("pfad/datei.pdf")  
eMail.Attachments.Add(mailDatei)
```

Die Eigenschaft `Attachments` bildet eine Kollektion; es können deshalb leicht mehrere Dateianhänge hinzugefügt werden.

HTML-Mails

Die letzte Sonderfunktion betrifft HTML-Mails. Dazu ist nur eine einzige Zeile Code hinzuzufügen:

```
eMailMessage.BodyFormat = Mailformat.Html
```

`Mailformat` ist eine Aufzählung, die als Aufzählungswerte `Text` oder `Html` enthält. Jetzt müssen Sie natürlich den Text der E-Mail entsprechend aufbereiten:

```
<html> <body> <h1>Hallo, willkommen bei unserem  
HTML-Newsletter</h1> ...
```

11.4 Zugriff auf andere Server

Wenn der Zugang zu Daten auf anderen Servern mit den dafür designierten Methoden – Webservices oder XML-Datenaustausch – nicht gelingt, ist ein direkter Zugriff per HTTP angebracht. Auch zum Schreiben von Suchmaschinen-Robots ist dieses Verfahren, »Grabbing« genannt, bestens geeignet. Abgesehen von rechtlichen Aspekten ist das Prinzip in ASP.NET recht einfach. Ob die so gewonnenen Daten auch verwendet werden dürfen, ist allerdings keine technische Frage.

11.4.1 Prinzip des Netzwerkzugriffs

Was immer Sie bisher in ASP.NET programmiert haben, ein direkter Zugriff auf die Protokolle war kaum notwendig. Am Rande tauchte zwar der eine oder andere Begriff aus der HTTP-Welt auf, aber die ASP.NET-Klassen stellen doch eine starke abstrahierte Sicht auf die in den Tiefen laufenden Daten dar.

Soll der natürliche Weg verlassen werden und der Server zum Browser mutieren, müssen andere Techniken Verwendung finden. Die Basis bilden Klassen, die dem Namensraum `System.Net` entstammen.

System.Net

Beachten, Sie dass der Zugriff auf fremde Server dem eines Browsers ähnelt. Sie erhalten also niemals Zugang zu den programmierten Seiten selbst, sondern nur zum fertigen HTML. Wenn Sie eine ASP.NET-Seite abfragen, wird der fremde Server diese erst ausführen und Ihnen erst dann das Ergebnis senden.



Zugriff auf einen Server per HTTP

Das folgende Beispiel zeigt das Prinzip. Es erwartet die Eingabe eines URL und stellt die empfangenen Daten im Klartext dar – als HTML-Code also. Ein einfaches Formular bildet die Benutzerschnittstelle:

```
<h1>Webgrabbing</h1>
<h2>HTML-Code einer fremden Seite anschauen</h2>
<form id="WebGrabbing" method="post" runat="server">
  http://<asp:TextBox Runat="server" ID="GrabUrl"/>
  <asp:Button OnClick="Grab_Click" 
    Runat="server" ID="Grab"/>
```

```

        </form>
    </pre>
    <asp:Label Runat="server" ID="GrabbedSite" />
    </pre>

```

Listing 11.11: Erfassung einer URL und Platz für die Ausgabe (Relevanter Teil aus WebGrabbing.aspx)

Wie üblich wird die Arbeit in einer Code-Datei erledigt. Dort ist die Ereignisbehandlungsmethode *Grab_Click* zuständig:

```

Imports System.IO
Imports System.Net
Imports System.Text
Imports System.Collections
Imports System.ComponentModel
Imports System.Data

Public Class WebGrabbing
    Inherits System.Web.UI.Page
    Protected GrabbedSite As Label
    Protected GrabUrl As TextBox
    Protected WithEvents Grab As Button

    Const EOF As Integer = -1

    Private Sub Page_Load(ByVal sender As Object, _
                          ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Page.EnableViewState = False
    End Sub

    Public Sub Grab_Click(ByVal sender As Object, _
                        ByVal e As System.EventArgs)
        Dim Url As String = GrabUrl.Text
        If Not Url.Equals(String.Empty) Then
            Url = "http://" + Url
            Dim wr As WebRequest = WebRequest.Create(Url)
            Dim wp As WebResponse = wr.GetResponse()
            Dim sr As StreamReader _
                = New StreamReader(wp.GetResponseStream(), _
                                Encoding.UTF7)
            Dim sb As StringBuilder = New StringBuilder()
            Dim i As Integer = 0
            While sr.Peek() <> EOF
                sb.AppendFormat("{0,0:D4}: {1}<br/>", i, _
                                Server.HtmlEncode(sr.ReadLine()))
                i = i + 1
            End While
            GrabbedSite.Text = sb.ToString()
        End Sub
    End Class

```



```

Else
    GrabbedSite.Text = "Textfeld darf nicht leer sein"
End If
End Sub
End Class

```

Listing 11.12: Grabben von Webseiten und Ausgabe des Inhalts (WebGrabbing.aspx.vb)

Das Programm bildet im Wesentlichen zwei Vorgänge ab: Request und Response. Zuerst wird eine Abfrage erstellt:

Wie es funktioniert

```
Dim wr As WebRequest WebRequest.Create(Url)
```

Dann wird sie ausgeführt und der Zugriff auf den Eingangsdatenstrom gewährt:

```
Dim wp As WebResponse = wr.GetResponse()
```

Durch das hervorragende Klassenkonzept von .NET ist dieser Datenstrom kompatibel zu der in `System.IO` definierten `Stream`-Klasse und kann deshalb sofort an ein entsprechendes Objekt übergeben werden:

```
Dim sr As StreamReader = new StreamReader (
    wp.GetResponseStream(), Encoding.ASCII)
```

Mit dem `StreamReader` kann nun bequem auf die Daten zugegriffen werden. Im Beispiel wird das benutzt, um die HTML-Codes zeilenweise auszugeben. Die Daten werden in einem `StringBuilder`-Objekt gesammelt, das mit großen Zeichenketten effizienter umgeht als eine einfache Zeichenkettenverknüpfung:

```
Dim sb As StringBuilder = New StringBuilder()
```

Dann wird der Datenstrom zeilenweise gelesen:

```
While sr.Peek() <> EOF
```

Die Zeilen werden von einer fortlaufenden Zeilennummer eingeleitet, außerdem soll der empfangene Code kodiert werden, sodass die HTML-Tags nicht ausgeführt, sondern angezeigt werden:

```
sb.AppendFormat (" {0,0:D4}: {1}<br/>",
    i,
    Server.HtmlEncode (sr.ReadLine ()))
i = i + 1
```

Mit der Zuweisung an das `Label`-Steuerelement ist der Vorgang auch schon beendet:

```
GrabbedSite.Text = sb.ToString()
```

Zu beachten ist bei der Angabe der Adresse, dass diese vollständig, mit führendem »http://« angegeben werden muss.

**Anzeigestatus
unterdrücken**

Außerdem wurde der Anzeigestatus unterdrückt, sonst landet der gesamte Inhalt des `Label`-Steuerelementes nochmal im *ViewState* und bläht die Seite unnütz auf.

Webgrabbing

HTML-Code einer fremden Seite anschauen

```

0000: <html>
0001:   <head>
0002:     <meta http-equiv="Refresh" content="1;URL=http://www.bueninging-kr
0003:     <meta name="Keywords" content="ASP, ASP.NET, FrontPage, 2002, 20
0004:   </head>
0005:   <body bgcolor="#ffffff">
0006:   <h1 style="margin:40px; padding:20px; border:dotted blue 2px; color:blue;
0007: Die Website zu den B?chern "ASP.NET lernen", "ASP.NET mit C#" und "ASP.NET
0008: </h1>
0009:   <div style="margin:40px; padding:20px; border:dotted blue 2px; color:blu
0010:
0011:   Weiterleitung erforderlich. Klicken Sie auf den Link, wenn der Browser i
0012:
0013:   <p/>
0014:
0015:   Neue Adresse:
0016:   <a href="http://www.bueninging-krause.de/templates/dotnet/default.aspx">ht
0017:
0018:   </div>
0019: </body>
0020: </html>

```

Abbildung 11.9: Auslesen einer fremden Webseite

**Nach Code-Teilen
suchen**

Wollen Sie nun nach bestimmten Stellen in der Seite suchen, verwenden Sie am besten reguläre Ausdrücke. Diese werden in Abschnitt 4.7, »Reguläre Ausdrücke« ab Seite 267 detailliert vorgestellt.



Das vorgestellte Beispiel ist in einer Beziehung noch unfertig: Wenn die Adresse nicht existiert, wird der Laufzeitfehler `WebException` ausgelöst. Erweitern Sie das Programm selbstständig um den erforderlichen Try-Catch-Block.

Abfrage von WHOIS-Servern

**Netzwerkpro-
grammierung:
Domains
erkunden**

WHOIS ist ein Internetdienst, der Informationen über die Inhaber einer Domain bereit stellt. Die Schnittstelle zu der dahinter liegenden Datenbank ist über den TCP-Port 43 erreichbar. Um mit ASP.NET darauf zugreifen zu können, müssen Sie den bekannten Weg der vorgefertigten Klassen verlassen und noch eine Ebene tiefer gehen, hinab zu den Sockets. Sockets sind kom-

plette Verbindungsbeschreibungen; das sind paarweise auftretende Kombinationen aus einer IP-Adresse und einem TCP-Port. Wenn Browser und Server miteinander kommunizieren, dann geschieht dies auf hoher Protokollebene (im OSI-Modell) mit HTTP. Darunter jedoch arbeitet TCP/IP und stellt primitivere Datenübertragungswege bereit.

Natürlich kann auch diese Schicht mit .NET programmiert werden. Das ist beispielsweise dann notwendig, wenn ein »exotischer« Dienst wie WHOIS benötigt wird.

Das Programm besteht aus zwei Teilen: einem einfachen Formular zur Ein- und Ausgabe und dem Code mit den Socket-Funktionen. Das Formular finden Sie unter dem Namen *WhoIsDotNet.aspx*, der Code wird nachfolgend vorgestellt:

```
Imports System.IO
Imports System.Net
Imports System.Net.Sockets
Imports System.Text
Imports System.Collections
Imports System.ComponentModel

Public Class WhoIsDotNet
    Inherits System.Web.UI.Page
    Protected WhoisDomain As TextBox
    Protected WhoisResult As Label
    Protected WithEvents Grab As Button

    Const EOF As Integer = -1

    Private Sub Page_Load(ByVal sender As Object, 
        ByVal e As System.EventArgs)
        Page.EnableViewState = False
    End Sub

    Public Sub WhoIs_Click(ByVal sender As Object, 
        ByVal e As System.EventArgs)
        Dim tc As TcpClient = New TcpClient()
        Try
            tc.SendTimeout = 8000
            tc.Connect("whois.denic.de", 43)
            Dim domain As String = WhoisDomain.Text + vbCrLf
            Dim ba() As Byte 
                = Encoding.ASCII.GetBytes(domain.ToCharArray())
            Dim s As Stream = tc.GetStream()
            s.Write(ba, 0, domain.Length)
            Dim sr As StreamReader 
```

```

        = New StreamReader(tc.GetStream(),
                               Encoding.ASCII)
    Dim sb As StringBuilder = New StringBuilder()
    Dim i As Integer = 0
    While sr.Peek() <> EOF
        sb.AppendFormat("{0,0:D3}: {1}<br/>", i,
sr.ReadLine())
        i = i + 1
    End While
    WhoisResult.Text = sb.ToString()
    tc.Close()
    Catch ex As SocketException
        WhoisResult.Text = "WHOIS-Fehler: " + ex.Message
    End Try
End Sub
End Class

```

Listing 11.13: Abfrage des WHOIS-Servers des DENIC (WholsDotNet.aspx.vb)

Wie es funktioniert

Das Programm basiert auf einer neuen TCP-Verbindung (die IP-Ebene wird durch die Netzwerkkonfiguration des Servers bestimmt):

```
Dim tc As TcpClient = new TcpClient()
```

Damit Sie im Fehlerfall nicht endlos warten, wird eine Zeitüberschreitung von acht Sekunden festgelegt – die Methode erwartet ein Argument in Millisekunden:

```
tc.SendTimeout = 8000
```

WHOIS-Server im Netz

Dann wird Verbindung mit einem WHOIS-Server aufgenommen. Eine Liste von derartigen Diensten finden Sie unter anderem bei www.whoisservers.net/. Hier wurde der Dienst des DENIC ausgewählt, die Abfrage gelingt deshalb nur für »de«-Domains:

```
tc.Connect("whois.denic.de", 43)
```

Nun muss ein WHOIS-Kommando in den Datenstrom geschrieben werden. Das ist einfach, weil es sich nur um den Domain-Namen mit einem Zeilenende-Zeichen handelt. Auf dieser Ebene wird mit Bytes gearbeitet, deshalb muss die interne Unicode-Darstellung umgewandelt werden:

```
Dim ba() As Byte = Encoding.ASCII.GetBytes(domain.ToCharArray())
```

Dann wird der Zugriff auf den Datenstrom organisiert:

```
Dim s As Stream tc.GetStream()
```

Nun werden die Bytes aus dem Array *ba* gesendet:

```
s.Write(ba, 0, domain.Length)
```

Danach geht es an die Auswertung der Antwort:

```
Dim sr As StreamReader = new StreamReader
    (tc.GetStream(), Encoding.ASCII)
```

Die Dekodierung erfolgt sofort nach ASCII, denn ein WHOIS-Server sendet nur ASCII-Daten zurück.

Dekodierung

Der Rest des Programms entspricht dem in Listing 11.12 und dient der Ausgabe über den *StringBuilder*. Die Fehlerkontrolle wurde hier gleich mit eingebaut. Die Ausnahme, die auftreten kann, heißt *SocketException*.

Damit ist das Programm schon fertig; abschließend ein Blick auf eine typische Ausgabe:

Whois vom whois.denic.de

WHOIS für eine Domain abfragen

Domain (ohne http://www.):

```

000: % Copyright (c)2002 by DENIC
001: %
002: % Restricted rights.
003: %
004: %
005: % Except for agreed Internet operational purposes, no part of this
006: % information may be reproduced, stored in a retrieval system, or
007: % transmitted, in any form or by any means, electronic, mechanical,
008: % recording, or otherwise, without prior permission of the DENIC
009: % on behalf of itself and/or the copyright holders. Any use of this
010: % material to target advertising or similar activities are explicitly
011: % forbidden and will be prosecuted. The DENIC requests to be notified
012: % of any such activities or suspicions thereof.
013:
014: domain:      joergkrause.de
015: deacc:       Joerg Krause
016: deacc:       Hans-Fallada-Strasse 36
017: deacc:       D-12683 Berlin
018: deacc:       Germany
019: nservic:     ns2.schlund.de
020: nservic:     ns1.schlund.de
021: status:      connect
022: changed:     19991126 052100
023: source:      DENIC
024:
025: [admin-c]
026: Type:        PERSON
027: Name:        Joerg Krause
028: Address:     Hans-Fallada-Strasse 36
029: City:        Berlin
030: Pcode:       12683
031: Country:     DE
032: Changed:     20000322 191021
033: Source:      DENIC
034:
035: [tech-c][some-c]
036: Type:        ROLE
037: Name:        Hostmaster SCHLUND
038: Address:     Schlund + Partner AG
039: Address:     Erbprinzenstr. 1
040: City:        Karlsruhe
041: Pcode:       76133
042: Country:     DE
043: Phone:       +49 721 91576 50
044: Fax:         +49 721 91576 20
045: Email:       hostmaster@schlund.de
046: Remarks:     Information: http://www.schlund.de
047: Remarks:     Question: hostinfo@hostmaster@schlund.de
048: Remarks:     For fastest response, please use only hostmaster@schlund.de
049: Remarks:     Role Account for the "Hostmaster of the Day"
050: Remarks:     for Domains managed by Schlund + Partner AG, Germany
051: Changed:     20000321 160314
052: Source:      DENIC
053:

```

Abbildung 11.10: Eine Whois-Abfrage mit ASP.NET

12 Servererweiterungen

Was früher als ISAPI-Filter und ISAPI-Erweiterung mühevoll mit C++ für den IIS programmiert werden musste, ist heute Bestandteil von ASP.NET und mit VB.NET einfacher als jemals zuvor. Lesen Sie in diesem Kapitel, wie Sie das HTTP-Handler-Konzept von ASP.NET und eigene Anwendungen erweitern können.

12.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

12.1.1 Über dieses Kapitel

Die Erweiterung der Funktionalität vom alten ASP war Sache der Programmierung der ISAPI-Schnittstelle. ISAPI steht für *Internet Services Application Programming Interface* und stellt eine Schnittstelle zum IIS dar, die – vorzugsweise – mit C++ programmiert werden konnte. Dabei gab es zwei Verfahren. ISAPI-Erweiterungen dienten der Behandlung bestimmter, individuell vergebener Datei-Erweiterungen und konnten damit fehlende Funktionen in ASP kompensieren. ISAPI-Filter dagegen wurden in den Datenstrom zwischen Browser und Client »eingehängt« und konnten diesen verändern – in beide Richtungen.

Die Programmierkenntnisse, die ein ISAPI-Entwickler mitbringen musste, waren erheblich größer als für die ASP-Programmierung. Außerdem war eine effektive Entwicklung nur mit C++ möglich, was vermutlich die von VBScript am weitesten entfernte Sprache ist. Insofern war die ISAPI-Programmierung nur selten Teil der

täglichen Arbeit des ASP-Programmierers, obwohl es sicher zahlreiche Anwendungsfälle gegeben hätte.

Mit .NET hat sich auch die Programmierung der Schnittstellen grundlegend geändert. Da nun eine richtige Programmiersprache und eine umfassende Klassenbibliothek zur Verfügung steht, besteht kein Grund mehr, für derartige Filter oder Handler auf eine andere Sprache auszuweichen.

Auch wenn die Programmierung heute anders aussieht, gibt es dennoch eine direkte Entsprechung der beiden ISAPI-Verfahren. Generell werden alle Anfragen aus dem Web von einem so genannten *HttpHandler* beantwortet. Genau diese Module kann man – verhältnismäßig einfach – auch selbst programmieren. Sie stellen dann ungefähr die Leistungen zur Verfügung, wie die alten ISAPI-Erweiterungen. Mehr dazu finden Sie in Abschnitt 12.2, »HTTP-Handler« am Ende dieser Seite.

Auch für die ISAPI-Filter gibt es Ersatz: Das *HttpModule*. Damit können Sie den Datenstrom – unabhängig von anderen Programmteilen – umleiten. Wie Sie diese Module programmieren und verwenden beschreibt Abschnitt 12.3, »HTTP-Module« ab Seite 1037.

12.2 HTTP-Handler

ASP.NET verwendet generell so genannte Handler zur Abarbeitung der per HTTP eintreffenden Anfragen. Das sind Programme, die sich innerhalb der Prozesskette für bestimmte Aktionen zuständig fühlen. Das gilt auch für die fest implementierte Laufzeitumgebung. Durch die Möglichkeit, solche Handler selbst zu programmieren, ist es leicht, ASP.NET zu erweitern.

Die Auswahl des für eine Anforderung benötigten Handlers erfolgt auf Basis des Dateinamens. Zulässig ist die Angabe von Platzhalterzeichen. Damit werden dann praktisch auch die von ASP.NET selbst verarbeiteten Dateiendungen bedient.

12.2.1 Vorhandene HTTP-Handler

Für die vorhandenen Handler ist die Konfiguration in der Datei *machine.config* verantwortlich. Der Abschnitt `<httpHandlers>` enthält die entsprechenden Angaben:


```
<configuration>
  <system.web>
    <httpHandlers>
      <add verb="*" path="trace.axd"
        type="System.Web.Handlers.TraceHandler"/>
      <add verb="*" path="*.aspx"
        type="System.Web.UI.PageHandlerFactory"/>
      <add verb="*" path="*.ashx"
        type="System.Web.UI.SimpleHandlerFactory"/>
      <add verb="*" path="*.asmx"
        type="System.Web.Services.Protocols.
          WebServiceHandlerFactory,
          System.Web.Services,
          Version=1.0.3300.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"
        validate="false"/>
      <add verb="*" path="*.rem"
        type="System.Runtime.Remoting.Channels.
          Http.HttpRemotingHandlerFactory,
          System.Runtime.Remoting,
          Version=1.0.3300.0, Culture=neutral,
          PublicKeyToken=b77a5c561934e089"
        validate="false"/>
      <add verb="*" path="*.soap"
        type="System.Runtime.Remoting.Channels.
          Http.HttpRemotingHandlerFactory,
          System.Runtime.Remoting,
          Version=1.0.3300.0, Culture=neutral,
          PublicKeyToken=b77a5c561934e089"
        validate="false"/>
      <add verb="*" path="*.asax"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.ascx"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.config"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.cs"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.csproj"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.vb"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.vbproj"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.webinfo"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.asp"
        type="System.Web.HttpForbiddenHandler"/>
      <add verb="*" path="*.licx"
```

```

        type="System.Web.HttpForbiddenHandler"/>
    <add verb="*" path="*.resx"
        type="System.Web.HttpForbiddenHandler"/>
    <add verb="*" path="*.resources"
        type="System.Web.HttpForbiddenHandler"/>
    <add verb="GET,HEAD" path="*"
        type="System.Web.StaticFileHandler"/>
    <add verb="*" path="*"
        type="System.Web.HttpMethodNotAllowedHandler"/>
</httpHandlers>
</system.web>
</configuration>

```

Die Attribute Der Abschnitt ist relativ leicht lesbar. Verknüpft wird eine Dateierweiterung (Attribut *path*) – optional auch ein konkreter Dateiname mit oder ohne Platzhalterzeichen – mit einer Klasse, die für die Verarbeitung zuständig ist (Attribut *type*). Zusätzlich kann noch mit dem Attribut *verb* bestimmt werden, welche HTTP-Kommandos akzeptiert werden. Außer dem Sternchen, das alle zulässt, können Sie hier GET, POST, DEBUG usw. einsetzen. Informationen zu HTTP finden Sie auch in Abschnitt 1.3.1, »Das Protokoll HTTP« ab Seite 51.

Bemerkenswert ist, dass auch die Abarbeitung von normalen *aspx*-Seiten lediglich durch einen der Standard-Handler erfolgt – dem Page-Handler. Im Grunde ist der modulare Aufbau also nicht nur ein gutes Entwurfswerkzeug, sondern Basis des gesamten Unterbaus von ASP.NET. Es spricht deshalb vieles dafür, dass die Architektur auf lange Sicht leistungsfähig genug ist, auch große Änderungen am Bedarf der Softwareentwickler mitzumachen.

Verbotene Dateiendungen Aus dieser Liste erklärt sich auch, warum Dateien mit der Endung *.config* nicht verarbeitet werden können. Hier wird *System.Web.HttpForbiddenHandler* aufgerufen, eine Klasse, die den Zugriff verweigert und eine entsprechende Meldung ausgibt. Diese internen Handler sind allerdings nicht öffentlich implementiert, können also nicht verändert oder konfiguriert werden.

12.2.2 Erweiterung durch eigene Handler

Um nun eigene Handler verwenden zu können, muss dieser Handler erweitert werden. Selbstverständlich können Sie dies auch »pro Applikation« tun, indem der entsprechende Abschnitt der Datei *web.config* hinzugefügt wird. Normalerweise existiert

dieser Eintrag noch nicht; Sie können ihn dann unterhalb des Zweiges `<system.web>` anlegen.

Das genügt jedoch noch nicht, denn die Dateierweiterung muss auch im IIS verknüpft werden. Der IIS empfängt die Anforderung vom Browser immer als Erster und entscheidet nun anhand des angeforderten Dateityps, welche Applikation dafür zuständig ist. Die Verknüpfung der von ASP.NET verarbeiteten Typen wurde durch die Installation hergestellt. Andere Einträge müssen von Hand hinzugefügt werden.

12.2.3 Programmierung eines eigenen Handlers

Schaut man sich die Möglichkeiten an, die HTTP-Handler bieten, stellt man sich zuerst die Frage, wozu das zu gebrauchen ist. Generell gilt auch hier: Nicht alles was .NET bietet, müssen Sie auch verwenden. Es geht am Anfang mehr darum, einen Überblick zu erhalten, was es alles gibt und sich dann im Bedarfsfall daran zu erinnern. Sie machen also nichts falsch, wenn Sie mit HTTP-Handlern nichts anfangen können.

Nichtsdestotrotz gibt es natürlich Einsatzfälle. Vor den ersten Listings sollten Sie sich ein wenig mit dem Prinzip auseinander setzen.

Das Prinzip der HTTP-Handler-Programmierung

Ausgangspunkt eines eigenen Handlers ist wie immer eine Klasse. Damit Sie nichts falsch machen, muss die Klasse von der Schnittstelle `IHttpHandler` abgeleitet werden. Sie verlangt, dass eine Eigenschaft und eine Methode zu implementieren sind:

► `IsReusable`

Diese Eigenschaft vom Typ `Boolean` gibt an, ob jede Anfrage eine neue Instanz des Handlers erfordert oder nicht. Der Standardwert, den diese schreibgeschützte Eigenschaft zurückgibt, ist `True`. Normalerweise existiert also nur eine Instanz für alle Anfragen. Wenn Sie keinen triftigen Grund haben, das Verhalten zu ändern, belassen Sie es dabei.

► `ProcessRequest`

Diese Methode erledigt die eigentliche Arbeit. Als Parameter wird der Kontext der aktuellen Verbindung erwartet, Typ

HttpContext. Damit haben Sie Zugriff auf Request, Response usw. Zurückgegeben wird nichts.



Mehr Informationen zu Request, Response und Context finden Sie im Abschnitt 8.2, »Die Welt der Standardobjekte« ab Seite 747.

Nun muss ASP.NET noch mitgeteilt werden, dass Sie unter bestimmten Umständen die Verwendung des eigenen Handlers wünschen. Die »bestimmten Umstände« können anhand der Verwendung einer bestimmten Dateierweiterung oder auch eines konkreten Dateinamens erkannt werden.

Dateierweiterung

Die Erkennung einer Dateierweiterung setzt voraus, dass diese überhaupt bis zu ASP.NET vordringt. Dazu muss der IIS entsprechend instruiert werden. Gehen Sie dazu folgendermaßen vor:

1. Öffnen Sie die Managementkonsole INTERNET-INFORMATIONSDIENSTE. Wählen Sie die STANDARDWEBSITE oder eine andere Website Ihres Servers oder ein als Applikation konfiguriertes virtuelles Verzeichnis.
2. Starten Sie dann aus dem Kontextmenü den Dialog EIGENSCHAFTEN.
3. Auf der Registerkarte VERZEICHNIS klicken Sie auf KONFIGURATION. Falls die Schaltfläche inaktiv ist, müssen Sie erst eine Anwendung erstellen. Besser ist, es eine von Visual Studio .NET erzeugte Anwendung zu verwenden.
4. Sie gelangen nun in den Dialog ANWENDUNGSKONFIGURATION. In der Liste ANWENDUNGSZUORDNUNGEN sehen Sie, wie beispielsweise die Dateierweiterung *.aspx* mit ASP.NET – konkret mit der ISAPI-DLL *aspnet_isapi.dll* – verknüpft ist. Merken Sie sich den Pfad aus dem Eingabefeld AUSFÜHRBARE DATEI des Dialogs BEARBEITEN.
5. Erzeugen Sie nun mit HINZUFÜGEN einen neuen Eintrag. Tragen Sie in dem Eingabefeld AUSFÜHRBARE DATEI den Pfad zu ASP.NET ein und unter Erweiterung *.pix*. Der IIS leitet nun alle Dateien mit der Endung *.pix* an ASP.NET weiter. Deaktivieren Sie des Weiteren die Option PRÜFEN, OB DATEI EXISTIERT. Füllen Sie die übrigen Einstellungen so aus, wie in der folgenden Abbildung gezeigt:

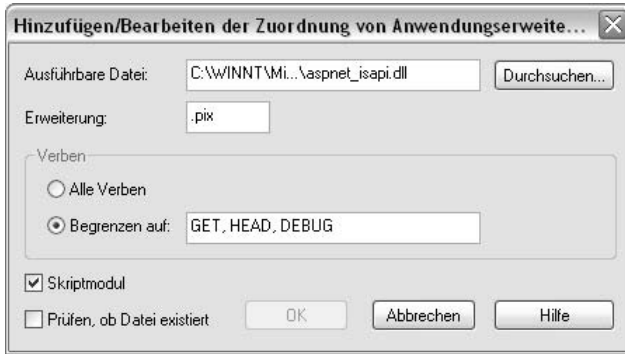


Abbildung 12.1: Dateierweiterung mit ASP.NET verbinden

Für Windows XP-Benutzer: Klicken Sie nach der Auswahl des Pfades nochmals in das Feld AUSFÜHRBARE DATEI, damit die OK-Schaltfläche aktiviert wird. Im .NET-Server und unter Windows 2000 ist das nicht erforderlich.



6. Speichern Sie alle Einstellungen.

Damit ist der IIS nun in der Lage, Anfragen mit der Erweiterung ».pix« zu erkennen und an ASP.NET weiterzuleiten. Freilich kann ASP.NET damit nichts anfangen, noch nicht. Die entsprechende Verknüpfung auf der »anderen Seite« wird in der Datei *web.config* vorgenommen.

Praktische Programmierung

Bevor jedoch der Handler in der Datei *web.config* verknüpft wird, muss eine Klasse erstellt werden, die die eigentliche Verarbeitung vornimmt. Für Standardaktionen existieren derartige Handler natürlich, die Verarbeitung von eigenen Erweiterungen müssen Sie selbst programmieren. Im Beispiel geht es um das dynamische Erzeugen von Bildern. Eine Einführung in dieses Thema wurde bereits im Abschnitt 11.2, »Bilder dynamisch erstellen« ab Seite 990 gebracht. Es geht nun darum, ein Bild dynamisch zu erstellen, indem als Bildpfad Folgendes verwendet wird:

```

```

Dabei kommt es nur auf die Dateierweiterung an, der Name ist beliebig und die aufgerufene Datei muss nicht existieren.

**Bilder dynamisch
per Handler
erzeugen**



Beachten Sie, dass in Abbildung 12.1 das Kontrollkästchen PRÜFEN, OB DATEI EXISTIERT, deaktiviert wurde. Ohne diese Maßnahme funktioniert das Verfahren mit der nicht existenten Datei nicht.

Im Projekt *vbnetadvanced* ist dieses Prinzip im Beispiel *PictureHandler.aspx* realisiert worden:

```
<body MS_POSITIONING="GridLayout">
<h1>Dynamische Bilder, mit Handler erzeugt</h1>

<br/>
<br/>
<br/>
<br/>
</body>
```

Listing 12.1: Der Aufruf der dynamisch erzeugten Bilder (*PictureHandler.aspx*)

Die Bedeutung der Parameter können Sie der Tabelle 11.1 entnehmen. Freilich steckt das Know-how woanders. Die Klasse *PictureClass* finden Sie in Listing 11.6 im Abschnitt 11.2.3, »Eine kleine Grafikbibliothek« ab Seite 998. Der Handler nutzt diese Klasse zur Bilderzeugung. Er selbst wird als allein stehende Klassendatei programmiert. Diese erzeugen Sie in Visual Studio .NET folgendermaßen:

- ▶ Wählen Sie im Kontextmenü des Projekts HINZUFÜGEN | KLASSE HINZUFÜGEN.
- ▶ Vergeben Sie einen passenden Namen und achten Sie auf die Auswahl der richtigen Projektkategorie (VB-Projekte).
- ▶ Ändern Sie das Klassentemplate entsprechend dem folgendem Beispiel *PictureHandler.vb*:

```
Public Class HandlePics
    Implements IHttpHandler

    Public Sub ProcessRequest(ByVal context As HttpContext) _
        Implements IHttpHandler.ProcessRequest
        Dim pc As PictureClass
        pc.Context = context
        pc.ProcessPicture()
    End Sub

    Public ReadOnly Property IsReusable() As Boolean _
        Implements IHttpHandler.IsReusable
```

```

        Get
            Return True
        End Get
    End Property
End Class

```

Listing 12.2: Der komplette Handler zum dynamischen Erzeugen von Bildern implementiert die Schnittstelle `IHttpHandler` (`PictureHandler.vb`).

Auf die Aspekte der Grafikprogrammierung soll hier nicht erneut eingegangen werden. Interessanter ist das Prinzip des Handlers, das auf der Nutzung der Schnittstelle `IHttpHandler` basiert.

Wie es funktioniert

Zugriff auf den Datenstrom von und zum Browser erhalten Sie über den Parameter vom Typ `HttpContext`. Dieser wird an die Eigenschaft `context` der Zielklassen weitergeleitet. Die Entnahme der Objekte `Request` und `Response` ist nicht zwingend erforderlich, wird aber für Ausgaben im Anwendungsfall benötigt. Sollte Ihr Handler nur eine Datenbank bedienen, können Sie darauf verzichten. Bleibt noch die Eigenschaft `IsReusable`. Gibt diese `True` zurück, wird nur eine Instanz der Klasse angelegt und immer wieder verwendet. Das ist schneller, muss aber bei der Programmierung beachtet werden, denn so wird der Konstruktor nur einmal aufgerufen. Wenn Ihr Handler verschiedene Sitzungen bedienen soll und die Abfrage der Sitzungsdaten im Konstruktor erfolgt, wird das nur funktionieren, wenn `IsReusable` den Wert `False` zurückgibt.

IHttpHandler

Das eigentliche Grafikprogramm wird in Abschnitt 11.2.3, »Eine kleine Grafikbibliothek« ab Seite 998 genauer diskutiert.

Den Handler in `web.config` registrieren

Wenn Sie das Programm jetzt übersetzen, wird es noch nicht funktionieren. Denn die Dateierweiterung `.pix` gelangt zwar zu ASP.NET, wird aber nicht weiter beachtet. Sie müssen noch die Konfiguration in `web.config` anpassen.

Wenn Sie Zugriff auf den Server haben, können Sie die Einstellung auch für alle Projekte in `machine.config` vornehmen.



Tragen Sie in `web.config` folgendes im Zweig `<system.web>` ein:

```

<configuration>
  <system.web>
    <httpHandlers>

```

```
<add verb="GET" path="*.pix"
      type="Addison.VBNet.Advanced.HandlePics,
vbnetadvanced" />
</httpHandlers>
```

Der Handler soll nur auf HTTP-GET reagieren – dies wird durch das Attribut `verb` bestimmt. Dann wird in `path` die Dateierweiterung verknüpft. Sie können hier auch »bild.pix« schreiben, dann ist die Wahl des Dateinamens nicht mehr frei. Bleibt als Letztes noch das Attribut `type`. Hier ist der Name der Klasse mit vollständigem Namensraum und, durch Komma getrennt, der Name der Assembly anzugeben.



Den Assemblynamen können Sie den Eigenschaften des Projekts entnehmen:

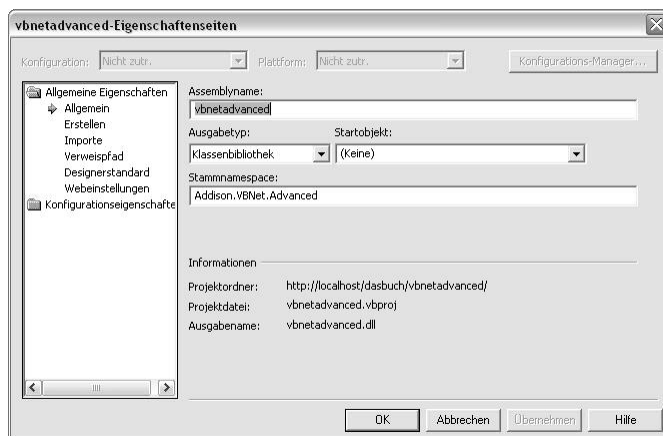


Abbildung 12.2: Der Name der Assembly steht im Feld ASSEMBLYNAME.

Danach steht einem Test nichts mehr im Wege. Das am Anfang gezeigte Beispielprogramm erzeugt folgende Ausgabe:



Abbildung 12.3: Über HTTP-Handler dynamisch erzeugte Bilder

12.2.4 HTTP-Handler ohne IIS-Konfiguration verwenden

Wenn Sie nun denken, dass Sie das Verfahren auch im Webpace, beim Provider verwenden können, werden Sie schnell an eine Grenze stoßen. Denn der Provider wird vermutlich nicht zulassen, dass Sie die IIS-Konfiguration ändern. Überdies ist das auch ein zusätzlicher Schritt, den man sich ersparen kann. Wie zuvor bereits angedeutet, trägt das Attribut `path` des Elements `<httpHandlers>` nicht nur eine Dateierweiterung, sondern auch einen vollständigen Dateinamen. Wenn Sie für diesem eine bereits vorhandene Dateierweiterung verwenden, ist eine Änderung an der IIS-Konfiguration nicht erforderlich.

Erstellen Sie in `web.config` beispielsweise folgendes zusätzliche Element:

```
<add verb="GET" path="pixx.aspx" ⚡  
      type="Addison.VBNet.Advanced.HandlePics, vbnetadvanced"/>
```

Ändern Sie nun den Aufruf in der `aspx`-Datei folgendermaßen:

```
  
...  
<br/>
```

Listing 12.3: Aufruf des Handlers mit einem vollständigen Dateinamen und bekannter Erweiterung (Ausschnitt aus `PictureHandlerNollS.aspx`)

Schon funktioniert der HTTP-Handler wieder und der IIS ist aus dem Rennen. Das über verschiedene Wege ein und derselbe Handler verwendet wird, ist völlig in Ordnung, denn auch die standardmäßigen Handler werden vielfältig genutzt.

12.3 HTTP-Module

HTTP-Module sind nicht an bestimmte Aufrufe oder die Verknüpfung mit einer Dateierweiterung gebunden. Sie agieren ähnlich wie die früheren ISAPI-Filter und beeinflussen den gesamten Ein- und Ausgabedatenstrom. Das ist dann interessant, wenn Sie globale Verhaltensweisen wünschen, aber nicht in bestehende Applikationen eingreifen möchten.

12.3.1 Grundsätzliche Funktionsweise

Ereignisorientiert HTTP-Module funktionieren etwas anders als Handler. Auch hier wird zwar eine Klasse erstellt und in *web.config* verknüpft, aber es erfolgt kein direkter Zugriff auf *Request* oder *Response*. Stattdessen können Sie für alle Applikationsereignisse Ereignisbehandlungsmethoden schreiben und damit den Zugriffspunkt exakt definieren. Eine Übersicht über diese Ereignisse und deren Bedeutung finden Sie in Abschnitt 8.6.1, »Einführung in das Applikationsereignismodell« ab Seite 802.

Realisiert wird ein Modul, indem Sie eine Klasse schreiben, die von der Schnittstelle *IHttpModule* ableitet. Sie müssen hier zwei Methoden implementieren:

- ▶ *Sub Init(ha As HttpApplication)*
Diese Methode initialisiert die Ereignisbehandlungsmethoden. Um den Zugriff auf das aktuelle *Application*-Objekt zu ermöglichen, wird es als Parameter übergeben.
- ▶ *Sub Dispose()*
Falls Aufräumvorgänge notwendig sind, platzieren Sie diese hier. Normalerweise bleibt diese Methode jedoch leer. Parameter werden nicht übergeben.

Die folgende Tabelle zeigt die Reihenfolge, in der die Ereignisse abgearbeitet werden:

Ereignis	Aktion
<i>BeginRequest</i>	Start der Anforderung
<i>AuthenticateRequest</i>	Authentifizierung (Benutzererkennung)
<i>AuthorizeRequest</i>	Autorisierung (Rechtvergabe)
<i>ResolveRequestCache</i>	Cache prüfen
	HTTP-Handler instanziiieren (die für die Seite (<i>Page</i>), aber auch eigene)
<i>AcquireRequestState</i>	Status prüfen
<i>PreRequestHandlerExecute</i>	Vor der Anforderungsverarbeitung
	HTTP-Handler ausführen
<i>PostRequestHandlerExecute</i>	Nach der Ausführung, Seite fertig
<i>ReleaseRequestState</i>	Freigabe der Ressourcen
	Aufruf von Ausgabefiltern (siehe <i>Response.Filter</i>)

Ereignis	Aktion
UpdateRequestCache	Cache aktualisieren
EndRequest	Anforderung fertig, Daten senden

HTTP-Module sind also auch im Hinblick auf die Prozessverarbeitung sehr leistungsfähig. Sie können nämlich ihre Leistungen zu jedem Zeitpunkt der Prozessverarbeitungskette anbieten. Vergleichen Sie aber in diesem Zusammenhang auch die Möglichkeiten, die Eingriffe in die Datei *global.asax* bieten. Mehr dazu finden Sie im Abschnitt 8.6.2, »Die Datei *global.asax*« ab Seite 808.

HTTP-Module und *global.asax*

Registrierung des Moduls in der Datei *web.config*

Ähnlich wie bei den HTTP-Handlern muss auch hier eine Registrierung in *web.config* erfolgen. Kennen sollten Sie den Namensraum, den Namen der Klasse und den der Assembly, um das Element `<httpModules>` entsprechend zu konfigurieren:

```
<configuration>
  <system.web>
    <httpModules>
      <add type="Addison.VBNet.Advanced.CopyrightModule, ↵
        vbnetadvanced" ↵
        name="CopyrightModule"/>
    </httpModules>
```

Das Attribut `type` nimmt den Namen der Klasse auf und – durch Komma getrennt – den Namen der Assembly. Abbildung 12.2 zeigte bereits, wie Sie den Assemblynamen ermitteln können. In `name` wird außerdem noch ein Name des Moduls vergeben. Empfehlenswert ist es, den Klassennamen zu verwenden.

12.3.2 Anwendungsbeispiel

Als Beispiel soll hier gezeigt werden, wie Sie verdeckte Copyright-Informationen in alle gesendeten Seiten einfügen können:

```
Imports System
Imports System.Web

Public Class CopyrightModule
  Implements IHttpModule
  Private copyrightBefore As String ↵
```

```

        = "<!-- (c) 2003 by Joerg Krause, " &
            Uwe Buenning -->" + vbCrLf
    Private copyrightAfter As String
        = vbCrLf + "<!-- " &
            + System.DateTime.Now.ToLongDateString() + " "

    Private Sub SendCopyrightBefore(ByVal sender As Object,
                                    ByVal e As EventArgs)
        Dim ha As HttpApplication = CType(sender,
                                            HttpApplication)
        ha.Context.Response.Write(copyrightBefore)
    End Sub

    Private Sub SendCopyrightAfter(ByVal sender As Object,
                                    ByVal e As EventArgs)
        Dim ha As HttpApplication = CType(sender,
                                            HttpApplication)
        ha.Context.Response.Write(copyrightAfter)
    End Sub

    Public Sub Init(ByVal app As HttpApplication) _
        Implements IHttpModule.Init
        AddHandler app.BeginRequest,
            New EventHandler(AddressOf SendCopyrightBefore)
        AddHandler app.EndRequest,
            New EventHandler(AddressOf SendCopyrightAfter)
    End Sub

    Public Sub Dispose() _
        Implements IHttpModule.Dispose
    End Sub

```

End Class

Listing 12.4: Ein HTTP-Modul (CopyrightModule.vb)

Wie es funktioniert

Aufgabe dieses Programm ist es, oberhalb und unterhalb jeder gesendeten Seite einen HTML-Kommentar einzufügen. Dazu wird in der Methode `Init` zuerst eine Ereignisbehandlung für den Anfang der Anforderung definiert:

```

AddHandler app.BeginRequest,
    New EventHandler(AddressOf SendCopyrightBefore)

```

Dann noch eine für das Ende:

```

AddHandler app.EndRequest,
    New EventHandler(AddressOf SendCopyrightAfter)

```

Die Methode *CopyrightBefore* greift nun auf das Application-Objekt zu:

```
Dim ha As HttpApplication = CType(sender, HttpApplication)
```

Sie gibt dann mit *Response.Write* den gewünschten Text aus:

```
ha.Context.Response.Write(copyrightAfter)
```

Am Ende der Anforderung funktioniert das vergleichbar. Ein Blick in eine beliebige Seite des Projekts verrät, dass es funktioniert hat:



Abbildung 12.4: Die Wirkung des Moduls im Quelltext einer Seite

HTTP-Module sind ein sicheres Mittel, allgemeine Vorgänge, die unabhängig vom Benutzer sind, zu genau definierten Zeitpunkten auszuführen. Dazu gehören beispielsweise auch eigene Implementierungen für eine Authentifizierung oder die Steuerung der Daten des Ein- oder Ausgabestromes.

13 Webservices

Webservices sind für die meisten Entwickler erst seit kurzem ein Thema. Während Microsoft sie hartnäckig als Technologie der Zukunft propagiert, bleiben Anwendungen rar. Dies könnte sich ändern, denn mit .NET ist es sehr einfach, auch komplexe Webservices zu programmieren. Dies gilt für beide Varianten, die Nutzung eines fremden Webservices ebenso wie das Anbieten eines eigenen.

13.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

13.1.1 Über dieses Kapitel

Die Protagonisten einer schillernden Zukunft der Softwarezunft reden seit Monaten über nichts anderes als Webservices. An vernünftigen Anwendungen mangelte es jedoch bisher. Damit einher geht manches Unverständnis der ohnehin von ständig neuen Dingen geplagten Entwickler, sich nun auch noch mit einem Dienst auseinander setzen zu sollen, der in der täglichen Praxis so recht keinen Platz zu haben scheint. Tatsächlich scheitert der Einsatz aber nicht an der Machbarkeit, sondern am klassischen Henne-Ei-Problem. Denn Webservices sind – wie der Name schon suggeriert – Dienste. Dienste kann man nutzen, wenn sie von jemandem angeboten werden. Man kann sie auch anbieten, wenn man Nutzer dafür findet. Offensichtlich warten hier die einen auf die anderen.

Dabei ist es nicht so schwer, sich Dienste zu überlegen und der verhältnismäßig geringe Aufwand, einen Webservice zu erstellen, dürfte die Hemmschwelle sehr niedrig legen. In diesem Kapitel werden zwei Varianten der Webservices vorgestellt. Einmal die Konsumierung eines fremden Dienstes am Beispiel der Suchmaschine Google. Dies erfolgt einmal unter Einsatz des Werkzeugs *wsdl.exe*, mit dem aus der den Dienst beschreibenden WSDL-Datei eine Proxyklasse erstellt werden kann. Die andere Variante zeigt, wie die WSDL-Informationen programmtechnisch extrahiert und verwertet werden ohne dass *wsdl.exe* benötigt wird.

Zum anderen wird das Anbieten eines eigenen Webservices behandelt. Mögliche Einsatzfälle sind Partnerprogramme oder einfach praktische Dienstleistungen für andere Serveranwendungen.

Zu Beginn werden einige grundlegende Informationen über die zugrunde liegenden Technologien beschrieben und ein Ausblick auf UDDI gegeben.

13.2 Grundlagen

Ein Grundverständnis für Webservices erhält man, wenn die Kommunikationswege geklärt sind. So findet die Kommunikation zwischen dem Anbieter des Webservice und dem Client (hier Consumer genannt) auf Serverebene statt. Es geht also nicht unbedingt darum, Benutzern einen Webservice zur Verfügung zu stellen, sondern bestenfalls darum, diesen die Leistungen mit einem Webinterface zugänglich zu machen. Aber auch das dürfte der Ausnahmefall sein.

Die Protokolle

Wenn Server miteinander reden können – ohne menschliches Zutun – ist das ein wenig Science Fiction und in Zukunft ein starkes Geschäft. Denn die Kommunikation wird durch die zugrunde liegenden Technologien, XML, SOAP, HTTP, WSDL usw. so billig und einfach, dass sie in der Breite zur Anwendung gelangen können. Das ist der Unterschied zu früher, wo es Serverkommunikation auch schon gab, diese sich aber auf wenige Anwendungen beschränkte. Denkbar sind vor allem Partnernetzwerke, von der Bannerschaltung über einen gemeinsamen Ad-Server bis hin zu den beliebten Webringen. Egal, ob Sie Artikel verkaufen und Subunternehmer suchen oder als Sammelshop auf mehrere Lieferanten zugreifen möchten, Webservices sind die ideale Kommunikations-

lösung. Es gibt aber weitergehende Leistungen. So könnten Websites einen Aktualisierungsdienst anbieten, der auf Anfrage (eines anderen Servers) alle geänderten, gelöschten oder hinzugefügten Links zu einem vereinbarten Thema liefert. Eine intelligente Linkliste reorganisiert sich dann selbstständig. Auf eben diesem Wege lassen sich auch Nachrichten des Tages oder Sportergebnisse liefern.

Der Vorteil all dieser Vorgänge basiert auf den genannten Technologien, die eine nähere Betrachtung Wert sind.

Die hier beschriebenen Grundlagen zu den verwendeten Protokollen sind stark vereinfacht und stellen nur den Teil dar, der zum Verständnis der vorgestellten Programme notwendig ist. Tatsächlich sind alle Spezifikationen außerordentlich umfangreich. Es ist zu allen Themen eine Reihe von Spezialliteratur verfügbar.



13.2.1 Die Protokolle

Mit XML, SOAP, HTTP und WSDL stehen gleich mehrere Protokolle, Normen und Verfahren bereit, Entwicklern und Anwenden den Weg zu Webservices zu erleichtern. Die Vielfalt klingt kompliziert, intern betrachtet sind es jedoch nur noch zwei Protokolle, die wichtig sind: XML und HTTP. Für Webprogrammierer dürften beides alte Bekannte sein. SOAP und WSDL sind XML und dass als Transportprotokoll HTTP zum Einsatz kommt, erleichtert die Programmierung ungemein.

Abstraktion ↑	Ausführung	.NET-Klasse
	Funktionsaufruf	SOAP
	Transport	HTTP, SMTP

Abbildung 13.1: Ablauf des entfernten Funktionsaufrufes aus Sicht der Protokollschichten

Auch wenn hier nicht weiter darauf eingegangen wird, sollten Sie wissen, dass SOAP nicht zwingend mit HTTP verknüpft ist. Als Transportprotokolle können auch HTTPS (verschlüsseltes HTTP) und SMTP (E-Mail) auftreten. Prinzipiell schreibt der Standard überhaupt kein Protokoll vor. Es muss nur in der Lage sein, XML-Daten zu übertragen.



Mehr Informationen zu XML-Protokollen, kurz als XP bezeichnet, finden Sie unter folgender Adresse: <http://www.w3.org/2000/03/29-XML-protocol-matrix>.

SOAP – Simple Object Access Protocol

Als Geheimnis der Webservices wird an erster Stelle SOAP (Simple Object Access Protocol) angeführt. SOAP definiert ein XML-Protokoll, das im Körper einer HTTP-Nachricht versendet wird. Das ist der eigentliche Knüller, denn HTTP geht durch vermutlich jede Firewall der Welt und ist damit der ideale Weg, wirklich jeden Server in das Netzwerk der Webservice-Teilnehmer einzubinden. Ganz nebenbei ist XML der zweite Knüller. Denn die in allen Plattformen und Sprachen verfügbaren Parser können dieses Format leicht lesen und die Daten so einfach zur Verfügung stellen, dass nur geringer Programmieraufwand anfällt. Der Prozess der Datenannahme und auch der Datenerzeugung lässt sich gut automatisieren. Das muss auch so sein, denn ohne eine automatische Weiterverarbeitung ist eine automatische Kommunikation sinnlos.

SOAP ist beim W3C standardisiert. Die Spezifikation der aktuellen Version 1.1 finden Sie unter folgender Adresse: <http://www.w3.org/TR/SOAP/>.

Die folgende Abbildung zeigt den prinzipiellen Aufbau einer SOAP-Nachricht. Die Anzahl der Blöcke mit den Daten der Anfrage oder der Antwort ist beliebig.

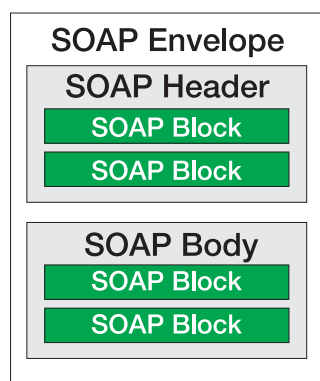


Abbildung 13.2: Prinzipieller Aufbau einer SOAP-Nachricht

Da es sich bei SOAP um XML handelt, gibt es eine gut lesbare Darstellung des prinzipiellen Aufbaus der Nachrichten:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    :      :      :
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    :      :      :
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Hier wird bereits deutlich, dass als zusätzlicher Namensraum unter anderem »xsd« verwendet wird, die XML Schema Definition Language (XSD). Die dort definierten Datentypen bilden damit das Rückgrat der typstrengen Übermittlung von Daten zwischen den Kommunikationspartnern.

Datentypen

Das Wurzelement bildet der so genannte SOAP-Envelope, auf deutsch der Umschlag. Der URI des Namensraumes lautet immer wie folgt:

```
http://schemas.xmlsoap.org/soap/envelope/
```

Der Präfix ist frei wählbar, wird jedoch standardmäßig auf SOAP-ENV gesetzt. Die beiden Namensräume für die Schema-Definition und Schema-Instanz sollten hinzugefügt werden und müssen vorhanden sein, wenn die entsprechenden Datentypen verwendet werden. Schema-Instanz definiert komplexe Typen und Null als *nil* in dieser Definition.

Der Kopf (SOAP-Header) der Nachricht ist optional, der Körper (SOAP-Body) muss immer vorhanden sein. DTD-Deklarationen als Alternative zu XSD sind grundsätzlich nicht zulässig. Der SOAP-Body enthält das eigentliche Dokument, das für sich genommen wohlgeformt nach den XML-Regeln sein muss. Sonderzeichen oder binäre Daten sind durch <[CDATA[...]]>-Abschnitte zu schützen. Das ist schon die einzige Regel, SOAP selbst beachtet den Inhalt der Nachricht nicht weiter. Es ist Sache der beteiligten Prozessoren, die Daten auszuwerten, mithin also Ihres eigenen Programmes.

SOAP-Nachricht Abschließend ein Blick auf eine vollständige SOAP-Nachricht:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
      <key xsi:type="xsd:string">00000000...0000000000</key>
      <q xsi:type="xsd:string">shrdlu winograd maclisp
teletype</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      <filter xsi:type="xsd:boolean">true</filter>
      <restrict xsi:type="xsd:string"></restrict>
      <safeSearch xsi:type="xsd:boolean">false</safeSearch>
      <lr xsi:type="xsd:string"></lr>
      <ie xsi:type="xsd:string">latin1</ie>
      <oe xsi:type="xsd:string">latin1</oe>
    </ns1:doGoogleSearch>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Listing 13.1: Eine SOAP-Nachricht, hier eine Anfrage an den Suchdienst Google (aus dem Google-API-Paket)

Antworten Antworten werden ebenso in SOAP formuliert. Das folgende Beispiel zeigt die Antwort auf die zuvor gezeigte Anfrage, ebenfalls dem Google-API-Paket entnommen:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ↵
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" ↵
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" ↵
      SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="ns1:GoogleSearchResult">
        <documentFiltering xsi:type="xsd:boolean">
          false</documentFiltering>
        <estimatedTotalResultsCount xsi:type="xsd:int">
          3</estimatedTotalResultsCount>
        <directoryCategories ↵
xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/" ↵
          xsi:type="ns2:Array" v
          ns2:arrayType="ns1:DirectoryCategory[0]">
          </directoryCategories>

```

```

    <searchTime xsi:type="xsd:double">
      0.194871</searchTime>
    <resultElements
      xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns3:Array"
      ns3:arrayType="ns1:ResultElement[3]">
      <item xsi:type="ns1:ResultElement">
        <cachedSize xsi:type="xsd:string"> 12k</cachedSize>
        <hostName xsi:type="xsd:string"></hostName>
        <snippet xsi:type="xsd:string">
          ... Hier steht der Antworttext dieser Abfrage ...
        </snippet>
        <directoryCategory
          xsi:type="ns1:DirectoryCategory">
            <specialEncoding xsi:type="xsd:string">
            </specialEncoding>
            <fullViewableName xsi:type="xsd:string">
            </fullViewableName>
          </directoryCategory>
        <relatedInformationPresent
          xsi:type="xsd:boolean">
            true
          </relatedInformationPresent>
        <directoryTitle
          xsi:type="xsd:string"></directoryTitle>
        <summary xsi:type="xsd:string"></summary>
        <URL xsi:type="xsd:string">
          ... http://die_url_der_suchanfrage ...
        </URL>
        <title xsi:type="xsd:string">...Titel...</title>
      </item>
    </resultElements>
    <endIndex xsi:type="xsd:int">3</endIndex>
    <searchTips xsi:type="xsd:string"></searchTips>
    <searchComments xsi:type="xsd:string">
    </searchComments>
    <startIndex xsi:type="xsd:int">1</startIndex>
    <estimateIsExact xsi:type="xsd:boolean">
      true</estimateIsExact>
    <searchQuery xsi:type="xsd:string">
      shrdlu winograd macclisp teletype</searchQuery>
  </return>
</ns1:doGoogleSearchResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Listing 13.2: Eine SOAP-Antwort, die *<item>-Elemente* würden sich in der Praxis wiederholen.

SOAP-Praxis Nun erscheint der Umgang mit SOAP nicht besonders einfach. Trotz leistungsfähiger XML-Funktionen in .NET wäre einiger Programmieraufwand nötig, die Nachrichten in jeder Situation exakt zu formulieren und die Antwort entsprechend auszuwerten. Vor allem dann, wenn der Anbieter die Struktur seines Dienstes ändert, wird es problematisch, denn dann muss das eigene Programm geändert oder schlimmstenfalls neu geschrieben werden. Da sich Anbieter und Konsument eines Webservices nicht kennen müssen und vermutlich oft geografisch weit auseinander sitzen, sind weitere Schritte hin zur praktischen Nutzbarkeit notwendig.

Das Problem wird noch verschärft, wenn Anbieter und Konsument mit verschiedenen Systemen arbeiten, beispielsweise auf der Serverseite mit Java und auf der Clientseite mit .NET. Abhilfe schafft die Sprache WSDL.

WSDL – Web Services Description Language

Als Nächstes soll die Beschreibungssprache WSDL (*Web Services Description Language*) näher untersucht werden. Mit WSDL wird die Struktur eines Webservice beschrieben, also die verfügbaren Funktionen, deren Parameter und die Art und Weise der Anforderungsgestaltung. Damit ist auch der Dienst selbst automatisierbar, denn der anfragende Server kann seine Anfrage entsprechend dem aktuellen Entwicklungsstand des Dienstes modifizieren. Insofern sind die Kommunikationswege über eine gewisse Zeit hinweg änderungsresistent oder wenigstens wartungsfreundlich. Das Geheimnis der Webservices ist also weit mehr als nur SOAP; und es ist auch etwas komplizierter. Die gesamte Kommunikation kann nämlich auch die Suche nach einem Dienst selbst einschließen.

WSDL ist ebenso wie SOAP beim W3C, derzeit als Version 1.1, standardisiert: <http://www.w3.org/TR/wsdl>.

Auch WSDL basiert auf XML. Die prinzipielle Struktur und die Abhängigkeiten der Elemente zeigt in vereinfachter Form die Abbildung 13.3.

Der Anbieter eines Dienstes definiert also zuerst ein Element `<service>`, das die Basisdienste deklariert. Erreichbar sind diese jeweils über Ports (`<portType>`), die Bindungen (`<binding>`) beschreiben die geforderte Kodierung des Datenstromes. Auf die Portdefinition folgt die Beschreibung einer oder mehrerer Operationen, die jeweils eine Ein- und Ausgabenachricht (`<Message>`) definieren.

Die Nachrichten können mehrere Parameter aufnehmen, die auf nativen XSD-Datentypen basieren oder denen komplexe Definitionen zugrunde liegen. Ist das der Fall, kommt noch der Abschnitt `<types>` zur Anwendung, wo XSD-Typdefinitionen untergebracht sind.

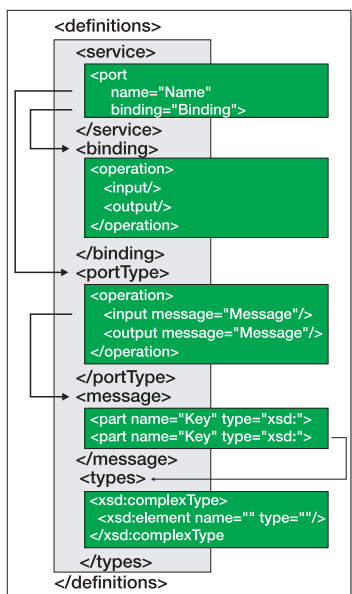


Abbildung 13.3: Prinzipieller Aufbau einer WSDL-Datei

```

<definitions name="urn:GoogleSearch"
  targetNamespace="urn:GoogleSearch"
  xmlns:typens="urn:GoogleSearch"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types> </types>

  <message name="MessageName">
    <part name="parameter" type="xsd:string"/>
    <part name="parameter" type="xsd:string"/>
  </message>

  <portType name="PortName">
    <operation name="Operation">

```

```

        <input message="typens:TypDefinition"/>
        <output message="typens:TypDefinition"/>
    </operation>
</portType>

<binding name="BindingName" type="typens:PortName">
    <soap:binding style="rpc"

transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="OperationName">
        <soap:operation soapAction="urn:ActionName"/>
        <input>
            <soap:body use="encoded"
                namespace="urn:ActionName"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </input>
        <output>
            <soap:body use="encoded"
                namespace="urn:ActionName"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
    </operation>
</binding>

<service name="ServiceName">
    <port name="PortName" binding="typens:BindingName">
        <soap:address location="http://url_des_dienstes.de"/>
    </port>
</service>

</definitions>

```

Listing 13.3: WSDL-Datei des Google-Webservice

Diese Struktur lässt sich gut in einer Klasse abbilden, die dann im Programm verwendet wird. Eine solche Klasse wird als Proxy-Klasse bezeichnet.

UDDI – Universal Description, Discovery and Integration

Als Letztes bleibt noch der Weg zu klären, auf dem Sie an die WSDL-Dateien gelangen. Es gibt hier drei Möglichkeiten:

1. Sie haben die Adresse der WSDL-Datei von Ihrem Geschäftspartner erhalten.

2. Sie besuchen ein öffentliches Verzeichnis von Webdiensten. Dazu finden Sie mehr Informationen im Abschnitt 13.3.1, »Hinweise zu öffentlichen Diensten« ab Seite 1054.
3. Sie verwenden UDDI. Das Zentrum der UDDI-Welt ist <http://www.uddi.org>.

Mehr als 80 führende Technologie- und Business-Unternehmungen haben sich zusammengerauft und das Projekt UDDI in Gang gesetzt. UDDI-Server (Universal Description, Discovery and Integration) werden verwendet, um den URL vermutlich passender Dienste zu finden. Anhand einer von diesen gesendeten Discovery-Datei (.disco) kann der eigene Server herausfinden, welcher Dienst am besten passt und wo die WSDL-Datei zu finden ist. Die Discovery-Datei informiert nicht wie WSDL über technische Parameter, sondern beispielsweise über den Preis und die Konditionen, unter denen der Dienst genutzt werden darf.

Das Prinzip der Discovery-Dateien (DISCO) ist eine Erfindung von Microsoft. Alternativ dazu hat IBM das ADS (Advertisement and Discovery Protocol) entwickelt, das vergleichbare Aufgaben übernimmt. Vermutlich wird sich am Ende eine Mischung aus beiden durchsetzen, wie es schon bei WSDL der Fall war, das auch aus zwei konkurrierenden Standards entstand.



Vorerst gibt es aber zuwenig Dienste, um derartigen Verzeichnissen etwas abgewinnen zu können.

Der gesamte Ablauf, insbesondere die hinter UDDI stehenden Protokolle, sind sehr komplex. Auch wenn sie auf XML aufbauen, zeigt allein das Anforderungsprofil, dass die automatisierte Erkennung und Nutzung von Diensten nicht trivial werden kann. Denn letztlich »verhandelt« der Client mit mehreren Servern, um den für ihn optimalen Partner zu finden. Mangels ausreichend vorhandener Verhandlungspartner hat UDDI bis heute (Anfang 2003) noch keine wirklich praxisrelevante Verbreitung gefunden. Es entwickelt sich aber sehr rasch weiter und wird in .NET weit reichend unterstützt, sodass mit neuem Schwung bei den Anwendungen gerechnet werden darf.

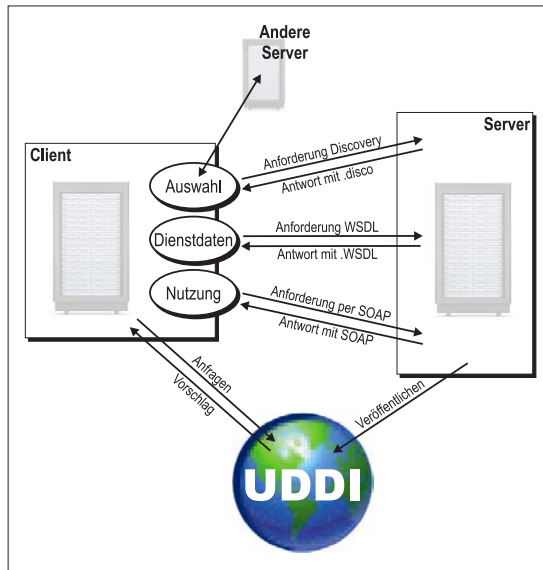


Abbildung 13.4: Nutzung von UDDI für die automatische Ermittlung von Webservices

13.3 Webservices konsumieren

Das Verwenden eines Webservices, allgemein als »Konsumieren« bezeichnet, setzt die Kenntnis eines URL voraus, unter welcher der Dienst erreicht werden kann, bzw. unter der weitere Informationen darüber zu finden sind.

13.3.1 Hinweise zu öffentlichen Diensten

Neben den beiden bekanntesten Angeboten – Google und Amazon – gibt es inzwischen viele kleinere Server, die Webservices anbieten. Ein guter Ausgangspunkt sind frei zugängliche Listen, die vor allem die ersten Schritte erleichtern. Im Gegensatz zu UDDI ist der Anspruch an die Anbieter, sich im Verzeichnis einzutragen, sehr viel geringer. Auch deshalb sind diese Listen beliebt und für Experimente gut geeignet. Hier eine Auswahl:

- SANTRA Technology's iON service
<http://www.mysantra.com>
- XMethods' directory of Web services
<http://www.xmethods.com>

► Sal Services Web services brokerage

<http://www.salcentral.com>

Das Interessante an SANTRA ist die Verfügbarkeit fertiger Proxy-Klassen für die angebotenen Dienste in C#, VB 6/COM und Java. Ebenso wichtig ist die Möglichkeit, einen eigenen Dienst kostenfrei anzumelden. Letzteres bieten alle drei Webseiten.

Die Auswahl selbst ist in allen Fällen recht bescheiden und reicht von Wetter- bis zu Umfragediensten, die die eigene Seite ergänzen können. Die Wahl für dieses Buch fiel auf Google, die als eine der größeren Anbieter vermutlich längere Zeit stabilen Betrieb garantieren können. Die Methodik ist aber exemplarisch auf alle anderen Dienste übertragbar.

Bevor Sie mit Google arbeiten, müssen Sie sich einen Lizenzschlüssel beschaffen. Derzeit kann Google bis 1.000 Mal pro Tag abgefragt werden. Pro Anfrage sind maximal zehn Antworten möglich. Die Nutzung ist kostenlos, Informationen über ein kommerzielles Modell, das eine intensivere oder kommerzielle Nutzung erlaubt, waren Anfang 2003 noch nicht verfügbar.



13.3.2 Ein einfaches Programm am Beispiel Google

Im Fall von Google lohnt zuerst ein Blick in die WSDL-Datei. Im API-Paket, das Sie auch auf der CD in der Fassung vom Juli 2002 finden, ist die Datei als *GoogleSearch.wsdl* zu finden. Es ist sinnvoll, immer die aktuelle Version zu verwenden. Ein Blick in *GoogleSearch.wsdl* zeigt eine Zusammenfassung der Dienste im Zweig `<portType>`, wo derzeit drei Elemente `<operation>` existieren, die für die Nachrichten die Ein- und Ausgabefunktionen definieren. Hier ein kleiner Ausschnitt:

WSDL analysieren

```
<portType name="GoogleSearchPort">
  <operation name="doGetCachedPage">
    <input message="typens:doGetCachedPage" />
    <output message="typens:doGetCachedPageResponse" />
  </operation>
  <!-- ... -->
</portType>
```

Die korrespondierenden Elemente `<input>` bzw. `<output>` enthalten wiederum die Anweisung, wie die Nachricht in Abfrage- bzw. Antwortrichtung aufzubauen ist. Für die im Beispielprogramm

verwendete Funktion *doGoogleSearch* sieht dies folgendermaßen aus:

```
<message name="doGoogleSearch">
  <part name="key"           type="xsd:string"/>
  <part name="q"             type="xsd:string"/>
  <part name="start"         type="xsd:int"/>
  <part name="maxResults"    type="xsd:int"/>
  <part name="filter"        type="xsd:boolean"/>
  <part name="restrict"      type="xsd:string"/>
  <part name="safeSearch"    type="xsd:boolean"/>
  <part name="lr"            type="xsd:string"/>
  <part name="ie"            type="xsd:string"/>
  <part name="oe"            type="xsd:string"/>
</message>
```

Definiert werden die Namen der Parameter und ihr Datentyp. Der Parameter *key* kommt in allen Abfragen vor und enthält den von Google übermittelten Lizenzschlüssel. Die Definition der Rückgabe erfolgt ähnlich, denn der Transport der Daten vom Server zum Client läuft ebenfalls via SOAP:

```
<message name="doGoogleSearchResponse">
  <part name="return" type="typens:GoogleSearchResult"/>
</message>
```

Der Attributwert `typens:GoogleSearchResult` deutet auf einen eigenen Datentyp hin, da die Rückgabewerte komplex sein können. Im Abschnitt `<types>` der WSDL-Datei sind diese Typen definiert. Clients können daraus die Schnittstelle ableiten. Dank der in WSDL verwendeten XSD-Datentypen ist die Schnittstelle typstreu. Es ist jedoch einiges Know-how nötig, um damit tatsächlich universelle SOAP-Clients zu erstellen. Glücklicherweise gibt es .NET, denn mit den in Visual Studio .NET und im SDK enthaltenen Werkzeugen können Sie die praktische Programmierung sehr entspannt angehen.

Die Google-Funktionen auf einen Blick

Bevor Sie richtig loslegen können, den Webservice zu nutzen, müssen Sie die bereitgestellten Funktionen auch kennen. Zuerst ein Blick auf die derzeit angebotenen Dienste:

Dienstname	Einsatzzweck
doGoogleSearch	Standardsuche, benötigt die in Tabelle 13.2 gezeigten Parameter
doSpellingSuggestion	Korrekturvorschlag für Schreibfehler anfordern. Parameter: <i>key</i> und <i>phrase</i>
doGetCachedPage	Suche im Google-Cache. Parameter: <i>key</i> und <i>url</i>

Tabelle 13.1: Derzeit verfügbare Funktionen des Webservices Google

Die Parameter der Standardsuchfunktion sind etwas umfangreicher:

Parameter	Bedeutung	Datentyp
<i>key</i>	Lizenzschlüssel	xsd:string, Zeichenkette
<i>q</i>	Die Abfrage	xsd:string, Zeichenkette
<i>start</i>	Start der Suche in der Ergebnisliste (beispielsweise gibt 30 die Ergebnisse 30 bis 35 aus, wenn <i>maxResults</i> 5 ist)	xsd:int, Ganzzahl
<i>maxResults</i>	Anzahl der Ergebnisse pro Seite (derzeit maximal zehn)	xsd:int, Ganzzahl
<i>filter</i>		xsd:boolean, Ja/Nein
<i>restrict</i>	Generelle Einschränkung	xsd:boolean, Ja/Nein
<i>safeSearch</i>	Unterdrückt jugendgefährdende Inhalte	xsd:boolean, Ja/Nein
<i>lr</i>	Sprache (<i>lr=Language Restriction</i>), beispielsweise "lang_de"	xsd:string, Zeichenkette

Tabelle 13.2: Parameter der Funktion *doGoogleSearch*

Im Falle von *GoogleSearch* beschreibt die WSDL-Datei lapidar einen Parameter »q« mit dem Datentyp »xsd:string«. Echte Freude kommt aber nur auf, wenn man »q« (steht für »query«) mit den passenden Suchzeichen füllen kann:

Suchform	Beschreibung
+wort	Wort muss vorkommen
-wort	Wort darf nicht vorkommen
"Eine Phrase"	Exakte Übereinstimmung

Tabelle 13.3: Die Suchoptionen für die Abfrage

Suchform	Beschreibung
Wort1 OR Wort2	Wort1 oder Wort2 müssen vorkommen. Die UND-Verknüpfung wird standardmäßig verwendet
site:	Die angegebene URL wird durchsucht

Tabelle 13.3: Die Suchoptionen für die Abfrage (Forts.)

Eine Proxy-Klasse erstellen

Wie in .NET üblich, brauchen Sie eine Klasse, die die benötigten Methoden zur Verfügung stellt. In der vorgestellten Applikation soll für eine Site eine Linkliste erzeugt werden. Google soll dazu auf Anfrage eine Liste von zehn Links liefern, die zum Thema der Seite passen. Da sich das Web schnell ändert, vertrauen wir der Suchmaschine die Aktualisierung der Links an. Die Abfrage selbst wird fest programmiert, lässt sich aber mit wenigen Handgriffen auf »Formularbetrieb« umstellen. Die Linkliste wird im Code erzeugt, die Webform selbst enthält lediglich ein Placeholder-Steuerelement:

```
<%@ Page language="vb" Codebehind="GoogleLinks.aspx.vb"
AutoEventWireup="false"
Inherits="Addison.VBNet.Advanced.GoogleLinks" %>

<html>
<head><title>GoogleLinks</title></head>
<body MS_POSITIONING="GridLayout">
    <h1>Unsere Linkliste</h1>
    <asp:Placeholder Runat="server" ID="LinkListe"/>
</body>
</html>
```

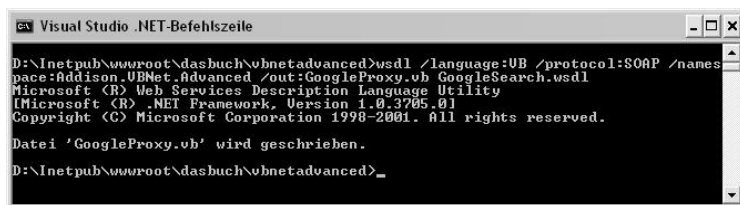
Listing 13.4: Die Vorlage für die Linkliste (GoogleLinks.aspx)

wsdl.exe

Damit die Abfrage der Daten per SOAP funktioniert, wird eine von `SoapHttpClientProtocol` abgeleitete Klasse benötigt. Diese wird als Proxy-Klasse bezeichnet, weil sie eine Art Schnittstelle zum Dienst darstellt. Damit das nicht zu kompliziert wird, befindet sich im Framework SDK das Werkzeug *wsdl.exe*. Verfügt der Dienst über eine WSDL-Datei, erstellt dieses Programm automatisch die Proxy-Klasse. Haben Sie die WSDL-Datei aus dem Google-API-Kit bereits auf der Festplatte, erstellen Sie die Klasse mit folgendem Aufruf auf der *Visual Studio .NET-Befehlszeile*:

```
wsdl /language:vb /protocol:SOAP
    /out:GoogleProxy.vb GoogleSearch.wsdl
```

Dieser Aufruf erzeugt die Proxy-Klasse in der Datei *GoogleProxy.vb* und verwendet den Namensraum *Addison.VBNet.Advanced*, der für alle Programme dieses Kapitels gilt. Für eine eigene Applikation ändern Sie die Angaben entsprechend. Außerdem setzt der Aufruf voraus, dass die WSDL-Datei *GoogleSearch.wsdl* im selben Verzeichnis liegt. Die Angabe der Sprache VB.NET dürfte eindeutig sein, ebenso wie das gewünschte Protokoll (wobei SOAP auch der Standardwert ist, die Angabe kann entfallen).



```
Visual Studio .NET-Befehlszeile
D:\Inetpub\wwwroot\dashbuch\vbnetadvanced>wsdl /language:VB /protocol:SOAP /names
pace:Addison.VBNet.Advanced /out:GoogleProxy.vb GoogleSearch.wsdl
Microsoft (R) Web Services Description Language Utility
[Microsoft (R) .NET Framework, Version 1.0.3705.0]
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.
Datei 'GoogleProxy.vb' wird geschrieben.
D:\Inetpub\wwwroot\dashbuch\vbnetadvanced>_
```

Abbildung 13.5: Die Proxy-Klasse wurde erfolgreich erstellt.

Wenn Sie mit Visual Studio .NET-Projekten und VB.NET arbeiten, wird normalerweise in den Standardeinstellungen des Projekts ein zentraler Namespace festgelegt. Wenn Sie nun die Klassen erstellen und einen Namensraum angeben und diese Klasse dann dem Projekt hinzufügen, »verdoppelt« sich quasi der reale Name des Namensraumes. Das funktioniert natürlich nicht. Lassen Sie deshalb wie beschrieben die Angabe des Namensraumes beim Aufruf des Tools *wsdl* weg, auch wenn andere Quellen dies so nicht beschreiben. C# verhält sich an dieser Stelle übrigens anders und die meisten Beispiele im Internet sind dafür ausgelegt. Dennoch können alle Funktionen auch mit VB.NET programmiert werden.

Umgang mit dem Namensraum

Webservices in Visual Studio .NET verfügbar machen

Standardmäßig fehlt der Verweis auf die Assembly, die die Klassen für Webservices enthält. Damit die Übersetzung des Programms gelingt, müssen Sie noch die Assembly *System.Web.Services.dll* einbinden. Im Visual Studio .NET erfolgt dies durch Hinzufügen eines Verweises (siehe Abbildung 13.6). Benötigt wird diese Assembly von der Proxy-Klasse.

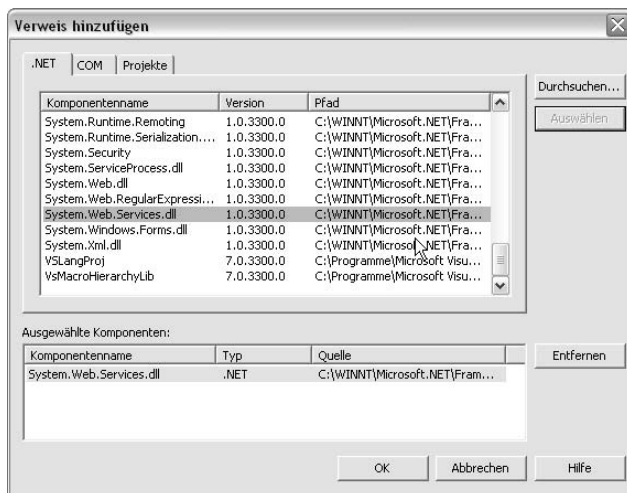


Abbildung 13.6: Hinzufügen der Assembly mit den Webservices-Klassen in VS.NET

Nach der Aufnahme des Verweises können Sie den Namensraum in Ihrem Programm spezifizieren:

```
Imports System.Web.Services
```

Die Proxy-Klasse verwenden

Mit dieser Datei gelingt eine eigene Applikation nun recht schnell, vorausgesetzt, Sie kennen die angebotenen Dienste überhaupt. Tabelle 13.1 zeigte die Suchfunktionen, Tabelle 13.2 die wichtigsten Parameter der Hauptfunktion `doGoogleSearch`. Der nächste Schritt ist logisch. Es wird eine Instanz der Proxy-Klasse gebildet und dann können die Funktionen aufgerufen werden:

```
Dim Google As GoogleSearchService = New GoogleSearchService()
```

Eine zweite Klasse, `GoogleSearchResult`, dient der Kontrolle der Ergebnisse. Dieser wird das Resultat der Abfrage übergeben:

```
Dim GoogleResult As GoogleSearchResult = Google.doGoogleSearch ↵
    (gKey, gQ, gStart, gMaxResults, ↵
    false, "", false, "de", "", "")
```

Es verbleibt noch die verhältnismäßig leicht Aufgabe, die Variablen mit den passenden Werten zu füllen und die Ergebnisse entsprechend zu verwerten. `gQ` enthält die Suchwörter, so wie Sie sie

auch in das Google-Suchformular eingeben würden. Testen Sie am besten dort, wie Sie die besten Resultate erzielen. Das folgende Listing zeigt die fertige Applikation:

```
Imports System.Web.Services

Public Class GoogleLinks
    Inherits System.Web.UI.Page

    Public LinkListe As PlaceHolder

    Private Const gKey As String = "XXXXXXXXXX IHR KEY
XXXXXXXXXX"
    Private Const gMaxResults As Integer = 10
    Private gStart As Integer = 0
    Private gQ As String

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Dim l As Label = New Label()
        l.Text = "Unsere aktuelle Linkliste"
        Dim Google As GoogleSearchService = New
GoogleSearchService()
        gQ = "WSDL SOAP WebService VB.NET"
        Dim GoogleResult As GoogleSearchResult
        = Google.doGoogleSearch(gKey, gQ, gStart,
            gMaxResults,
            False, "", False, "lang_de",
            "", "")
        If GoogleResult.resultElements.Length > 0 Then
            Dim results As Integer =
GoogleResult.resultElements.Length
            Dim hl As HyperLink
            Dim gcBreak As HtmlGenericControl
            Dim i As Integer
            For i = 0 To results - 1
                hl = New HyperLink()
                gcBreak = New HtmlGenericControl("br")
                hl.NavigateUrl =
GoogleResult.resultElements(i).URL
                hl.Text = GoogleResult.resultElements(i).title
                LinkListe.Controls.Add(hl)
                LinkListe.Controls.Add(gcBreak)
            Next
```

```

Else
    l.Text = "Leider gab es keine Treffer"
    LinkListe.Controls.Add(l)
End If
End Sub
End Class

```

Listing 13.5: Erstellung der Linkliste mit Hilfe des Google-Webservices (GoogleLinks.aspx.vb)

Bevor Sie das Programm bei sich starten, müssen Sie sich von Google den am Anfang erwähnten Lizenzschlüssel beschaffen und statt der XX-Reihe im Listing eintragen (Variable *gKey*). Auf einem Entwicklungssystem ist außerdem natürlich eine Internet-Verbindung notwendig. Eine Firewall ist kein Problem, da als Transportprotokoll nur HTTP (Port 80) verwendet wird.

Unsere Linkliste

```

ASP.NET Web Services
Web Services 101 in ASP.NET
Session State in ASP.NET Web Services
Web Services in Anwendungen konsumieren
Web Services 101 in ASP.NET
Session State in ASP.NET Web Services
XML und SOAP
C# und .NET C# und .NET - - Komponentenarchitektur ...
C# und .NET- Komponentenarchitektur
SIGS DATACOM -

```

Abbildung 13.7: Die Linkliste – jederzeit frisch von Google erstellt

Wie es funktioniert

Der Ablauf beginnt mit der Instanziierung der Proxy-Klasse (der Name ist durch Webservice selbst festgelegt, kann also nicht geändert werden):

```
Dim Google As GoogleSearchService = New GoogleSearchService()
```

Das Ergebnis der Abfrage wird von der Methode bestimmt, die aufgerufen wird. Für die Standardfunktion `doGoogleSearch` hat dies den Typ `GoogleSearchResult`. Die Parameter werden durch die Definition in der WSDL-Datei bestimmt:

```

GoogleSearchResult GoogleResult = Google.doGoogleSearch
(gKey, gQ, gStart, gMaxResults,
false, "", false, "lang_de", "", "")

```

Falls nun Ergebnisse ermittelt wurden, kann dies durch den Antwortparameter `resultElements` ermittelt werden:

```
If GoogleResult.resultElements.Length > 0 Then
```

Auch die Struktur der Antwort kann der WSDL-Datei entnommen werden. Darüber hinaus ist natürlich auch die Proxyklasse lesbar und damit funktioniert auch Intellisense in Visual Studio .NET. Für die Anzeige wird nun für jedes Element ein Hyperlink erzeugt:

```
h1 = New HyperLink()
```

Die Trennung erfolgt mit einfache `<br>`-Tags:

```
gcBreak = New HtmlGenericControl("br")
```

Der Hyperlink wird nun mit der URL und dem Text versorgt:

```
h1.NavigateUrl = GoogleResult.resultElements(i).URL
```

```
h1.Text = GoogleResult.resultElements(i).title
```

Zuletzt werden die beiden erzeugten Steuerelemente dem Placeholder-Steuerelement hinzugefügt:

```
LinkListe.Controls.Add(h1)
```

```
LinkListe.Controls.Add(gcBreak)
```

Wie es weiter geht

Die bisherige Lösung zeigt, dass die Nutzung eines Webservices mit .NET extrem einfach ist. Damit keine Langeweile aufkommt, soll nun etwas weiter geforscht werden im Reich der Möglichkeiten. Die Erstellung der Proxyklasse mit *wsdl.exe* ist schnell und einfach. Leider müssen Sie immer dann, wenn der Dienst sich etwas ändert, diese Übersetzung erneut vornehmen. Besser wäre es, die Übersetzung aus dem Code heraus vorzunehmen. Als erste Übung soll die WSDL-Datei gelesen und analysiert werden. Ausgangspunkt bildet die Klasse `ServiceDescription` aus dem Namensraum `System.Web.Services.Description`. Sie benötigen auch dafür den Verweis zur Assembly `System.Web.Services.dll`, wie im Abschnitt »Webservices in Visual Studio .NET verfügbar machen« ab Seite 1059 bereits beschrieben.

13.3.3 WSDL programmtechnisch auswerten

Googles WSDL-Informationen werden mit folgender Anweisung programmtechnisch erobert:

```
Dim WsdlFile As String = Server.MapPath("GoogleSearch.wsdl")
Dim gWsdL As ServiceDescription ←
    = ServiceDescription.Read(WsdlFile)
```

Um auf die WSDL-Informationen reagieren zu können, wird zuerst eine Information über die verfügbaren Funktionen benötigt. Eine Funktion wird im WSDL-Dialekt als »Operation« bezeichnet und folgerichtig enthält die Klasse `ServiceDescription` eine Auflistung `Operations`. Die übergeordnete Instanz dazu ist »Ports«, wobei Google nur einen Port zu seinen Diensten anbietet, was die Gestaltung des Programms vereinfacht. Die Liste der Dienste kann nun an ein `RadioButtonList`-Steuerelement übergeben werden:

```
GoogleServices.DataSource = gWsdL.PortTypes(0).Operations
GoogleServices.DataTextField = "Name"
GoogleServices.DataBind()
```

Fügt nun Google mehr Dienste hinzu, verlängert sich die Liste automatisch, auch ohne den Einsatz von *wsdl.exe*. Auf Basis der getroffenen Auswahl soll nun ein Formular entworfen werden, das exakt nach der Gestaltung der Eingabedaten erzeugt wird. Dazu definiert WSDL für jede Operation eine so genannte »Message«, und zwar eine für die Eingabe (`Input`, zum Aufruf der Funktion) und eine für die Ausgabe (`Output`, der Rücklauf vom Dienst). Diese Definition bildet auch die Basis für die Gestaltung der SOAP-Nachricht. Im Beispiel wird die `Operations`-Kollektion durchlaufen, um die Auswahl zu ermitteln. Dann besteht Zugriff auf den Eingabetyp:

```
Dim currentInputName As String ←
    = gWsdL.PortTypes(0).Operations(i).Messages.Input.Message.Name
```

Der Index *i* durchläuft die Schleife der `Operations`-Kollektion. Aus dem Namen der Message wird nun die Message selbst extrahiert. Das Objekt entspricht exakt dem Abschnitt `<message>` in der WSDL-Datei:

```
Dim currentInputMessage As Message = gWsdL.Messages(currentInputName)
```

Dieser Abschnitt enthält für jedes erforderliche Feld eine Definition `<part>`, die letztlich in einem passenden Eingabefeld resultieren soll. `<part>` enthält den Namen und den Datentyp, angegeben im Format der Beschreibungssprache XSD (XML Schema Definition). Dort findet man Bezeichner wie »xsd:string« usw. Da der Namensraum (hier »xsd:«) nicht weiter interessiert, wird die Auswahl auf den Namen bezogen:

```
Select currentInputMessage.Parts(j).Type.Name
```

Jetzt gilt es nur noch, jeden erforderlichen Datentyp in den *Case-Zweigen* zu erzeugen und die passenden Eingabefelder dazu zu erkennen. Vergleichbar funktioniert das auch mit den Output-Nachrichten. Damit kann das Programm erkennen, was Google auf Grund der Anfrage zurücksendet.

Etwas schwieriger ist der Start der Anfrage selbst, denn nun werden nicht nur die Informationen aus der WSDL-Datei benötigt, wie für die Erstellung des Formulars, sondern die gesamte Proxy-Klasse.

Einsatz von Reflektion zur dynamischen Kompilierung

Die Aufgabe besteht nun darin, die WSDL-Datei zur Laufzeit in eine Proxyklasse zu konvertieren. Der Reflektion genannte Mechanismus, der hier zum Einsatz kommt, basiert auf einer Sammlung von Klassen im Namensraum `System.Reflection`. Da die Datei zur Laufzeit übersetzt werden soll, sind außerdem die Namensräume `System.CodeDom` und `System.CodeDom.Compiler` nötig, die Zugriff auf den jeweiligen Compiler haben. Das vollständige Listing finden Sie unter dem Namen *WsdChecker.aspx* bzw. *WsdChecker.aspx.vb* im Projekt *vbnetadvanced*. Exemplarisch soll hier nur der Teil mit der dynamischen Kompilierung vorgestellt werden – die Methode *WsdCompileAssembly*:

Dynamisch eine Klasse für VB.NET in C# erzeugen

```
public void WsdCompileAssembly (ServiceDescription gWsd, bool
ForceRecompile)
{
    if (!File.Exists (AssemblyFile) || ForceRecompile)
    {
        CodeNamespace cn = ➤
            new CodeNamespace ("Addison.VBNet.Advanced");
        ServiceDescriptionImporter sdi ➤
            = new ServiceDescriptionImporter ();
        sdi.AddServiceDescription (gWsd, null, null);
```

```

sdi.ProtocolName = "Soap";
sdi.Import (cn, null);
CSharpCodeProvider cscp = new CSharpCodeProvider ();
ICodeGenerator icg = cscp.CreateGenerator ();
StringBuilder sb = new StringBuilder ();
StringWriter sw = new StringWriter (sb);
icg.GenerateCodeFromNamespace (cn, sw, null);
sw.Close ();
CompilerParameters cp = new CompilerParameters ();
cp.ReferencedAssemblies.Add ("System.dll");
cp.ReferencedAssemblies.Add ("System.Data.dll");
cp.ReferencedAssemblies.Add ("System.Xml.dll");
cp.ReferencedAssemblies.Add ("System.Web.Services.dll");
cp.GenerateExecutable = false;
cp.GenerateInMemory = false;
cp.OutputAssembly = AssemblyFile;
cp.IncludeDebugInformation = false;
ICodeCompiler icc = cscp.CreateCompiler ();
cr = icc.CompileAssemblyFromSource (cp, sb.ToString ());
if (cr.Errors.Count > 0)
{
    throw new Exception (
        String.Format("{0} Fehler beim kompilieren",
            cr.Errors.Count));
}
}
}

```

Listing 13.6: Dynamischer Compiler zum Übersetzen der WSDL-Datei in eine Proxyklasse

Das Programm erzeugt eine Proxy-Klasse im Namensraum des aktuellen Projekts: *Addison.VBNet.Advanced*. Das können Sie natürlich ändern. Den Import der WSDL-Datei übernimmt *ServiceDescriptionImporter*. Die Erstellung der Klasse in C#¹ erledigt *CSharpCodeProvider*. Mit *CompilerParameters* werden die Bedingungen der Datei festgelegt, unter anderem, welche Assemblies benötigt werden:

```
cp.ReferencedAssemblies.Add ("System.dll");
```

Dann wird bestimmt, dass die Assembly als Datei und als nicht ausführbar erzeugt wird. Der Dateiname bestimmt, wo die Assembly landet:

1 Die Wahl der Sprache erfolgte auch aus demonstrativen Gründen, ebenso gut wäre VB.NET in der Lage, die Aufgabe zu erfüllen. Der Aufruf des C#-Compilers zeigt die Interoperabilität in .NET, denn die Verwendung erfolgt später mit VB.NET.

```
cp.OutputAssembly = AssemblyFile;
```

Die Variable *AssemblyFile* wurde bereits in der Deklarationssektion definiert (dies ist nicht im abgedruckten Listing zu sehen). Dort finden Sie Folgendes (steht alles in einer Zeile):

```
string AssemblyFile =  
System.Web.HttpContext.Current.Server.MapPath  
(System.Web.HttpContext.Current.Request.ApplicationPath+@  
"\\bin\\Wds1Assembly.dll");
```

Mit diesen Daten kann nun kompiliert werden. Dazu wird ein virtueller Compiler erzeugt:

```
ICodeCompiler icc = cscp.CreateCompiler();
```

In der folgenden Zeile wird dann die Übersetzung angestoßen:

```
cr = icc.CompileAssemblyFromSource(cp, sb.ToString());
```

Damit das verwendet werden kann, ist nun noch einiger Aufwand nötig. Ein Blick auf das Ergebnis zeigt, dass es sich lohnt. Zuerst das automatisch aus den WSDL-Daten erzeugte Formular:

Automatischer Google-Service

Wählen Sie einen Dienst:

☐ doGetCachedPage
☐ doSpellingSuggestion
☒ doGoogleSearch

KEY	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Q	ASP.ET
START	0
MAXRESULTS	10
FILTER	☐
RESTRICT	
SAFESEARCH	☐
LR	lang_de
IE	
OE	

Abbildung 13.8: Automatisch erzeugtes Formular

Eine Version in VB.NET

Abschließend soll noch der Code der kompletten VB.NET-Version gezeigt werden. Dies ist nun die ultimative Applikation, mit automatischer Auswertung der WSDL-Datei und Generierung der Ergebnisse, wie vorstehend bereits abgebildet.

```
Imports System.IO
Imports System.Text
Imports System.Reflection
Imports System.CodeDom
Imports System.CodeDom.Compiler
Imports System.Xml
Imports System.Xml.Schema
Imports System.Web.Services
Imports System.Web.Services.Description
Imports System.Web.Services.Protocols
Imports System.Collections
Imports System.ComponentModel
Imports System.Data
Imports Microsoft.VisualBasic

Public Class WsdlCompiler
    Protected cr As CompilerResults
    Dim AssemblyFile As String =
System.Web.HttpContext.Current.Server.MapPath(System.Web.
HttpContext.Current.Request.ApplicationPath +
"\bin\WsdlAssembly.dll")
    Public currentType As Type
    Public ResultElementType As Type
    Public DirectoryCategoryType As Type

    Public Sub WsdlCompileAssembly(ByVal gWsdL As
ServiceDescription, ByVal ForceRecompile As Boolean)
        If Not File.Exists(AssemblyFile) Or ForceRecompile Then
            Dim cn As CodeNamespace = New
CodeNamespace("Addison.VBNet.Advanced")
            Dim sdi As ServiceDescriptionImporter = New
ServiceDescriptionImporter()
            sdi.AddServiceDescription(gWsdL, Nothing, Nothing)
            sdi.ProtocolName = "Soap"
            sdi.Import(cn, Nothing)
            Dim cscp As VBCodeProvider = New VBCodeProvider()
            Dim icg As ICodeGenerator = cscp.CreateGenerator()
            Dim sb As StringBuilder = New StringBuilder()
            Dim sw As StringWriter = New StringWriter(sb)
            icg.GenerateCodeFromNamespace(cn, sw, Nothing)
            sw.Close()
            Dim cp As CompilerParameters = New
```

```

CompilerParameters()
    cp.ReferencedAssemblies.Add("System.dll")
    cp.ReferencedAssemblies.Add("System.Data.dll")
    cp.ReferencedAssemblies.Add("System.Xml.dll")

cp.ReferencedAssemblies.Add("System.Web.Services.dll")
    cp.GenerateExecutable = False
    cp.GenerateInMemory = False
    cp.OutputAssembly = AssemblyFile
    cp.IncludeDebugInformation = False
    Dim icc As ICodeCompiler = cscp.CreateCompiler()
    cr = icc.CompileAssemblyFromSource(cp, sb.ToString())
    If cr.Errors.Count > 0 Then
        Throw New Exception(String.Format("{0} Fehler
beim kompilieren", cr.Errors.Count))
    End If
End If
End Sub

Public Function WsdlCall(ByVal SoapFunction As String, ByVal
parameters As Object()) As Object
    If Not File.Exists(AssemblyFile) Then
        Throw New FileNotFoundException("Assembly wurde nicht
gefunden. Compilerfehler?")
    End If
    Dim ass As System.Reflection.Assembly =
System.Reflection.Assembly.LoadFrom(AssemblyFile)
    Dim t As Type = ass.GetType("Addison.VBNet.Advanced." +
"GoogleSearchService")
    Dim o As Object = Activator.CreateInstance(t)
    Dim m As MethodInfo =
Activator.CreateInstance(t).GetType().GetMethod(SoapFunction)
    Dim r As Object = m.Invoke(o, parameters)
    If Not r Is Nothing Then
        currentType = r.GetType() ' save to convert
correctly
        ' Type für Subtypen
        ResultElementType =
ass.GetType("Addison.VBNet.Advanced." + "ResultElement")
        DirectoryCategoryType =
ass.GetType("Addison.VBNet.Advanced." + "DirectoryCategory")
    End If
    Return r
End Function

End Class

Public Class WsdlForm
    Inherits System.Web.UI.Page

```

```

Protected WithEvents GoogleServices As RadioButtonList
Protected GoogleMessage As Placeholder
Protected GoogleResultStream As Label
Protected WithEvents SendButton As Button
Private Const gKey As String =
"jsvVwe6c5Wcj6QnmzMi+fjuhv/eONjEl"
Dim gWsd1 As ServiceDescription
Private parameters() As Object

Public Sub CallService(ByVal sender As Object, ByVal e As
EventArgs)
    gWsd1 =
ServiceDescription.Read(Server.MapPath("GoogleSearch.wsdl"))
    Dim myGoogle As Wsd1Compiler = New Wsd1Compiler()
    myGoogle.Wsd1CompileAssembly(gWsd1, True) ' Recompile
aktiv in der Testphase (später False einsetzen)
    Dim currentService As String =
GoogleServices.SelectedItem.Value
    Dim i As Integer = GetCurrentOperation(gWsd1,
currentService)
    Dim currentInputName As String =
gWsd1.PortTypes(0).Operations(i).Messages.Input.Message.Name
    Dim currentOutputName As String =
gWsd1.PortTypes(0).Operations(i).Messages.Output.Message.Name
    Dim currentInputMessage As Message =
gWsd1.Messages(currentInputName)
    Dim currentOutputMessage As Message =
gWsd1.Messages(currentOutputName)
    parameters = New Object(currentInputMessage.Parts.Count -
1) {}
    Dim j As Integer
    For j = 0 To currentInputMessage.Parts.Count - 1
        Select Case currentInputMessage.Parts(j).Type.Name
            Case "string"
                parameters(j) =
Request.Form(currentInputMessage.Parts(j).Name)
            Case "boolean"
                parameters(j) =
(Request.Form(currentInputMessage.Parts(j).Name) <> Nothing)
            Case "int"
                parameters(j) =
Convert.ToInt32(Request.Form(currentInputMessage.Parts(j).Name))
        End Select
    Next
    ' TODO: set parameters here
    Dim GoogleResult As Object =
myGoogle.Wsd1Call(currentService, parameters)
    ' Rückgabetyp
    Select Case

```

```

currentOutputMessage.Parts("return").Type.Name
    Case "string"
        If Not GoogleResult Is Nothing Then
            GoogleResultStream.Text =
GoogleResult.ToString()
        Else
            GoogleResultStream.Text = "Kein Ergebnis
gefunden"
        End If
    Case "base64Binary"
        Dim encEncoder As System.Text.Encoding =
System.Text.ASCIIEncoding.ASCII
        GoogleResultStream.Text =
encEncoder.GetString(CType(GoogleResult, System.Byte()), 0,
CType(GoogleResult, System.Byte()).Length)
    Case Else
        ' any user defined type ?
        If Not GoogleResult Is Nothing Then
            Dim o As Object =
Activator.CreateInstance(myGoogle.currentType)
            Dim subo As Object
            o = Convert.ChangeType(GoogleResult,
myGoogle.currentType)
            Dim r As Object
            Dim subr() As Object
            Dim t As Table = New Table()
            t.BorderStyle = BorderStyle.Solid
            t.GridLines = GridLines.Both
            t.BorderWidth = Unit.Pixel(1)
            Dim fi As FieldInfo
            For Each fi In o.GetType().GetFields()
                If fi.FieldType.Namespace = "google" Then
                    Dim hc As HtmlGenericControl = New
HtmlGenericControl("ul")
                    Dim li As HtmlGenericControl
                    Dim lc As LiteralControl
                    Dim content As String = String.Empty
                    subr =
CType(o.GetType().GetField(fi.Name).GetValue(o), Object())
                    Dim k As Integer
                    Select Case fi.FieldType.Name
                        Case "ResultElement[]"
                            For k = 0 To subr.Length - 1
                                subo =
Activator.CreateInstance(myGoogle.ResultElementType)
                                subo =
Convert.ChangeType(subr(k), myGoogle.ResultElementType)
                                Dim subfi As FieldInfo
                                For Each subfi In

```

```

subo.GetType().GetFields()
                                content =
subo.GetType().GetField(subfi.Name).GetValue(subo).ToString()
                                li = New
HtmlGenericControl("li")
                                lc = New
LiteralControl(subfi.Name + ":
" + content)
                                li.Controls.Add(lc)
                                hc.Controls.Add(li)
                                Next
                                li = New
HtmlGenericControl("li")
                                lc = New
LiteralControl("<hr>")
                                li.Controls.Add(lc)
                                hc.Controls.Add(li)
                                Next
                                Case "DirectoryCategory[]"
                                For k = 0 To subr.Length - 1
                                subo =
Activator.CreateInstance(myGoogle.DirectoryCategoryType)
                                subo =
Convert.ChangeType(subr(k), myGoogle.DirectoryCategoryType)
                                Dim subfi As FieldInfo
                                For Each subfi In
subo.GetType().GetFields()
                                content =
subo.GetType().GetField(subfi.Name).GetValue(subo).ToString()
                                li = New
HtmlGenericControl("li")
                                lc = New
LiteralControl(subfi.Name + ":
" + content)
                                li.Controls.Add(lc)
                                hc.Controls.Add(li)
                                Next
                                li = New
HtmlGenericControl("li")
                                lc = New
LiteralControl("<hr>")
                                li.Controls.Add(lc)
                                hc.Controls.Add(li)
                                Next
                                End Select
                                t.Rows.Add(GetNewRow(fi.Name, hc))
Else
r =
o.GetType().GetField(fi.Name).GetValue(o)

```

```

        t.Rows.Add(GetNewRow(fi.Name, New
LiteralControl(r.ToString()))
    End If
Next
    GoogleResultStream.Controls.Add(t)
End If
End Select
End Sub

Private Function GetNewRow(ByVal cell1 As String, ByVal cell2
As Control) As TableRow
    Dim tc1 As TableCell = New TableCell()
    Dim tc2 As TableCell = New TableCell()
    tc1.Text = CType(cell1, String)
    tc2.Controls.Add(CType(cell2, Control))
    Dim tr As TableRow = New TableRow()
    tr.Cells.Add(tc1)
    tr.Cells.Add(tc2)
    Return tr
End Function

Private Function GetCurrentOperation(ByVal gWsd1 As
ServiceDescription, ByVal currentService As String) As Integer
    Dim i As Integer
    For i = 0 To gWsd1.PortTypes(0).Operations.Count - 1
        If gWsd1.PortTypes(0).Operations(i).Name =
currentService Then
            Return i
        End If
    Next
    Return 0
End Function

Public Sub GoogleSelectForm(ByVal sender As Object, ByVal e
As EventArgs) _
    Handles GoogleServices.SelectedIndexChanged
    gWsd1 =
ServiceDescription.Read(Server.MapPath("GoogleSearch.wsdl"))
    Dim currentService As String =
GoogleServices.SelectedItem.Value
    Dim formTable As Table = New Table()
    Dim i As Integer = GetCurrentOperation(gWsd1,
currentService)
    Dim currentInputName As String =
gWsd1.PortTypes(0).Operations(i).Messages.Input.Message.Name
    Dim currentInputMessage As Message =
gWsd1.Messages(currentInputName)
    Dim j As Integer
    For j = 0 To currentInputMessage.Parts.Count - 1

```

```

        Select Case currentInputMessage.Parts(j).Type.Name
            Case "string"
                Dim hi As HtmlInputText = New
HtmlInputText("text")
                hi.MaxLength = 50
                hi.Size = 50
                hi.ID = currentInputMessage.Parts(j).Name
                hi.Name = currentInputMessage.Parts(j).Name
                If currentInputMessage.Parts(j).Name = "key"
Then
                    hi.Value = gKey
                End If
                formTable.Rows.Add(GetNewRow(
currentInputMessage.Parts(j).Name.ToUpper(), hi))
            Case "boolean"
                Dim hc As HtmlInputCheckBox = New
HtmlInputCheckBox()
                hc.ID = currentInputMessage.Parts(j).Name
                hc.Name = currentInputMessage.Parts(j).Name
                hc.Value = "1"
                formTable.Rows.Add(GetNewRow
(currentInputMessage.Parts(j).Name.ToUpper(), hc))
            Case "int"
                Dim ht As HtmlInputText = New
HtmlInputText("text")
                ht.MaxLength = 5
                ht.Size = 5
                ht.Value = "0"
                ht.ID = currentInputMessage.Parts(j).Name
                ht.Name = currentInputMessage.Parts(j).Name
                formTable.Rows.Add(GetNewRow
(currentInputMessage.Parts(j).Name.ToUpper(), ht))
        End Select
    Next
    GoogleMessage.Controls.Add(formTable)
    SendButton.Visible = True
End Sub

Public Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) _
    Handles MyBase.Load
    If Not Page.IsPostBack Then
        gWsd1 =
ServiceDescription.Read(Server.MapPath("GoogleSearch.wsd1"))
        GoogleServices.DataSource =

```

```
gWsd1.PortTypes(0).Operations
    GoogleServices.DataTextField = "Name"
    GoogleServices.DataBind()
End If
End Sub

End Class
```

Listing 13.7: Das Programm WsdlChecker.aspx.vb – ein automatischer Webservice

Der Code soll hier unkommentiert gezeigt werden, als Anregung für eigene Experimente. Er zeigt, dass auch komplexe Webservices mit überschaubarem Aufwand zu programmieren sind.

13.4 Webservices anbieten

Vielleicht sind Sie durch die Nutzung eines Webservices auf den Geschmack gekommen und möchten nun selbst solche Dienste anbieten. Ihr Webspace bei einem Provider mit ASP.NET-Unterstützung reicht dafür völlig aus. Es mangelt oft eher an der Idee als an der Machbarkeit.

13.4.1 Anregungen für den praktischen Einsatz

Dem praktischen Einsatz von Webservices steht heute nichts im Wege. Mit ASP.NET ist es ausgesprochen einfach, eine Applikation um das entsprechende Angebot zu erweitern. Was fehlt, sind praktische Ideen und vermutlich die passenden Kommunikationspartner.

Ein paar Anregungen sollen helfen, den richtigen Weg zu finden. Wenn Sie als Softwareentwickler arbeiten, können Sie die eine oder andere Idee vielleicht Ihrem Kunden nahe legen:

- Denkbar wäre ein Reservierungssystem mit externen Partnern. Jeder Partner betreibt ein eigenes System und stellt den aktuellen Reservierungsstand per Webservice zur Verfügung. Bucht jemand persönlich am Terminal oder über Callcenter, werden alle Partner über Webservices nach freien Plätzen abgesucht. Das System spart einen zentralen Server.
- Bieten Sie Dienstleistungen für Webserver an, beispielsweise Konvertierungen für Dateiformate, Bildbearbeitungen oder Rechtschreibkorrekturen. Anbieter können damit ihre Webseiten perfektionieren, ohne selbst investieren zu müssen.

- ▶ Sie handeln mit Waren aller Art? Dann bieten Sie Lieferanten an, deren Daten per Webservice zu aktualisieren. Oder suchen Sie Wiederverkäufer, die ihre Server über den Webservice mit aktuellen Artikeldaten versorgen – und verkaufen Sie die Programmierung gleich mit ;-).
- ▶ Sie produzieren News, Rezensionen oder ähnliche immaterielle Dinge. Die Welt wartet darauf und mit einem Webservice ist eine komfortable Übernahme auf andere Server möglich. Ist Inhalt und Angebot reizvoll, kann man damit den einen oder anderen Euro verdienen.

Es ist also nicht so schwer, Anwendungen für Webservices zu finden. Für die ersten Schritte ist freilich ein einfaches Beispiel besser geeignet.

13.4.2 Praktische Umsetzung eines Dienstes

Für die Bereitstellung eines Webservices werden Sie zuerst eine neue Direktive kennen lernen: `@WebService`. Diese ist erforderlich, wenn Sie direkt in der Seite programmieren. Mit Code Behind sieht diese Zeile folgendermaßen aus:

Direktive

```
<%@ WebService Language="vb"  
    Codebehind="ServiceVariables.asmx.vb"  
    Class="Addison.VBNet.Advanced.ServiceVariables" %>
```

Des Weiteren ist in Visual Studio .NET die bereits im vorhergehenden Abschnitt gezeigte Verknüpfung der Assembly *System.Web.Services.dll* erforderlich.

Die Erstellung des Webservices unterscheidet sich kaum von der herkömmlichen Programmierung. Letztlich läuft es darauf hinaus, dass eine Anfrage erkannt wird – eine Methode der Basisklasse der Seite ist dafür zuständig – und eine Antwort generiert wird. Die Methode sollte also auch etwas zurückgeben. Initiatoren für das gegenüber ASP.NET geänderte Verhalten sind zwei Maßnahmen:

- ▶ Deklaration der Methoden mit dem Attribut `<WebMethod>`
- ▶ Nutzung der Dateierweiterung *.asmx* an statt *.aspx*

Die Dateierweiterung ist erforderlich, weil nun ein anderer Handler benutzt werden soll.

Webservice in Visual Studio .NET erstellen

In Visual Studio .NET gehen Sie folgendermaßen vor, um einen Webservice Ihrem Projekt hinzuzufügen:

1. Wählen Sie im Kontextmenü des Projekts HINZUFÜGEN.
2. Im nächsten Menü klicken Sie auf WEBDIENST HINZUFÜGEN.
3. Aus der Vorlagenliste belassen Sie die Auswahl bei Webdienst. Vergeben Sie einen Namen und klicken dann auf OK.

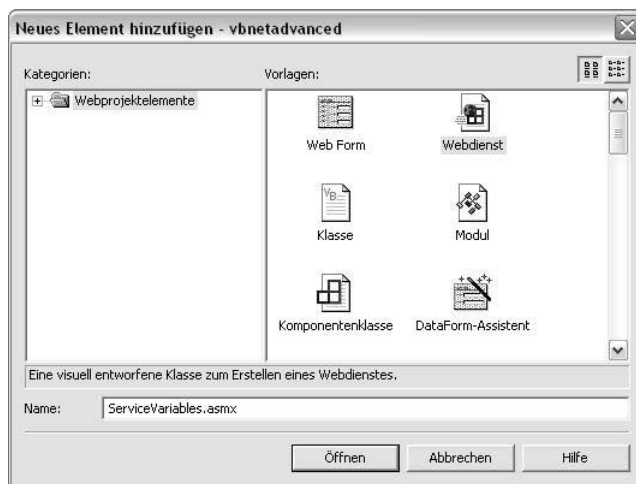


Abbildung 13.10: Ein neuer Webservice wird dem Projekt hinzugefügt

Wechseln Sie nun mit **[F7]** in den Code.

Programmierung des Webservices

Im Beispiel soll der Dienst Dateiinformationen übertragen. Dahinter steckt kein besonderer Sinn, abgesehen von der krampfhaften Vermeidung eines weiteren »Hello World«. Wenn Sie als Parameter den Namen einer Datei übergeben, wird das Dateidatum zurückgeliefert. OK, ein wenig mehr als »Hello World« ist das schon, denn andere Server könnten so die Aktualität bestimmter Dateien prüfen, ohne dass Sie denen den Zugriff auf die Dateien selbst gestatten. Hier der Dienst auf einen Blick:

```
Imports System.IO
Imports System.Collections
Imports System.Diagnostics
```

```
Imports System.ComponentModel
Imports System.Web.Services

<WebService(Name:="Dateiaktualisierungsdienst", _
            Namespace:="http://www.joergkrause.de")> _
Public Class ServiceVariables
    Inherits System.Web.Services.WebService
    Public Sub New()
        'CODEGEN: Dieser Aufruf ist für den
        ASP.NET-Webdienst-Designer erforderlich.
        InitializeComponent()
    End Sub

    <WebMethod(Description:="Stellt fest, ob eine Datei in
    diesem Projekt geändert wurde und gibt
    dafür das Datum des
    letzten Zugriff aus.")> _
    Public Function GetFileDate(ByVal FileName As String) _
    As String
        Dim datum As String = String.Empty
        If FileName.Length > 0 Then
            Dim path As String = Server.MapPath(FileName)
            Try
                Dim dt As DateTime = File.GetLastWriteTime(path)
                datum = dt.ToString() + " " + dt.ToString()
            Catch
                datum = "Datei nicht gefunden"
            End Try
        End If
        Return "Letzte Änderung am " + datum
    End Function
End Class
```

Listing 13.8: Der Webservice (Ausschnitt aus ServiceVariables.asmx, befreit von den automatisch generierten Codes des Visual Studio .NET-Designers)

Praktisch ist hier wenig von Webservices zu sehen – vom Attribut `<WebMethod>` mal abgesehen. Dass es dennoch funktioniert, soll ein erster Test beweisen.

Sie können den gesamten Dienst konfigurieren, indem Sie vor der Klassendefinition das Attribut `<WebService>` einfügen.



Einen eigenen Webservice testen

Der Test besteht darin, die *asmx*-Datei aufzurufen. Das führt zu folgendem Bild:

ServiceVariables

Folgende Vorgänge werden unterstützt. Eine ausführliche Definition finden Sie in der [Dienstbeschreibung](#).

- [GetFileDate](#)

Der Webservice verwendet `http://tempuri.org/` als Standardnamespace.

Empfehlung: Ändern Sie den Standardnamespace, bevor der XML-Webservice publiziert wird.

Alle XML-Webservices erfordern einen eindeutigen Namespace, um die Clientanwendungen von anderen Diensten im Web zu unterscheiden. `http://tempuri.org/` ist für XML-Webservices verfügbar, die gerade entwickelt werden. Bereits veröffentlichte XML-Webservices sollten einen permanenten Namespace verwenden.

Ihr XML-Webservice muss von einem Namespace identifiziert werden, der von Ihnen gesteuert wird. Sie können z.B. den Internetdomännennamen Ihres Unternehmens als Teil des Namespace verwenden. Obwohl viele XML-Webservicespaces wie URLs aussehen, müssen sie nicht auf tatsächliche Ressourcen im Web zeigen. (XML-Webservicespaces sind URIs.)

Für XML-Webservices, die mit ASP.NET erstellt werden, kann der Standardnamespace mit der Namespace-Eigenschaft des `WebService`-Attributs geändert werden. Das `WebService`-Attribut wird auf die Klasse angewendet, die die XML-Webservicesmethoden enthält. Im folgenden Codebeispiel wird der Namespace auf `"http://microsoft.com/webservices/"` festgelegt:

```
C#
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // Implementierung
}
```

Visual Basic.NET

```
<WebService(Namespace="http://microsoft.com/webservices/")> Public Class MyWebService
    ' Implementierung
End Class
```

Weitere Informationen über XML-Namespace finden Sie in den W3C-Empfehlungen unter [Namespaces in XML](#).

Weitere Informationen über WSDL finden Sie unter [WSDL-Spezifikation](#).

Weitere Informationen über URI finden Sie unter [RFC 2396](#).

Abbildung 13.11: Automatische Beschreibungsseite der Anwendung

Sie finden oben den Namen der Methode wieder. Es ist nahe liegend, dass hier mehrere Methoden aufgeführt werden können. Wählen Sie nun die vorhandene Methode aus (siehe Abbildung 13.12).

Es wird ein Eingabefeld für jeden Parameter generiert. Da es sich im Beispiel um eine Abfrage von Dateidaten handelt, geben Sie hier einen Dateinamen ein (siehe Abbildung 13.13).

Mit Klick auf Aufrufen wird nun der eigentliche Dienst gestartet und die Antwort wird angezeigt (siehe Abbildung 13.14).

Wenn Sie sich nun wundern, warum Sie hier XML sehen: Webservices dienen der Kommunikation zwischen Servern, nicht zwischen Server und Mensch. Deshalb ist das Ausgabeformat XML.

ServiceVariables

Klicken Sie [hier](#), um die vollständige Vorgangsliste anzuzeigen.

GetFileDate

Testen

Klicken Sie auf 'Aufrufen', um den Vorgang mit dem HTTP GET-Protokoll zu testen.

Parameter	Wert
FileName:	

SOAP

Folgendes ist eine Beispielfrage- und Antwort für SOAP. Die angezeigten Platzhalter müssen durch richtige Werte ersetzt werden.

```
POST /dabuch/csharpadvanced/ServiceVariables.aspx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetFileDate"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetFileDate xmlns="http://tempuri.org/">
      <FileName>string</FileName>
    </GetFileDate>
  </soap:Body>
</soap:Envelope>
```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetFileDateResponse xmlns="http://tempuri.org/">
      <GetFileDateResult>string</GetFileDateResult>
    </GetFileDateResponse>
  </soap:Body>
</soap:Envelope>
```

HTTP GET

Folgendes ist eine Beispielfrage- und Antwort für HTTP GET. Die angezeigten Platzhalter müssen durch richtige Werte ersetzt werden.

```
GET /dabuch/csharpadvanced/ServiceVariables.aspx/GetFileDate?FileName=string HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://tempuri.org/">string</string>
```

HTTP POST

Folgendes ist eine Beispielfrage- und Antwort für HTTP POST. Die angezeigten Platzhalter müssen durch richtige Werte ersetzt werden.

```
POST /dabuch/csharpadvanced/ServiceVariables.aspx/GetFileDate HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

FileName=string

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://tempuri.org/">string</string>
```

Abbildung 13.12: Beschreibung der Webservice-Methode

GetFileDate

Testen

Klicken Sie auf 'Aufrufen', um den Vorgang mit dem HTTP GET-Protokoll zu testen.

Parameter	Wert
FileName:	ServiceVariables.aspx

Abbildung 13.13: Bedienung des Parameters

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Letzte Änderung am Mittwoch, 21. August 2002 12:14:29</string>
```

Abbildung 13.14: Die Antwort – computerlesbar

Welche Bedeutung hat tempuri.org?

In der XML-Antwort finden Sie die Angabe eines Namensraumes mit dem etwas mysteriös klingenden Namen »http://tempuri.org«. Was hat das zu bedeuten?

Dies ist ein temporärer Namensraum, der immer dann verwendet wird, wenn Sie keinen explizit spezifizieren. XML-Namensräume identifizieren eindeutig einen XML Webservice. Wenn also jemand anders einen vergleichbaren Dienst mit demselben Namen für die Methode anbietet, müssen diese dennoch unterscheidbar sein. Der Namensraum löst das Problem. Sie sollten deshalb bei einem Live-Server den Namensraum ändern. Es bietet sich an, hierzu den Domain-Namen zu verwenden, unter der der Server erreichbar ist, denn dieser Namen ist eindeutig. Dazu wird einfach das Attribut <WebService> erweitert:

```
<WebService(Namespace:="http://www.joergkrause.de")>
```

Das Attribut <WebService> steuert den gesamten Dienst, nicht die Methoden.

Konfiguration mit Attributen

Die Konfiguration des Webservice erfolgt mit den Attributen <WebService> und <WebMethod>. Die zulässigen Parameter finden Sie in den beiden folgenden Tabellen:

Parameter	Bedeutung
Description	Eine allgemeine Beschreibung
Name	Der gewünschte öffentliche Name
Namespace	Der Namensraum; ohne Angabe wird »tempuri.org« verwendet

Tabelle 13.4: Parameter des Attributes <WebService> – alle vom Typ String

Für jede einzelne Methode sind folgende Parameter anwendbar:

Parameter	Datentyp	Bedeutung
BufferResponse	Boolean	Pufferung der Ausgaben, standardmäßig eingeschaltet
CacheDuration	Integer	Zwischenspeicherung der Clientanforderungen
Description	String	Beschreibung
EnableSession	Boolean	Sitzungs-Management soll unterstützt werden. Dies ist standardmäßig deaktiviert.
MessageName	String	Alias-Name der Methode
TransactionOption	TransactionOption	Eine Aufzählung, die Optionen über die Verwendung von Transaktionen bestimmt

Tabelle 13.5: Parameter des Attributes `<WebMethod>`

Die folgende Abbildung zeigt, wie die geänderten Parameter der Attribute die Anzeige der Testumgebung modifizieren:

Dateiaktualisierungsdienst

Klicken Sie [hier](#), um die vollständige Vorgangsliste anzuzeigen.

GetFileDate

Stellt fest, ob eine Datei in diesem Projekt geändert wurde und gibt dafür das Datum des letzten Zugriffs aus.

Testen

Klicken Sie auf 'Aufrufen', um den Vorgang mit dem HTTP GET-Protokoll zu testen.

Parameter: Wert

FileName:

Aufrufen

Abbildung 13.15: Erzeugen einer eigenen Beschreibung für den Webservice

Mit der passenden Anwendung steht der intensiven Nutzung nichts mehr im Wege. Im Detail kann man zwar noch einiges am Verfahren verkomplizieren, aber dies sollten Sie erst nach den ersten erfolgreichen Versuchen tun.

D Anhang

14 Über die Autoren

Für Fragen, Anregungen aber auch Kritik und Hinweise stehen Ihnen die Autoren gern zur Verfügung. Dieses Buch soll Ihnen den schnellen Einstieg in eine komplexe Materie ermöglichen. Zugleich soll es aber auch einen professionellen Anspruch erfüllen, für die Autoren und Verlag gleichermaßen stehen. Dies war in keiner Phase des Projekts eine leichte Aufgabe.

Insofern sind Verbesserungsvorschläge und konstruktive Anmerkungen gern gesehen und finden in künftigen Auflagen sicher ihre Entsprechung.

14.1 Die Autoren kurz vorgestellt

Jörg Krause, Jahrgang 1964, wohnt mit seiner Familie (1 Kind) in Berlin. Er arbeitet als freier Autor, Software-Architekt und Consultant. Seine Arbeitsschwerpunkte sind:

- ▶ Entwurf und Programmierung von Internetapplikationen und Datenbankstrukturen (PHP, ASP, ASP.NET, C#/J#, XML/XSL, HTML, MS SQL-Server/Oracle/MySQL, UML)
- ▶ Consulting für Start-Ups und »Old Economy«: unabhängig, kompetent und professionell.
- ▶ Seminare über Webserverprogrammierung (HTML, ASP, PHP, ASP.NET, C#, VB.NET, Framework allgemein, XML) und WinForms-Programmierung mit .NET
- ▶ Journalistische Arbeiten, Artikel, Vorträge etc.

Bisherige Veröffentlichungen hatten meist Folgendes zum Thema:

- ▶ Windows 2000 (Carl Hanser)
- ▶ Microsoft Active Server Pages, ASP.NET mit C# und mit VB.NET (Addison-Wesley)
- ▶ XML im .NET-Framework (Carl Hanser)

Jörg Krause



- PHP 4, PHP Referenz (Carl Hanser)
- Electronic Commerce und Online Marketing (Carl Hanser)

Daneben arbeitet er für viele Fachzeitschriften, in denen regelmäßig Beiträge erscheinen, unter anderem im dotnet-Magazin, iX, Linux Enterprise, PHP-Magazin und Java-Magazin sowie PC Professional und PC Magazin.

Uwe Bünning



Uwe Bünning, Jahrgang 1967, wohnt mit seiner Familie in Berlin. Er hat zwei Kinder. Ausbildung und Abschluss als Diplom-Wirtschaftsinformatiker. Er arbeitet als freier Autor und Consultant.

Seine Arbeitsschwerpunkte:

- Aufbau und Integration heterogener Netzwerke
- Consulting Prepress-Bereich und Netzwerke
- Seminare zu Prepress und Netzwerktechnik

Fachbücher schrieb er bisher über folgende Themen:

- Windows 2000 (Carl Hanser)
- Microsoft Active Server Pages (Data Becker)
- ASP.NET mit C# und VB.NET (Addison-Wesley)

Daneben erschienen Fachbeiträge in der PC Professional und im PC Magazin.

14.2 Aktuelle Informationen finden Sie im Internet

Alle Programme, Bugfixes und Korrekturen finden Sie unter der folgenden Adresse:

<http://www.dotnet.comzept.de>



Auf dieser Seite sind auch alle Lösungen zu den Übungsaufgaben zu finden. Sie können dort auch eigene Lösungen veröffentlichen und mit anderen Lesern diskutieren.

Außerdem ist die offizielle ASP.NET-Seite von Microsoft immer einen Besuch wert:

<http://www.asp.net>

Die Autoren selbst können Sie auf ihrer Homepage näher kennen lernen: <http://www.buenning-krause.de>

14.3 Informationen über professionelle Unterstützung

Hilfe finden Sie bei den Autoren, wenn es um eines der folgenden »Probleme« geht:

- ▶ Entwicklung professioneller Websites jeder Größenordnung
- ▶ Projektmanagement und Programmierung in C#/VB.NET (ASP.NET und WinForms), JavaScript, HTML, aber auch PHP und ASP
- ▶ Schulungen, Seminare, Programmierkurse und Workshops, darunter auch spezielle Schulungen zu ASP.NET, XML und den Sprachen C# oder VB.NET und nach Kundenwunsch
- ▶ Fachliche Unterstützung für Start-Ups, Venture Capitalists und Old Economy ;-)

Anfragen senden Sie bitte direkt per E-Mail an:

info@buenning-krause.de.

Leider schaffen wir es nicht immer, jede E-Mail sofort zu beantworten. Sie können aber sicher sein, dass jede Nachricht gelesen wird. Insofern können Sie dieses Medium jederzeit nutzen, um irgendetwas los zu werden – was auch immer Sie gerne mitteilen möchten.

Stichwortverzeichnis

!

<Serializable> (Attribut) 785
.NET 59
 Host 68
.NET Framework
 installieren 40
 Redistributable 40
.NET Framework SDK 40-41
.NET-Vision 59
@@IDENTITY (T-SQL) 428
@@ROWCOUNT (T-SQL) 428
@Control 961
 AutoEventWireUp 962
@Control (Direktive) 655
@Import 959
@OutputCache 960
@OutputCache
 (Direktive) 811
@Page 955
 AutoEventWireUp 956
 Fehlersuche 956
@Reference (Direktive) 778
@Register 959
@Register (Direktive) 649
@Webservice (Direktive) 1077

A

Abandon (Methode) 784
Abandon (Session,
 Methode) 788
Abfragen (SQL) 367
Abfragesprache in
 RowFilter 900
Abs (Math, Methode) 251
abstrakt 201
Accept-Language 297
Accept-Languages-Feld 299
Access (Programm) 358
 SQL-Kommandos 361
Achsenbezeichner
 (XPath) 472
AcquireRequestState
 (Ereignis) 803
Add (ArrayList, Methode) 225
Add (Cookie, Methode) 800
Add (Forms-Methode) 519,
 559
Add (Hashtable,
 Methode) 228
Add (Methode, Session) 784
Add (Methode,
 Zeitberechnung) 289
AddAttribute (Methode,
 HtmlTextWriter) 724
AddDays (Methode) 288
AddHandler 180-181
AddMinutes (Methode) 288
AddMonth (Methode) 288
AddRange (ArrayList,
 Methode) 227, 230
AddressOf 180-181
AddSeconds (Methode) 288
AddStyleAttribute (Methode,
 HtmlTextWriter) 736
AddYears (Methode) 288
ADO.NET 871
 Beziehungen 874
 Einschränkungen 874
Aliase (SQL) 372
AllKeys (Cookie,
 Eigenschaft) 801
ALTER DATABASE
 (T-SQL) 383
ALTER PROCEDURE
 (T-SQL) 436
ALTER TABLE (T-SQL) 388
ALTER VIEW (T-SQL) 425
Alternativzweig (Select
 Case) 169
And 166
AndAlso 166
Anforderungen
 weiterleiten 767
Anzeige der Werte von
 Variablen 137
Anzeigestatus 598
Append (StringBuilder,
 Methode) 266
AppendFormat (StringBuilder,
 Methode) 266
Application (Klasse,
 Objekt) 807
Application (Objekt) 805
Application_Start
 (Ereignis) 809
Applikation 802
 Bedingte Ereignisse 805
 optimieren 821
 Standardereignisse 802
 Variablen 805
Applikations-
 ereignismodell 802
Applikationsmanagement 802
Applikationsvariablen 805
Ära nach Christus 306
ArgumentException
 (Ausnahme) 341
ArgumentNullException
 (Ausnahme) 341
ArrayList (Klasse) 225

Arrays

 BinarySearch 218
 BinarySearch (Array,
 Methode) 218
 Compare (Methode) 220
 Copy 218
 CopyTo 218
 CreateInstance 218
 GetEnumerator 218
 GetLength 218
 GetLength (Array,
 Eigenschaft) 218
 GetValue 218
 Length 218
 Rank 218
 Reverse 219
 SetValue 218
 SetValue (Array, Methode)
 218
 Sort 219
 Sortieren 220
 Verarbeiten 217
As 144
AS (SQL) 368
ASCII 263
ascx-Datei 648
asiatische Schriftzeichen 296
ASP 3.0 29
ASP-Einschränkungen 32
ASP.NET
 Sicherheits-
 funktionen 975
 Verarbeitung 69
 Vorbereitung 35
ASP.NET-Abarbeitung 151
ASPNET
 (Sicherheitskonto) 975
aspnet_regiis.exe 42
aspnet_wp.exe *siehe* Worker
 Process
Assemblies 68, 968
AssemblyInfo.vb 742
Attributes (DirectoryInfo,
 Eigenschaft) 335
Attributes (FileInfo,
 Eigenschaft) 339
Attributes (Forms-Kollektion,
 Eigenschaft) 509
Ausdrücke 166
Ausgabepufferung 752
Ausgabeverhalten eines
 Steuerelements 723
Authenticate (Methode, Forms-
 Authentication) 983
AuthenticateRequest
 (Ereignis) 803

- Authentifizierung
 - in ASP.NET 975
 - Programmierung 980
- Authentifizierungs-
 - Cookie 982
- AuthorizeRequest
 - (Ereignis) 803
- Automatisches Senden 526
- AutoPostBack (Eigenschaft,
 - Page) 541
- AutoPostBack-Funktion 526
- Autoren 1087
- B**
- Basisdatentypen 283
- Basisklassen 213
 - Array 214
 - CharEnumerator 214
 - Convert 214
 - Enum 214
 - Math 214
 - Random 214
 - String 214
- Batchprogrammierung
 - (SQL-Server) 427
- Bedingungsfunktion 170
- BEGIN (T-SQL) 433
- BEGIN ... END (T-SQL) 430
- BeginRequest (Ereignis) 802
- Benennung von
 - Standardtypen 86
- Benutzeraktivität 788
- Benutzer-
 - authentifizierung 974
- Benutzer-Steuerelemente 647
 - Alternativen 648
 - Aufbau 647
 - Eigenschaften 652
- Bereichskontroll-
 - Steuerelement 626
- BETWEEN (SQL) 370
- Beziehungen 353
- Bildauswahl, dynamisch 991
- Bilderzeugung 990, 994
 - Fehlersuche 1005
- Bildfunktionen 990
- BinarySearch (Array,
 - Methode) 218
- BinarySearch (ArrayList,
 - Methode) 227
- Bitmap (Klasse) 996
- Bitmap-Element 740
- Boolean 155
- Boolesch 161
- BoundColumn 607
- BREAK (T-SQL) 432
- browscap.ini 667
- Browserabhängig 662
 - Clientskript 663
 - Fallback-Regeln 664
 - machine.config 667
- Browsertyp 664, 770
- Buch-CD 24
- BufferOutput (Eigenschaft,
 - Response) 753
- Button (Klasse,
 - Steuerelement) 552
- ByRef 184
- Byte 155
- ByVal 184
- C**
- Cache (Klasse) 816
- CacheDependency
 - (Klasse) 818
- CacheItemRemovedCallback
 - (Delegat, Cache) 819
- CacheItemRemoved-
 - Reason 819
- Caching 809
 - Seiten 809
 - Steuerelemente 809
- Calendar (Klasse,
 - Steuerelement) 635
- Capacity (ArrayList,
 - Eigenschaft) 225
- Capacity
 - (Forms-Eigenschaft) 559
- Capacity (StringBuilder,
 - Eigenschaft) 267
- Cascading Style Sheets 510
- CASE ... WHEN ... THEN
 - (T-SQL) 431
- CAST (T-SQL) 405
- Catch 208, 340
- Ceiling (Math, Methode) 251
- Cells (Table, Kollektion) 234
- Char 155
- Chars (Eigenschaft,
 - String) 254
- CheckBoxList (Forms,
 - Steuerelement) 564
- CheckBoxList (Klasse,
 - Steuerelement) 560
- ChildNodes (XmlDocument,
 - Methode) 476
- Choose 170
- Clear (ArrayList,
 - Methode) 225
- Clear (Forms-
 - Methode) 518, 559
- Clear (Methode, Session) 784
- Clientseitige Feldprüfung 617
- Clientskript 663
- Clustered *siehe* Gruppierter
 - Index
- Code Behind 79
- Code dahinter 79
 - erzeugen 79
- Codekonventionen 85
- Color (Struktur) 737, 996
- Command (Klasse) 915
- CommandBuilder (Klasse) 904
 - Weitere Informa-
tionen 915
- COMMIT (T-SQL) 433
- Common Type System 154
- Compare (Array) 220
- Compare Validator *siehe*
 - Vergleichs-Steuerelement
- compilation 967
- Compiler 60
 - Optimierungs-
strategien 146
- Compilerfehler 130
- Compileroptionen 966
- COMPUTE (T-SQL) 402
- Concat (Methode, String) 261
- Const 161
- Consultingleistung 976
- Contains (ArrayList,
 - Methode) 227
- ContentLength (Eigenschaft,
 - HtmlInputFile) 533
- Content-type (für Bilder) 991
- ContentType
 - (Eigenschaft) 763
- ContentType (Seitendirektive,
 - Attribut) 991
- Context-Handler *siehe*
 - Kontext-Handler
- CONTINUE (T-SQL) 432
- ContrainCollection 873
- Control (Basisklasse) 513
- ControlCollection
 - (Forms-Kollektion,
Eigenschaft) 511
- ControlCollection (Klasse,
 - Typ) 518
- CONVERT (T-SQL) 404
- Cookie-Kollektion 799
- Cookies 791
 - Aufbau 794
 - Expires (Eigenschaft) 798
 - Funktionsweise 793
 - Name (Eigenschaft,
Cookies) 798
 - Setzen 797
 - Value (Eigenschaft,
Cookies) 798
- Cookies (Kollektion,
 - Request) 799
- Copy (Array, Methode) 218
- CopyTo (Array, Methode) 218
- CopyTo (Methode, String) 261
- Count (ArrayList,
 - Eigenschaft) 225

- Count (Eigenschaft, Groups, Match, Regex) 274
- Count (Forms-Eigenschaft) 559
- COUNT(*) (T-SQL) 397
- Counter 806
- Create (DirectoryInfo, Methode) 337
- Create (File, Methode) 338
- Create (FileInfo, Methode) 340
- CREATE DATABASE (T-SQL) 382
- CREATE FUNCTION (T-SQL) 438
- CREATE INDEX (T-SQL) 390
- CREATE INDEX CLUSTERED (T-SQL) 390
- CREATE INDEX UNIQUE (T-SQL) 390
- CREATE PROCEDURE (T-SQL) 435
- CREATE TABLE (T-SQL) 384
 - CHECK 386
 - DEFAULT 387
 - FOREIGN KEY 386
 - IDENTITY 386
 - NOT NULL 386
 - PRIMARY KEY 385
 - UNIQUE 386
- CREATE TRIGGER (T-SQL) 439
- CREATE VIEW (T-SQL) 424
- CreateChildControls 714
- CreateDirectory (Directory, Methode) 335
- CreateInstance (Array, Methode) 218
- CreateSpecificCulture (Methode, CultureInfo) 305
- CreateSubdirectory (DirectoryInfo, Methode) 337
- CreateText (FileInfo, Methode) 340
- CreateText (StreamWriter, Methode) 344
- CreationTime (DirectoryInfo, Eigenschaft) 335
- CreationTime (FileInfo, Eigenschaft) 339
- CSS *siehe* Cascading Style Sheets
- CSS 2.0 662
- CTS *siehe* Common Type System
- CType 145
- CultureInfo (Klasse) 304
- Currency 148
- Current (IEnumerable, Eigenschaft) 236
- CurrentUICulture (Eigenschaft, CurrentThread) 323
- CURRENT_TIMESTAMP (T-SQL) 408
- Custom Controls *siehe* Kundenspezifische Steuerelemente
- Custom Validation Controls 629
- D**
- Data Definition Language 382
- DataAdapter (Klasse) 892
 - DataSet 895
 - Einführung 892
 - Ereignisse 920
- DataBind 927
- DataBind (DropDownList, Eigenschaft) 235
- DataBinding (Ereignis) 936
- DataColumnCollection (Klasse) 873
- DataGrid 605
- DataGrid (Steuerelement), spaltenweises Sortieren 611
- DataRelationCollection (Klasse) 873
- DataRowCollection (Klasse) 873
- DataSet 872
- DataSet (Klasse)
 - DataTable 880
 - erzeugen 878
 - isoliert 875
 - Struktur einer Tabelle 884
 - XSD 895
- DataSet (Steuerelement), mehrseitige Ausgabe 948
- DataSource (DropDownList, Eigenschaft) 235
- DataTableCollection (Klasse) 872
- DataRow (Klasse)
 - DataRowView 896
 - RowFilter 900
 - Sort 900
- Date (Eigenschaft) 286
- DATEADD (T-SQL) 410
- DATEDIFF (T-SQL) 410
- Dateiattribute 336
- Dateieigenschaften 339
- Dateien erzeugen 341
- Dateierweiterung 152
- Dateiinhalt ausgeben 751
- Dateisystem 329
- Dateizugriffsfehler 340
- Daten
 - filtern 897
 - per URL 755
 - sortieren 897
 - von Seiten zu Seite übertragen 755
- Datenadapter 892
- DATENAME (T-SQL) 409
- Datenansicht 873, 896
- Datenaustausch 750
- Datenbank 347, 352
 - Bestandteile 351
- Datenbankabfragesprache 349
- Datenbankmanagementsystem 346
- Datenbankzeiger (T-SQL) 411
- Datenbindung 571
 - Code 572
 - Einführung 571
 - Ereignisbehandlung 936
 - Schnittstellen 928
- Datenbindungssyntax 927
 - Hintergrund 935
- Datenfluss 750
- Datennormalisierung 353
- Datenquelle 928
- Datensatz 351
- Datenspeicher 872
- Datentypen 154
 - Currency 148
 - deklarieren 144
 - Eigenschaften 282
 - Methoden 282
- Datentypen deklarieren 144
- DATEPART (T-SQL) 409
- DateTime (Struktur) 285
- Datum 284
 - ermitteln 285
 - Globalisierungs-einstellungen 295
- Datum- und Zeit-Funktionen (T-SQL) 408
- Datumsberechnungen 288
- Datumsformat
 - Einzelformate 292
 - Kombinationsformate 292
- Datumsoperatoren 289
- Datumswerte 287
 - formatieren 291
- DayOfWeek (Methode) 287
- DBConcurrencyException (Ausnahme) 913
- DDL *siehe* Data Definition Language
- Deadlock 973
- Debug (Seitendirektive, Attribut) 956
- Debug-Modus 136
- Debuggen aktivieren 131
- Debuggerprobleme 139

- Decimal 155
- DECLARE (T-SQL) 429
- DECLARE CURSOR (T-SQL) 412
- Default 190
- DefaultEvent (Attribut) 744
- DefaultProperty (Attribut) 744
- Delete (Directory, Methode) 335
- Delete (DirectoryInfo, Methode) 337
- Delete (File, Methode) 339
- Delete (FileInfo, Methode) 340
- DELETE FROM (SQL) 364
- Dereferenzierung 183
- Design, Attribute 709
- Design-Time-Attribute 742
- Designer-Oberfläche 103
- Destruktor 183
- DHTML 622
- DictionaryEntry (Klasse) 231
- DIFFERENCE (T-SQL) 407
- Dim 154
- Directory (Klasse) 334
- DirectoryInfo (Klasse) 333
- DirectoryNotFoundException (Ausnahme) 333, 341
- Direktiven 954
- Discovery-Datei 1053
- display (Attribut, Steuerelement) 620
- Dispose 183
- Dispose (Graphics, Methode) 998
- Disposed (Ereignis) 805
- DISTINCT (SQL) 369
- Do...Loop 172
- Document Type Definition 449
- Dokumentendeklaration 494
- DOM-Modell 634
- dotnetfx.exe 42
- Double 155
- Downlevel 662
- DrawString (Graphics, Methode) 997
- Drive *siehe* Laufwerke
- DROP DATABASE (T-SQL) 383
- DROP INDEX (T-SQL) 390
- DROP PROCEDURE (T-SQL) 435
- DROP TABLE (T-SQL) 387
- DROP TRIGGER (T-SQL) 439
- DROP VIEW (T-SQL) 425
- DropDownList (Klasse, Steuerelement) 556
- DTD *siehe* Document Type Definition
- Duration
 - (@OutputCache) 812
- Dynamische Webseiten 56
- E**
- E-Mail 1006
 - Dateianhänge 1018
 - Empfohlene Reihenfolge der Header 1007
 - Fehlerquellen 1017
 - HTML-Mails 1018
 - Massenpost 1018
 - versenden, Zeilen-umbruch 1016
- E-Mail-Adresse 270
- ECMAScript 662
- Eigenschaften 187
 - Parameter 190
- Eigenschaften-Generator 122
- Eingebetteter Code 151
- Einzelprüfung 622
- Else 168
- Empty 148
- Empty (Methode, String) 261
- Encoding 266
- enctype-Attribut 530
- End (Ereignis) 805
- Endlosschleife 754
- EndOfStreamException (Ausnahme) 341
- EndRequest (Ereignis) 803
- EndsWith (Methode, String) 260
- Entity-Relationship-Modell 354
- Entwurfsphase 741
- Enum 205
- Enum.Format 207
- Equals (Methode) 156
- Equals (Type) 283
- Ereignisse (Formulare) 520
- Error 75
- Error (Ereignis) 805
- Erzeugt ein Verzeichnis 335
- Eulersche Zahl 251
- Eval (Methode, Datenbindung) 935
- Execute (Server, Methode) 771
- EXECUTE (T-SQL) 436
- Exists (Directory, Methode) 335
- Exists (File, Methode) 339
- Exit 173, 184
- Exit For 175
- Exkurs 21
- ExtendedProperties (Eigenschaft, DataSet) 873
- F**
- Fallback (XSLT) 481
- Fehlerarten 129
- Fehlersuche 128
- Feldkontrolle 617
- Feldprüfung
 - Datentyp 625
 - ErrorMessage (Eigenschaft) 626
 - serverseitig 622
- File (Klasse) 338
- FileAttributes (DirectoryInfo, Aufzählung) 336
- FileInfo (Klasse) 338
- FileName (Eigenschaft, HtmlInputFile) 533
- FileNotFoundException (Ausnahme) 341
- Fill (Methode, DataAdapter) 893
- FillRectangle (Graphics, Methode) 997
- FillSchema (Methode, DataAdapter) 895
- Finalizer 183
- Finally 208, 340
- FindByText (Forms-Methode) 559
- FindByValue (Forms-Methode) 559
- Float 155
- Floor (Math, Methode) 251
- Font (Klasse) 997
- FontInfo (Klasse) 737
- FontSize (Aufzählung) 737
- FontUnit (Struktur) 737
- For Each...Next 176
- FOR XML (T-SQL) 422
- For...Next 174
- ForeignKeyConstraint (Klasse) 874
- Format (Methode, String) 258
- Formatieranweisungen 291
- Formatierungen 258
- FormsAuthentication (Klasse) 983
- Formularbasiertes Hochladen von Dateien 530
- Formulare
 - Gestaltung 527
 - Namensraum 505
 - Objekthierarchie 510
 - Verarbeitungszyklus 506
- Framework, Überblick 62
- Framework-Installation 42
- Friend 177
- FromArgb (Color, Methode) 996
- FullName (DirectoryInfo, Eigenschaft) 335

FullName (FileInfo,
Eigenschaft) 339
Function 184
Funktionen in T-SQL 390

G

General-Header-Fields 55
George Boole 161
Gespeicherte Prozeduren
(T-SQL) 435
Parameter 436
Get 188
GetAttributes (File,
Methode) 338
GetCreationTime (Directory,
Methode) 334
GetCreationTime (File,
Methode) 338
GETDATE (T-SQL) 408
GetDirectories (Directory,
Methode) 334
GetDirectories (DirectoryInfo,
Kollektion) 333
GetElementById
(XmlDocument,
Methode) 476
GetElementsByTagName
(XmlDocument,
Methode) 476
GetEnumerator
(Methode) 231
GetFiles (Directory,
Methode) 334
GetFiles (DirectoryInfo,
Methode) 337
GetHashCode (Methode) 156
GetLastAccessTime (File,
Methode) 338
GetLastWriteTime (File,
Methode) 338
GetLength (Array,
Eigenschaft) 218
GetLogicalDrives (Methode,
Directory) 338
GetParent (Directory,
Methode) 335
GetPostBackEventReference
(Page, Methode) 729
GetRedirectUrl (Methode,
Forms-
Authentication) 983
GetType (Methode) 156
GetType (Type) 283
GetTypeInfo (Type) 283
global.asax 808
Prefix 808
global.asax (Applikations-
ereignisse) 805
Globalisierung 294

Globalisierungs-
einstellungen 294
globalization 328
GO (T-SQL) 427
Google (Webservice
nutzen) 1055
GoTo 170
Grabbing 1019
Graphics (Klasse) 996
Gregorianischer Kalender 306
GROUP BY (T-SQL) 395
GROUPING (T-SQL) 400
Groups (Klasse, Regex) 273
Gruppiertes Index 388

H

Haltepunkte setzen 134
Handler, eigene 1030
Handles 179
HasChildNodes (XmlNodeList,
Eigenschaft) 477
HasControls (Methode,
Control/Page) 511
Hashtable (Klasse) 228
Haufen *siehe* Heap
HAVING (T-SQL) 397
Header 759
siehe auch Header
Content-type 759, 763
Location 759
Heap 160
Hebräischer Kalender 307
Hello World 102
Herunterladen von
Dateien 759
Hexadezimalzahlen 158
History-Liste 139
Hochlade-Funktion 703
Hochladen von Dateien 530
Hour (Eigenschaft) 288
HTML Server-
Steuerelemente 504
HTML-Tags suchen 269
HTML Server Controls 504
HtmlContainerControl
(Klasse) 509
HtmlControl (Klasse) 509
HtmlControls
(Namensraum) 503
HtmlGenericControl
(Klasse) 508
HtmlInputButton (Klasse) 529
HtmlInputFile (Steuerelement,
Klasse) 530
HtmlTextWriter (Klasse) 724
HtmlTextWriterAttribute
(Aufzählung) 736
HtmlTextWriterStyle
(Aufzählung) 736

HTTP *siehe* HyperText Transfer
Protocol
HTTP-Handler 1028
ohne IIS 1037
Programmierung 1033
registrieren 1035
vorhandene 1028
HTTP-Kommando 52
HTTP-Kopfzeilen *siehe* Header
HTTP-Message-Header 54
HTTP-Module 1037
HTTP-Statuscodes 53
HttpApplication
(Klasse) 748, 750
HttpApplicationState
(Klasse) 748, 807
HttpBrowserCapabilities
(Klasse) 664
HttpCachePolicy (Klasse) 815
HttpCookie (Klasse) 798
HttpCookieCollection
(Klasse) 797, 801
HttpHandlers 1037
HttpModules 1039
HttpPostedFile (Klasse) 533
HttpServerUtility (Klasse) 754
HttpSessionState (Klasse) 783
HTTP_REFERER 760
HTTP_USER_AGENT 760
HybridDictionary (Klasse) 242
HyperText Transfer
Protocol 51

I

ICollection
(Schnittstelle) 230, 929
IComparer 220
IComparer (Schnittstelle,
Implementierung) 577
IDE 95
IE Webcontrols *siehe*
WebControls 1.0
IEnumerable
(Schnittstelle) 928
Implementierung 577
IEnumerator 218, 231
IEnumerator
(Schnittstelle) 928
If 167
IF ... ELSE (T-SQL) 430
IHttpHandler
(Schnittstelle) 1031
IHttpModule
(Schnittstelle) 1038
Iif 170
IIS
installieren 37
registrieren 42
IIS-Arbeitsverzeichnis 38
IList (Schnittstelle) 929

- ICollection (Schnittstelle) 929
- Image (Klasse, Steuerelement) 992
- Impersonifizierung 977
- Implementierung einer Schnittstelle 930
- Implementierungs-pflichtig 200
- Implements 201
- IN (SQL) 371
- In-Process 781
- Index (SQL) 368
- Indextdienst 937
 - Details 939
 - Suchen im Katalog 947
- IndexOf (Methode, String) 255
- IndexOfAny (Methode, String) 261
- Inherits 194
- Initialisieren 182
- INNER JOIN (SQL) 372
- InnerHTML (Forms-Eigenschaft) 516
- InnerText (Forms-Eigenschaft) 516
- InputStream (Eigenschaft, HtmlInputFile) 534
- Insert (Arraylist, Methode) 225
- Insert (Forms-Methode) 559
- Insert (Hashtable, Methode) 228
- INSERT (SQL) 362
- Insert (StringBuilder, Methode) 267
- INSERT INTO (SQL) 362
- Installation
 - Framework 41
 - IIS 5 37
- Instanzieren 182
- Instanziierung 176
- Integer 155
- Interface 201
- Internet Explorer, Sprache 296
- Interpreter 60
- InvalidConstraintException (Ausnahme) 882
- IOException (Ausnahme) 341
- IsArray (Type) 283
- IsAuthenticated (Eigenschaft, User) 985
- ISBN prüfen 629
- IsByRef (Type) 283
- IsClass (Type) 283
- IsCookieless (Eigenschaft) 783
- IsEnum (Type) 283
- IsLeapYear (Methode) 288
- IsNewSession (Eigenschaft) 784
- ISO 639 298
- ISO 3166 298
- ISO 8859 264
- IsPostBack (Eigenschaft, Page) 526
- IsPrivate (Type) 283
- IsPublic (Type) 283
- IsReadOnly (Eigenschaft) 783
- IsReusable (Eigenschaft, IHttpHandler) 1031
- IsSealed (Type) 283
- IsValueType (Type) 283
- Item (Forms-Eigenschaft) 559
- IUSR_MachineName 975
- IWAM_MachineName 975
- J**
 - JavaScript 526, 617
 - Join (Methode, String) 257
 - JOIN (T-SQL) 418
 - CROSS JOIN (T-SQL) 420
 - FULL OUTER JOIN (T-SQL) 419
 - LEFT JOIN (T-SQL) 419
 - LEFT OUTER JOIN (T-SQL) 419
 - RIGHT JOIN (T-SQL) 419
 - RIGHT OUTER JOIN (T-SQL) 419
- K**
 - Kalender 306
 - Kapselung 177
 - Key (DictionaryEntry, Eigenschaft) 231
 - Keys (Forms-Eigenschaft) 514
 - Klassen 176
 - Sichtbereich 177
 - Klassenansicht 202
 - Klickereignis 179
 - Knotentypen (XPath) 471
 - Kollektionen 223
 - Kommandoschaltflächen 552
 - Kommentare 162
 - Kommentare (T-SQL) 429
 - Kommentare (XSLT) 481
 - Kompilierung, dynamische 1065
 - Komplexe Steuerelemente 635
 - konkret 201
 - Konstanten 161
 - Konstruktoren 193
 - Kontext 773
 - Kontrollkästchen 560
 - Kontroll-Steuerelemente 616
 - Konvertierung von Datenquellen 935
 - Koreanischer Kalender 307
 - Kultur 298
- Kundenspezifische Steuerelemente 706
- Projekte 707
- L**
 - Label (Klasse, Steuerelement) 545
 - LastIndexOf (Methode, String) 255
 - LastIndexOfAny (Methode, String) 261
 - Laufwerke 338
 - Laufwerknamen 338
 - Laufzeitfehler 131
 - Lebenszyklus einer Seite 71
 - Length (Array, Eigenschaft) 218
 - Length (Eigenschaft, String) 254
 - Length (FileInfo, Eigenschaft) 339
 - Length (StringBuilder, Eigenschaft) 267
 - LIKE (SQL) 369
 - ListBox (Klasse, Steuerelement) 557
 - ListDictionary (Klasse) 242
 - ListItem (Klasse, Steuerelement) 557
 - ListItemCollection (Kollektion) 559
 - Literal (Klasse, Steuerelement) 545
 - LiteralControl (Klasse) 518
 - Literale 157
 - Literalzeichen 157
 - Load (XmlDocument, Methode) 476
 - Load (XslTransform, Methode) 486
 - LoadXml (XmlDocument, Methode) 476
 - Location (@OutputCache) 813
 - Lock (Methode) 807
 - Logische Operatoren 166
 - Logische Werte 160
 - Lokalisierung 327
 - Long 155
- M**
 - machine.config 966
 - MailMessage (Klasse) 1015
 - Makrorecorder 141
 - Makros 141
 - Manifest 742
 - MapPath (Server, Methode) 343
 - Match (Klasse, Regex) 273
 - Math (Klasse) 250

- Mathematische Operationen 250
- Max (Math) 251
- Me 195
- MeasureString (Graphics, Methode) 997
- Mehrfachverzweigung (XSLT) 482
- Mehrsprachige Seiten 296
- Mehrsprachigkeit 294
- MemoryStream (Klasse) 489
- Microsoft Indexing Service 937
- Microsoft.IE.WebControls 677
- Millisecond (Eigenschaft) 288
- MIME 763
 - siehe auch* Multipurpose Internet Mail Standard
- MIME-Typ 1006
- Min (Math) 251
- Minute (Eigenschaft) 288
- Missbrauch 793
- Mode (Eigenschaft) 784
- Modularisierungskonzepte 31
- Module 176
- Move (Directory, Methode) 335
- Move (File, Methode) 339
- MoveNext (IEnumerator, Methode) 235
- MoveTo (DirectoryInfo, Methode) 337
- MoveTo (FileInfo, Methode) 340
- MSIDX 937
- MSIL 61
- MultiPage 685
- Multipurpose Internet Mail Standard 54
- MustInherit 200
- MustOverride 199-200
- MyBase 195, 198
- MyClass 199
- N**
- Name (Cookie, Eigenschaft) 801
- Name (Eigenschaft, User) 985
- Name (FileInfo, Eigenschaft) 339
- Name (XmlTextReader, Eigenschaft) 463
- Namensraum 66, 202
 - automatisch importiert 959
 - System 213
- Namensräume, Visual Studio .NET 202
- Namensraumalias (XSLT) 480
- Namensraumhierarchie 202
- Namespace *siehe* Namensräume
- NameValueCollection (Klasse) 247, 761
- Nativer Typ 154
- Negate (Methode, TimeSpan) 291
- Netzwerkzugriff 1019
- New 193
- Next (Methode, Random) 252
- NextBytes (Methode, Random) 252
- NextDouble (Methode, Random) 252
- Nicht gruppierter Index 388
- NodeType (XmlTextReader, Eigenschaft) 463
- Non clustered *siehe* nicht gruppierter Index
- Normalisierung 353
- Normalisierungsregeln 355
- Not 166
- NOT (SQL) 371
- NotInheritable 200
- NotOverridable 199
- Now (Eigenschaft) 285
- O**
- Objekt 64, 176, 182
 - Eigenschaften 65
 - Lebenszyklus 182
 - Methoden 66
- Oktalzahlen 158
- OnEnd (Ereignis, global.asax) 785
- onServerChange (Client-Ereignis) 527
- onServerClick (Client-Ereignis) 527
- OnStart (Ereignis, global.asax) 785
- OOP-Funktionen 194
- Open (File, Methode) 338
- Open (FileInfo, Methode) 340
- OpenRead (File, Methode) 338
- OpenRead (FileInfo, Methode) 340
- OpenText (FileInfo, Methode) 340
- OpenWrite (File, Methode) 338
- OpenWrite (FileInfo, Methode) 340
- Operatoren 162
 - arithmetische 162
 - Bitoperatoren 164
 - logische 166
 - Vergleiche 166
- Zuweisungsoperator 163
- Optimierende Operatoren 166
- Optimierung 146, 820
 - des Datenverkehrs 809
- Optimierungsverfahren 809
- Optionale Parameter 186
- Optionsfelder 560
- Or 166
- ORDER BY (SQL) 368
- ORDER BY (T-SQL), WITH TIES (T-SQL) 401
- OrElse 166
- Out-Of-Process 782
- OutputStream (Response, Eigenschaft) 997
- Overridable 199
- P**
- PadLeft (Methode, String) 260
- PadRight (Methode, String) 260
- Panel (Klasse, Steuerelement) 544
- Parameter 184, 919
 - Optional 186
 - in Prozeduren 919
- Parameterschlange 756
- Parametertypen 283
- Parent (DirectoryInfo, Eigenschaft) 335
- Parent (FileInfo, Eigenschaft) 339
- PartialCaching (Attribut) 814
- Passport 976
- PathTooLongException (Ausnahme) 341
- PATH_TRANSLATED 760
- Pause 753
- Persistierung 826
- Personalisierung 984
- Philosophie 33
- Pi 251
- Placeholder (Klasse, Steuerelement) 545
- PostedFile (Eigenschaft, HtmlInputFile) 533
- PostRequestHandlerExecute (Ereignis) 803
- Pow (Math) 251
- PreRender 75
- PreRequestHandlerExecute (Ereignis) 803
- PreSendRequestContent (Ereignis) 804
- PreSendRequestHeaders (Ereignis) 804
- Private 177
- ProcessRequest (Methode, IHttpHandler) 1031
- Professional Edition 92
- Programmierprinzipien 77

- Programmierung,
 - Grundprinzip 59
- Programmsteuerung 769
- Programmsteuerung
 - (T-SQL) 429
- Projekt
 - anlegen 99
 - Eigenschaften 100
- Projektmappe 98
- Property 187
 - siehe auch* Eigenschaften
- Protected 177
- Protected Friend 177
- Proxy-Klasse 1052
- Proxy-Server 810
- Prozess recyceln 753
- Prüfung der
 - Anmeldeparameter 983
- Public 177
- Pufferspeicher 751

- Q**
- Q306172 140
- QueryString 756
 - Kodierung 758
- QueryString (Request,
 - Kollektion) 756
- QUERY_STRING 760
- Queue (Klasse) 250

- R**
- RadioButtonList (Klasse,
 - Steuerelement) 560
- Random (Klasse) 252
- RangeValidator Control *siehe*
 - Bereichskontroll-
 - Steuerelement
- Rank (Array, Eigenschaft) 218
- RDBMS *siehe* Relationales
 - Datenbankmanagement-
 - system
- Read (XmlTextReader,
 - Methode) 463
- ReadOnly 188
- Redirect (Response,
 - Methode) 768
- RedirectFromLoginPage
 - (Methode, Forms-
 - Authentication) 983
- Referenz 63
- Referenzieren 182
- Referenztypen 160
- Reflektion 1065
- Refresh (DirectoryInfo,
 - Methode) 337
- Refresh (FileInfo,
 - Methode) 340
- Regex (Klasse) 273
- RegexOptions (Aufzählung,
 - Regex) 273
- Regular Expression
 - Validator 627
- Reguläre Ausdrücke 267
 - Grundlagen 274
 - Gruppierung 273
 - Modifikatoren 276
- Regulärer Ausdruck-
 - Steuerelement 627
- Relationales
 - Datenbankmanagement-
 - system 347
- Relationales Modell 354
- Relationen 353
- Relaying prohibited 1016
- ReleaseRequestState
 - (Ereignis) 803
- Rem 162
- REMOTE_ADDR 760
- Remove (ArrayList,
 - Methode) 225
- Remove (Forms-
 - Methode) 518, 559
- Remove (Hashtable,
 - Methode) 228
- Remove (StringBuilder,
 - Methode) 267
- RemoveAt (ArrayList,
 - Methode) 225
- RemoveAt
 - (Forms-Methode) 518
- RemoveAt (Hashtable,
 - Methode) 228
- RemoveHandler 182
- Render 709
- RenderBeginTag (Methode,
 - HtmlTextWriter) 724
- RenderControl 728
- RenderEndTag (Methode,
 - HtmlTextWriter) 724
- RepeatColumns
 - (Forms-Eigenschaft,
 - Listen) 562
- RepeatDirection
 - (Forms-Eigenschaft,
 - Listen) 562
- Repeater, Repeater
 - (Steuerelement) mit
 - XML 459
- RepeatLayout
 - (Forms-Eigenschaften,
 - Listen) 563
- Replace (String, Methode) 260
- Replace (StringBuilder,
 - Methode) 267
- Request (Objekt) 755
- Request-Header-Fields 54
- Request.Form
 - (Steuerelemente) 540
- Request.QueryString 756
- Request.ServerVariables 759
- RequiredFieldValidator
 - (Steuerelement) 618
- ResEditor 313
- Reset (IEnumerator,
 - Methode) 236
- resgen 318
- ResolveRequestCache
 - (Ereignis) 803
- resource-Datei 310
- Response (Eigenschaft) 750
- Response (Objekt) 750
- Response-Header-Fields 54
- Response.BufferOutput 751
- Response.WriteFile 751
- Response.AppendHeader
 - (Methode) 759
- Response.ClearHeaders
 - (Methode) 759
- Response.Redirect 768
- Response.Write 751
- Response.ContentType 763
- Ressource-Dateien, binäre
 - Daten 312
- RessourceManager
 - (Klasse) 319
- Ressourcen 309
- Ressourcen-Datei 309
 - Binäre 319
 - Global Assembly
 - Cache 324
 - Praxistipps 327
 - Stammmname 314
- resx-Format 310
- RETURN (T-SQL) 432, 437
- RETURNS TABLE (T-SQL) 438
- Reverse (Array) 219
- Reverse (ArrayList,
 - Methode) 227
- RFC 822 1007
- RFC 1867 530
- RFC 1945 52
- RGB-Farbraum 158
- Rich Controls *siehe* Komplexe
 - Steuerelemente
- ROLLBACK (T-SQL) 433
- Root (DirectoryInfo,
 - Eigenschaft) 335
- Root (FileInfo,
 - Eigenschaft) 339
- Round (Math) 251
- ROWCOUNT (T-SQL) 401
- RowFilter (Eigenschaft,
 - DataRow) 900
- Rows (Table, Kollektion) 234
- RowStateFilter (Eigenschaft,
 - DataRow) 902

S

- Sal Services 1055
- SANTRA Technology 1054
- SaveAs (Eigenschaft,
 - HtmlInputFile) 533
- SAX *siehe* Simple API for XML
- Schaltflächen 551
- Schaltjahr 288
- Schleifen 171
- Schleifen (XSLT) 483
- Schreibweise 22
- Schriftart 997
- Schrittweise Abarbeitung 136
- Scripting.
 - Dictionary 147
- Scripting.
 - FileSystemObject 147
- SCRIPT_NAME 760
- Second (Eigenschaft) 288
- SecurityException
 - (Ausnahme) 341
- Seitendirektive 955
 - ClientTarget
 - (Attribut) 618
- Seitenobjekte 83
- Seitenreferenzierung 773
- Seitenverarbeitung,
 - Ereignisbehandlungsmethoden 76
- Selbstdefinierte
 - Kontrollelemente 629
- SELECT (SQL) 365
- SELECT (T-SQL), verschachteln
 - siehe* Subselect (T-SQL)
- Select Case 169
- SelectedIndex (DropDownList,
 - Eigenschaft) 235
- SelectNodes (XmlDocument,
 - Methode) 474
- SelectSingleNode
 - (XmlDocument,
 - Methode) 476
- Sendeschaltfläche 555
- Serialisierung 785
- Serializable () (Attribut) 931
- Serializable() 598
- Server (Klasse) 769
- Server (Objekt) 754
- Server Explorer 679
- Server Side Includes 31
- Server.Execute 771
- Server.Transfer 769
- Serverseitige Feldprüfung 617
- Servervariablen 759
- SERVER_NAME 760
- ServiceDescription
 - (Klasse) 1063
- Session-Ereignisse 785
- SessionID (Eigenschaft) 784
- Session_Start (Ereignis) 809
- Set 188
- SetCreationTime (Directory,
 - Methode) 334
- SetCreationTime (File,
 - Methode) 338
- SetDirectory (Directory,
 - Methode) 335
- SetDirectory (File,
 - Methode) 338
- SetLastAccessTime (File,
 - Methode) 338
- SetLastWriteTime (File,
 - Methode) 338
- SetValue (Array,
 - Methode) 218
- Shared 191
- Short 155
- Sicherheit 83, 973
- Sicherheitskonzept 84
 - des IIS 973
- Sichten 423
- Sign (Math) 251
- SignIn (Methode, Forms-
 - Authentication) 983
- Simple API for XML 442
- Simple Mail Transfer
 - Protocol 1007
- Simple Object Access
 - Protocol 1046
 - Datentypen 1047
- Sitzungen
 - Cookies deaktivieren 787
 - Installations-Skript 788
 - InstallSqlState.sql 788
 - SQL Server 788
 - TimeOut 784
- Sitzungs-Cookies 786
- Sitzungsmanagement 779
 - Konfiguration 786
- Sitzungsvariable 790
- Sleep (Methode, Thread) 753
- SmartNavigation 668
 - aktivieren 670
 - AutoPostBack 671
 - Einsatzmöglichkeiten 669
 - nutzen 671
- SMTP *siehe* Simple Mail
 - Transfer Protocol
- SmtMail (Klasse) 1016
- SOAP *siehe* Simple Object
 - Access Protocol
- SOAP-Nachricht 1048
- SocketException
 - (Ausnahme) 1025
- Sockets 1022
- SolidBrush (Klasse) 996
- Sort (Array) 219
- Sort (ArrayList, Methode) 227
- SortedList (Klasse) 236
- sortieren 897
- Sortieren (Arrays) 220
- Sortieren (XSLT) 483
- SOUNDEX (T-SQL) 407
- Spaghetticode 170
- Spalte 351
- Späte Bindung 935
- Split (Methode, String) 257
- Sprachanweisungen,
 - geänderte 146
- Sprachcode 298-299
- Spracheinstellung 294
- Sprachversion 294
- Sprungziel 171
- SQL 347
 - Geschichte 350
 - Integrität 376
- SQL Server
 - Datensatzzeiger 411
 - gespeicherte Prozeduren 435
- SQL Server 2000 373
- SqlCommand (Klasse) 915
- Stack (Klasse) 250
- Standard-Handler 1030
- Standardnamensraum
 - (XSLT) 479
- Standardobjekte 747
- Stapelverarbeitung 250
- Start (Ereignis) 805
- StartsWith (Methode,
 - String) 260
- Statische Mitglieder 191
- Statuscode (HTTP) 53
- Steuerelemente
 - Attribute 736
 - Datenbindung 926
 - erweiterte Entwicklung 723
 - Stile 736
 - Weitergabe 722
 - zusammengesetzte kundenspezifische 707
- Stilattribute 587
- StreamWriter (Klasse) 343
- String 155
- String (Klasse) 254
- StringBuilder (Klasse) 262, 266
- StringDictionary (Klasse) 245
- StringWriter (Klasse) 262, 771
- Structured Query
 - Language 347
- Struktur 204
 - einer Tabelle 884
- Style (Forms-Eigenschaft) 529
- Sub 184
- Subselect (T-SQL) 415
- Substring (Methode,
 - String) 255

- Subtract (Methode, Zeitberechnung) 289
- Suchformular 941
- Switch 170
- Symbole 23
- Syntaxfehler 129
- System.Collections (Namensraum) 223
- System.Collections.Specialized 242
- System.Drawing (Namensraum) 737
- System.Globalization (Namensraum) 294, 304
- System.IO (Namensraum) 330
- System.Math (Klasse) 251
- System.Net (Namensraum) 1019
- System.Object, Methoden 156
- System.Resources (Namensraum) 314
- System.Security (Namensraum) 341
- System.Security.Principal (Namensraum) 987
- System.Text.RegularExpressions (Namensraum) 271
- System.Threading (Namensraum) 753
- System.Web (Namensraum) 747
- System.Web.Mail (Namensraum) 1012
- System.Web.Security (Namensraum) 982
- System.Web.Services.Description (Namensraum) 1063
- System.Web.UI.Control (Namensraum) 539
- System.Web.UI.WebControls (Namensraum) *siehe* WebControls (Namensraum)
- System.Collection.IEnumerator 231
- SystemException 210
- Systemvoraussetzungen 35
- T**
- T-SQL *siehe* Transact-SQL
 - Aggregat-Funktionen 392
 - Datenbank anlegen 382
 - Datenbank löschen 382
 - Datenbank verwenden 383
 - Datenbankdefinition ändern 383
- Datentypen 376
 - benutzerdefinierte 381
 - Binärdaten 377
 - Binärwerte 379
 - Datums- und Zeitwerte 380
 - Numerische Datentypen 378
 - Zeichenketten 376
- Gruppierungen 395
- Indizes 388
- Indizes erzeugen 390
- Mathematische Funktionen 391
- Sicherheitsfunktionen 394
- Systemfunktionen 393
- Tabellen anlegen 384
- Tabellen leeren 388
- Tabellen löschen 387
- Tabellenstruktur ändern 388
- Text- und Bildfunktionen 395
- Tabelle 352
- TableCell (Klasse) 234
- TableRow (Klasse) 234
- TabStrip 683
 - onselectedIndexChanged 684
- TabSeparator 684
- TagName (Seitendirektive, Attribut) 650
- TagPrefix (Seitendirektive, Attribut) 650
- Task-Manager 969
- TCP-Verbindung 1024
- TcpClient (Klasse) 1024
- Templates (XSLT) 480
- Templatesystem 33
- Temporäre Dateien 967
- Text (Forms-Eigenschaft) 541
- Text (XSLT) 480
- TextBox (Klasse, Steuerelement) 548
- TextBox-Eigenschaften 550
- Texteingabefelder 548
- TextWriter (Klasse, abstrakt) 261
- Thread (Klasse) 753
- Throw 211
- Ticks 285
- Ticks (Eigenschaft) 288
- TicksPerHour (Feld, Zeitberechnung) 289
- TicksPerMinute (Feld, Zeitberechnung) 289
- TimeOfDay (Eigenschaft) 286
- TimeOut (Eigenschaft) 784
- Timeout-Wert 753
- TimeSpan (Struktur) 285, 289
- ToCharArray (Methode, String) 256
- Today (Eigenschaft) 285
- ToString (Methode) 291
- ToString (Methode) 291
- ToolBar 680, 699
 - ToolBarButton 700
 - ToolTip 700
- Toolbox 738
 - Drag&Drop-Verhalten 742
- ToolboxBitmap (Attribut) 742
- TOP (T-SQL) 401
- ToShortDateString (Methode) 291
- ToShortTimeString (Methode) 291
- ToString (Datumsformatierung) 291
- ToString (Methode) 156
- Trace (Seitendirektive, Attribut) 956
- Transact-SQL 376
- TRANSACTION (T-SQL) 433
- Transfer (Server, Methode) 769
- TreeView 680, 686
 - Beliebiges XML 694
 - statisch definieren 693
 - universelle Transformation von XML 697
 - XML als Knotenquelle 694
- Trigger (T-SQL) 438
- Trim (Methode, String) 258
- TrimEnd (Methode, String) 258
- TrimStart (Methode, String) 258
- TRUNCATE TABLE (T-SQL) 388
- Try 208, 340
- Type (Klasse) 283
- Typenklassen 159
- Typkonzept 159
- Typsymbole 157
- Typumwandlungen 158
- Typumwandlungen (T-SQL) 404
- Typumwandlungsfunktionen 158
- U**
- Über das Buch 23
- Überladung 186
- Überschreibungspflichtig 200

- UDDI *siehe* Universal
Description, Discovery
and In
- UnauthorizedAccessException
(Ausnahme) 341
- Ungepufferte Ausgaben 804
- Unicode 265
- UNION (T-SQL) 421
- Unit (Struktur) 737
- Universal Description,
Discovery and
Integration 1052
- UnLock (Methode) 807
- Unterstützung für Leser 23
- UPDATE (SQL) 365
- UpdateRequestCache
(Ereignis) 803
- Uplevel 662
- Upload 530, 703
- URL, Daten anhängen 755
- USE (T-SQL) 384
- User (Klasse) 984
- User Controls *siehe*
Benutzer-Steuerelemente
- UserControl (Klasse) 655
- UtcNow (Eigenschaft) 285
- V**
- Validation Controls *siehe*
Kontroll-Steuerelemente
- Validierung 445
- Value (Cookie,
Eigenschaft) 800
- Value (DictionaryEntry,
Eigenschaft) 231
- Value (XmlTextReader,
Eigenschaft) 463
- Variablen 154
- Variablen (T-SQL) 427
- Variablen (XSLT) 483
- Variant 148
- VaryByCustom
(@OutputCache) 813
- VaryByParam
(@OutputCache) 812
- VB 6 149
 - Datum und Uhrzeit 150
 - Mathematische
Funktionen 150
 - Zeichenketten 151
- VB.NET
 - Datentypen 154
 - Zuweisungen 163
- VBScript 143
 - Kompatibilität 147
- Verbindungsablauf 55
- Verbindungsloses Protokoll 52
- Verbotene
 - Dateiendungen 1030
- Vererbung 194
- Vergleichs-Steuerelement 624
- Vergleichsoperatoren 166
- Verknüpfungsoperator 261
- Verzeichnis 335
 - aktuelles 335
 - erzeugen 335
 - lesen 332
 - löschen 335
 - nicht gefunden 341
 - verschieben 335
- Verzeichnis, löschen 335
- Verzeichnis, verschieben 335
- Verzeichniseigenschaften 335
- Verzweigung 165
- Verzweigung (XSLT) 482
- Views *siehe* Sichten
- ViewState *siehe* Anzeigestatus
- Visible (Eigenschaft,
UserControl) 659
- Visible
(Forms-Eigenschaft) 547
- Visual Basic 6 148
- Visual Basic-Kompatibilitäts-
modus 149
- Visual Studio .NET 91
 - Benutzer-Steu-
element 648
 - Code Behind 83
 - Daten manipulieren 923
 - Datenbankassistent 115
 - Datenbankprojekt 114
 - Datenbankverbindung
374, 921
 - Datenbankzugriff 373
 - Debugger aktivieren 131
 - Grafikeditor 739
 - Kommentare 162
 - Programmierung 76, 102
 - Projekte 97
 - SELECT-Abfrage
generieren 923
 - Voraussetzungen 93
 - Webservice 1059, 1078
 - XML-Datei erfassen 457
 - XSD 454
- Visual Studio-Debugger
 - Befehlsfenster 138
 - Einzelschritt 136
 - Haltepunkt 134
 - Haltepunkt-Optionen 135
 - Makros 140
 - Prozedurschritt 136
 - Schnellüberwachung 137
 - Variablenwerte
einsehen 137
- Visual Studio-Debugger 134
- Volltextsuche 937
- W**
- Wartenschlangen-
verarbeitung 250
- Web Forms 501
 - Basiseigenschaften 539
 - Einführung 77
 - Einsatzprinzipien 539
- Web Matrix 45
- Web Server Controls 535
- Web Server-
Steuerelemente 535
- Web Services Description
Language 1050
- Web-Steuerelement-
bibliothek 707
- web.config 962
 - Compileroptionen 966
 - Globalisierungseinstellun-
gen 328
- WebControls
(Namensraum) 536
- WebControls 1.0 676
 - Installation 677
 - MultiPage 685
 - Namensraum 678
 - Registrierung 678
 - TabStrip 683
 - Toolbar 680, 699
 - TreeView 680, 686
- WebException
(Ausnahme) 1022
- WebGrabbing 1020
- WebRequest (Klasse) 1021
- WebResponse (Klasse) 1021
- Webservice (Attribut) 1082
- Webservices anbieten 1076
- Website zum Buch 26
- Websteuerelement-
bibliothek 708
- WebUIValidation.js 622
- Weiterleitung 768
- Wend 146
- Werttypen 160, 283
- WHERE (SQL) 370
- WHILE (T-SQL) 432
- While...End While 172
- WHOIS 1022
- Windows-
 - Authentifizierung 985
 - Anmeldeinforma-
tionen 987
 - einrichten 986
- WITH CUBE (T-SQL) 398
- WITH ROLLUP (T-SQL) 398
- WithEvents 179
- Wohlgeformtheit 444
- Worker Process 969
 - konfigurieren 969-970
- Wörterbücher 236

- Write (Methode,
 - HtmlTextWriter) 724
- WriteAttribute (Methode,
 - HtmlTextWriter) 724
- WriteEndElement
 - (XmlTextWriter, Methode) 466
- WriteEndTag (Methode,
 - HtmlTextWriter) 724
- WriteFullBeginTag (Methode,
 - HtmlTextWriter) 724
- WriteLine (Methode,
 - HtmlTextWriter) 724
- WriteLine (StreamWriter, Methode) 343
- WriteOnly 188
- WSDL *siehe* Web Services
 - Description Language programmtechnisch auswerten 1064
- wsdl.exe 1058
- X**
- XDR *siehe* XML Data Reduced
- XMethod 1054
- XML
 - Datenerfassung 457
 - DOM 443
 - filtern und sortieren 461
 - mit .NET verarbeiten 461
 - Navigieren ohne XPath 476
 - Praxis 454
 - Pull 443
 - Push 442
 - Schema zuordnen 457
 - Verarbeitung 458
 - Verarbeitungsprinzipien 442
- XML Data Reduced 449
- XML Schema Definition Language 449
- Attribute 453
- Datentypen 453
- Einführung 450
- Elemente 452
- Grundaufbau 450
- Typdefinition 451
- Visual Studio .NET 455
- XML-Grundlagen 443
- Grammatik 449
- Inhalt beschreiben 447
- Struktur entwerfen 446
- XML-Protokolle 1046
- XML-Schema 455
- XmlDocument (Klasse) 474
- XmlNode (Klasse) 476
- XmlNodeList (Klasse) 475, 477
- XmlNodeReader (Klasse) 462
- XmlTextReader (Klasse) 462
- XmlTextWriter (Klasse) 464
- XmlValidatingReader (Klasse) 462
- XP *siehe* XML-Protokolle
- XPath 470
 - Achsenbezeichner 472
- XSD *siehe* XML Schema
 - Definition Language
- xsl:apply-template (XSLT) 481
- xsl:choose (XSLT) 482
- xsl:for-each (XSLT) 483
- xsl:if (XSLT) 482
- xsl:include (XSLT) 485
- xsl:number (XSLT) 484
- xsl:output (XSLT) 485
- xsl:sort (XSLT) 483
- xsl:template (XSLT) 480
- xsl:variable (XSLT) 484
- XSLT 478
 - .NET 485
 - Ausgabeformatierung 490
 - Basisregeln 479
 - Einführung 479
 - Fallback 481
 - Funktionen 485
 - Knoten kopieren 484
 - Kommentare 481
 - Mehrfachverzweigung 482
 - Parameter 491
 - Programmieren 482
 - Schleifen 483
 - Sortieren 483
 - Templates 480
 - Templates (XSLT) 480
 - Text 480
 - Variablen 483
 - Verzweigung 482
- XsltArgumentList (Klasse) 495
- XsltTransform (Klasse) 485, 488
- Z**
- Zeichencodierung 263
- Zeichenketten 254
 - Methoden 255
- Zeichenketten-Funktionen (T-SQL) 405
- Zeichenkodierung
 - ASCII 266
 - Unicode 266
 - UTF7 266
 - UTF8 266
- Zeichenwerkzeuge 741
- Zeit 284
- Zeitdarstellungen 285
- Zeitüberschreitung 788
- Zeitzone 285
- Zero (Feld,
 - Zeitberechnung) 289
- Zufallsgenerator 991
- Zufallszahlen 252
 - Startwert 253
- Zugriff, anonym 976
- Zugriffsebene 283
- Zugriffsmodifizierer 177



... aktuelles Fachwissen rund
um die Uhr – zum Probelesen,
Downloaden oder auch auf Papier.

www.InformIT.de

InformIT.de, Partner von **Addison-Wesley**, ist unsere Antwort auf alle Fragen der IT-Branche.

In Zusammenarbeit mit den Top-Autoren von Addison-Wesley, absoluten Spezialisten ihres Fachgebiets, bieten wir Ihnen ständig hochinteressante, brandaktuelle Informationen und kompetente Lösungen zu nahezu allen IT-Themen.

The image displays three overlapping screenshots of the InformIT website interface. The top-left screenshot shows a search bar with 'GO' and a list of themes under 'Themenauswahl'. The top-right screenshot shows a 'Buchtipps' section with a book titled 'VB.NET' priced at 49,95 EUR. The bottom-left screenshot shows a 'Meine ganz persönliche Bibliothek' section with a list of books and a 'Diskussionsforum'. The bottom-right screenshot shows a 'Halo und Herzlich Willkommen bei InformIT' section with a login form and a 'Mitglied werden' button. The bottom-most screenshot shows a 'wenn Sie mehr wissen wollen ...' banner with the website URL.

wenn Sie mehr wissen wollen ... **www.InformIT.de**



Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt. Dieses eBook stellen wir lediglich als persönliche Einzelplatz-Lizenz zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschliesslich

- der Reproduktion,
- der Weitergabe,
- des Weitervertriebs,
- der Platzierung im Internet, in Intranets, in Extranets,
- der Veränderung,
- des Weiterverkaufs
- und der Veröffentlichung

bedarf der schriftlichen Genehmigung des Verlags.

Insbesondere ist die Entfernung oder Änderung des vom Verlag vergebenen Passwortschutzes ausdrücklich untersagt!

Bei Fragen zu diesem Thema wenden Sie sich bitte an: info@pearson.de

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf unseren Websites ist eine freiwillige Leistung des Verlags. Der Rechtsweg ist ausgeschlossen.

Hinweis

Dieses und viele weitere eBooks können Sie rund um die Uhr und legal auf unserer Website



herunterladen