

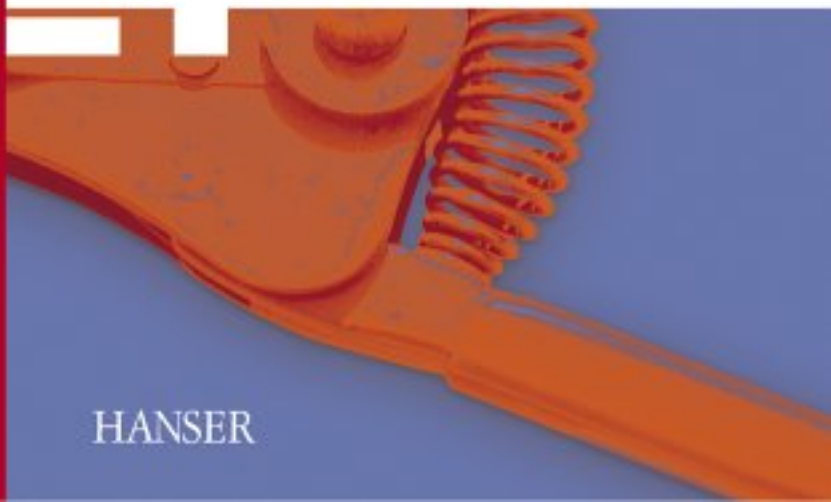


www.asp-buch.de

Patrick A. Lorenz

Kochbuch

ASP.NET



HANSER

Patrick A. Lorenz

ASP.NET

Kochbuch

HANSER

Der Autor:
Patrick A. Lorenz, Ihringen
www.asp-buch.de

Medienproduzent, Hensle CrossMedia GmbH, Freiburg
www.hensle-crossmedia.de

<http://www.hanser.de>

Alle in diesem Buch enthaltenen Informationen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht, auch nicht für die Verletzung von Patentrechten, die daraus resultieren können.

Ebenso wenig übernehmen Autor und Verlag die Gewähr dafür, dass die beschriebenen Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt also auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation
ist bei Der Deutschen Bibliothek erhältlich.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2002 Carl Hanser Verlag München Wien
Gesamtlektorat: Fernando Schneider
Copy-editing: Sandra Gottmann, Bonn
Herstellung: Monika Kraus
Datenbelichtung, Druck und Bindung: Kösel, Kempten
Printed in Germany

ISBN 3-446-22235-9



Inhalt

1	Einführung	18
1.1	Wer dieses Buch lesen sollte.....	18
1.2	Was dieses Buch leisten kann	18
1.3	Was dieses Buch nicht leisten kann	19
1.4	Wie Sie mit diesem Buch arbeiten.....	19
1.5	Die mitgelieferten Beispiele.....	20
1.6	Website zum Buch.....	21
1.7	Kontakt zum Autoren.....	21
1.8	Die Zukunft dieses Buches	21
	Teil I – Rezepte	23
2	Basics.....	26
2.1	... Debugging und Tracing nutzen?	26
2.2	... Debugging und Tracing mit Visual Studio .NET nutzen?	29
2.3	... die Kommunikation zwischen Client und Server belauschen?	30
2.4	... alle vorhandenen Kopfzeilen und Server-Variablen ausgeben?.....	32
2.5	... eine umfangreiche Website strukturieren?.....	34
2.6	... eine globale Seitenvorlage erstellen?	35
2.7	... eine zentrale Fehlerbehandlung für alle Seiten entwickeln?	40
2.8	... überprüfen, ob Cookies akzeptiert werden?	43
2.9	... überprüfen, ob JavaScript aktiviert ist?.....	44
2.10	... die Möglichkeiten des Client-Browsers ermitteln?	46
2.11	... Seiten für unterschiedliche Browser optimieren?	48
2.12	... Frames in ASP.NET verwenden?	50
2.13	... einen Redirect durchführen?	52
2.14	... einen Redirect für einen anderen Frame durchführen?	53
2.15	... einen Redirect serverseitig durchführen?.....	55
2.16	... den Namen und die Adresse der aktuellen Seite ermitteln?	57
2.17	... die zuvor aufgerufene Seite ermitteln	59

2.18	... die Ausführung einer Seite verzögern?	60
2.19	... eine Seite automatisch neu laden?	61
2.20	... einen beliebigen Inhalt an einer festgelegten Position ausgeben?	62
2.21	... Meta-Tags dynamisch übergeben?	64
2.22	... das Euro-Zeichen € im Browser ausgeben?	69
2.23	... die Sprache des Besuchers erkennen?	72
2.24	... eine eindeutige ID erstellen?	74
2.25	... feststellen, ob der Benutzer noch verbunden ist?	76
2.26	... einen Broken Link abfangen?	77
2.27	... Broken-Links finden und beseitigen?	79
2.28	... Broken Links bei Bildern verhindern?	81
2.29	... eine Liste aller Content-Types finden?	83
2.30	... die Standard-Programmiersprache ändern?	85
2.31	... mehrere Sprachen innerhalb einer ASP.NET-Seite verwenden?	87
2.32	... eine Code Behind-Datei kompilieren?	88
2.33	... alle Seiten automatisch von einer zentralen Code Behind-Klasse ableiten?	94
2.34	... die standardmäßig importierten Namespaces erweitern?	97
2.35	... die standardmäßig gesetzten Optionen beim Kompilieren verändern?	100
2.36	... eine Assembly aus dem Global Assembly Cache einbinden?	101
2.37	... eine DLL im Global Assembly Cache ablegen?	103
2.38	... den gesamten Query-String abfragen?	108
3	Sprachelemente	110
3.1	... eine sichere Typenkonvertierung durchführen?	110
3.2	... den Typ einer Variablen ermitteln?	111
3.3	... abfragen, ob ein Wert in einem Array vorhanden ist?	112
3.4	... ein beliebiges Array sortieren?	114
3.5	... ein Array inline erstellen?	118
3.6	... eine if-Abfrage inline durchführen?	119
3.7	... einer Methode eine unbekannte Anzahl von Parametern übergeben?	122
3.8	... eine Entsprechung zum VB-Schlüsselwort With in C# verwenden?	124
3.9	... einzelne Werte einer Enumeration abfragen?	127
3.10	... einen Enumerationswert aus einer Zeichenkette ermitteln?	129
3.11	... alle Werte einer Enumeration auflisten?	130
3.12	... alle Eigenschaften eines Objekts abfragen?	131
3.13	... eine Eigenschaft per Reflection abfragen oder setzen?	133
3.14	... eine indizierte Eigenschaft per Reflection abfragen oder setzen?	136
3.15	... eine Methode per Reflection aufrufen?	138
3.16	... Detailinformationen über eine Komponente ermitteln?	140
3.17	... eine eigene Collection erstellen?	142
3.18	... ein eigenes Dictionary erstellen?	145

3.19	... Hilfsfunktionen global hinterlegen?.....	147
3.20	... innerhalb einer beliebigen Klasse auf die ASP.NET-Objekte zugreifen?.....	150
4	Zeichenketten	154
4.1	... überprüfen, ob eine Zeichenkette einen numerischen Wert enthält?	154
4.2	... überprüfen, ob eine Zeichenkette einen Datumswert enthält?	156
4.3	... zwei Teilzeichenketten vertauschen?.....	157
4.4	... die ersten x Wörter einer Zeichenkette abfragen?.....	158
4.5	... die Zeichen einer Zeichenkette iterieren?	160
4.6	... ermitteln, um was für eine Art von Zeichen es sich handelt?	163
4.7	... das Vorkommen eines Zeichens in einer Zeichenkette zählen?.....	164
4.8	... die Anzahl Wörter in einer Zeichenkette zählen?	166
4.9	... eine Zeichenkette aus wiederholenden Zeichen zusammensetzen?	167
4.10	... eine Zeichenkette aufsplitten?.....	168
4.11	... eine Zeichenkette zusammenführen?	169
4.12	... eine Zahl in ein Wort umwandeln?	171
4.13	... reguläre Ausdrücke verwenden?	173
4.14	... Zeichen mit regulären Ausdrücken ersetzen?	173
5	Mathematik und Berechnungen	178
5.1	... die aktuelle Uhrzeit ermitteln?.....	178
5.2	... eine tageszeitabhängige Begrüßung anzeigen?	181
5.3	... den aktuellen Wochentag ermitteln?.....	181
5.4	... die Kalenderwoche eines Tages ermitteln?.....	183
5.5	... die Anzahl der Tage eines Monats abfragen?	186
5.6	... ermitteln, ob es sich um ein Schaltjahr handelt?	187
5.7	... das Datum der letzten Änderung ausgeben?	189
5.8	... Fahrenheit in Celsius umrechnen und umgekehrt?	190
5.9	... eine Zahl runden?.....	192
5.10	... die nächste Ganzzahl ermitteln?	193
5.11	... die kleinere/größere zweier Zahlen ermitteln?.....	194
5.12	... gerade/ungerade Zahlen ermitteln?	195
6	Eingabeformulare und Web Controls.....	198
6.1	... die Reihenfolge von Eingabefeldern festlegen?.....	198
6.2	... eine Benutzereingabe erzwingen?.....	199
6.3	... überprüfen, ob ein numerischer Wert eingegeben wurde?.....	200
6.4	... überprüfen, ob ein korrektes Datum eingegeben wurde?.....	202
6.5	... überprüfen, ob ein korrektes Datum in der Zukunft angegeben wurde?.....	203
6.6	... eine Email-Adresse syntaktisch überprüfen?	205

6.7	... eine CheckBox validieren?	206
6.8	... überprüfen, ob eine Eingabe einem individuellen Schema entspricht?	210
6.9	... ein Eingabefeld schreibschützen?	215
6.10	... eine mehrstufige Auswahlliste realisieren?	216
6.11	... eine Liste von Ländern anzeigen?	219
6.12	... ein einfaches Kontaktformular erstellen?	221
6.13	... ein Formular mit Datei-Upload erstellen?	225
6.14	... eine Email-Vorlage erstellen?	234
6.15	... mehrere Felder in einer DropDownList oder ListBox anzeigen?	238
6.16	... AutoPostBack und SmartNavigation parallel benutzen	240
6.17	... eine MessageBox im Browser ausgeben?	243
6.18	... eine JavaScript-Funktion vor dem Absenden eines Formulars ausführen?	245
6.19	... das Absenden eines Formulars bestätigen lassen?	248
6.20	... Seitenvariablen über einen PostBack hinweg speichern?	250
6.21	... Web Controls zur Laufzeit rendern?	253
6.22	... ein formatiertes Eingabefeld anbieten?	254
6.23	... eine sortierte ListBox anzeigen?	256
6.24	... einen Listeneintrag mit einem bestimmten Wert selektieren?	262
6.25	... verhindern, dass die Selektion beim PostBack zurückgesetzt wird?	263
6.26	... verhindern, dass ein File-Upload immer null ergibt?	266
6.27	... eine TextBox mit einem Farbverlauf versehen?	268
6.28	... die Zeichenlänge einer TextBox limitieren?	270
6.29	... Zahlen und Dati in Vorlagen formatieren?	271
6.30	... die Datensatz-ID unsichtbar in einem DataGrid-Control mitführen?	273
6.31	... die automatisch erstellten Spalten eines DataGrid-Controls anpassen?	276
6.32	... eine datengebundene DataGrid-Spalte individuell formatieren?	279
6.33	... mehrere Felder in einer DataGrid-Spalte anzeigen?	281
6.34	... auf Click-Ereignisse in einem DataGrid-Control reagieren?	285
6.35	... die Selektion eines DataGrid-Controls explizit aufheben?	288
6.36	... ein editierbares DataGrid-Control erstellen?	291
6.37	... auf die TextBox-Elemente eines DataGrid-Controls zugreifen?	294
6.38	... das Eingabefeld eines editierbaren DataGrid-Controls verändern?	299
6.39	... die Eingaben eines editierbaren DataGrid-Controls validieren?	301
6.40	... einen booleschen Wert in einem DataGrid-Control editieren?	303
6.41	... dem DataGrid-Control dynamisch Spalten anfügen?	313
6.42	... neue Spaltentypen für das DataGrid entwickeln?	317
6.43	... ein DataGrid sortieren?	324
6.44	... ein DataGrid automatisch seitenweise darstellen?	327
6.45	... ein DataGrid manuell seitenweise darstellen?	330

7	User Controls.....	336
7.1	... Werte an ein User Control übergeben?	336
7.2	... den Status eines User Controls speichern?	340
7.3	... ein User Control mit Code Behind verwenden?	344
7.4	... Eingabefelder in User Controls verwenden?	350
7.5	... ein User Control mit Ereignissen ausrüsten?	351
7.6	... ein komplexes Custom Control erstellen?	358
7.7	... ein User Control dynamisch platzieren?	364
7.8	... ein User Control dynamisch laden?	367
7.9	... ein Template dynamisch laden?	374
7.10	... ein Control aus einer Zeichenkette parsen?	377
7.11	... ein User Control direkt im Browser anzeigen?	381
8	Datenbanken.....	384
8.1	... ein strongly typed DataSet anlegen?	384
8.2	... auf eine System-DSN zugreifen?	388
8.3	... eine Connection lokal und beim Provider verwenden?	392
8.4	... das Schema einer Tabelle abfragen?	395
8.5	... per SQL eine neue Tabelle anlegen?	397
8.6	... per SQL eine neue Spalte anlegen oder ändern?	399
8.7	... per SQL einen neuen Datensatz anlegen?	402
8.8	... die automatische ID des zuletzt eingefügten Datensatzes abfragen?	405
8.9	... per SQL einen bestehenden Datensatz aktualisieren?	407
8.10	... per SQL einen Datensatz löschen?	410
8.11	... die Anzahl von Datensätzen in einer Tabelle ermitteln?	412
8.12	... ermitteln, ob und, wenn ja, wie viele Datensätze geändert wurden?	414
8.13	... SQL Injection verhindern?	415
8.14	... Datensätze sortieren?	419
8.15	... Relationen zwischen Tabellen herstellen?	423
8.16	... einen hierarchischen Datensatz über das DataGrid-Control anzeigen?	426
8.17	... zwei relational verknüpfte Datensätze neu anlegen?	430
8.18	... Daten in einem DataSet filtern?	433
8.19	... ein Memo-Feld im Browser darstellen?	438
8.20	... einen zufälligen Datensatz abfragen?	441
9	Dateisystem.....	448
9.1	... einen virtuellen Pfad in einen physikalischen umwandeln?	448
9.2	... das aktuelle physikalische Verzeichnis ermitteln?	449
9.3	... eine Verzeichnisliste erstellen?	450
9.4	... alle vorhandenen Dateien anzeigen?	453

9.5	... mit Wildcards ausgewählte Dateien anzeigen?.....	456
9.6	... eine beliebige Datei zum Client senden?	457
9.7	... überprüfen, ob ein Verzeichnis oder eine Datei existiert?	458
9.8	... eine Datei umbenennen?.....	459
9.9	... eine Textdatei auslesen?	460
9.10	... eine Textdatei erstellen oder ergänzen?.....	462
9.11	... eine binäre Datei auslesen?.....	463
9.12	... eine binäre Datei erstellen oder ergänzen?	465
9.13	... Änderungen im Dateisystem überwachen?.....	466
9.14	... auf eine ZIP-Datei zugreifen?.....	470
9.15	... den kurzen DOS-Dateinamen abfragen?	482
9.16	... einen eindeutigen, temporären Dateinamen erstellen?	483
9.17	... einzelne Elemente einer Pfadangabe ermitteln?	485
9.18	... zwei Pfadelemente verbinden?	485
9.19	... ein Verzeichnis samt Inhalt löschen?.....	486
9.20	... die Version einer Datei ermitteln?	487
9.21	... eine Datei exklusiv sperren?	489
10	Sessions	492
10.1	... globale Benutzerinformationen als Session-Variablen realisieren?.....	492
10.2	... eine Session ohne Cookies erzeugen?.....	498
10.3	... Session-Daten im State Service speichern?	499
10.4	... Session-Daten im SQL-Server speichern?.....	502
10.5	... ein DataSet im Session-Scope ablegen?	503
10.6	... eigene Strukturen und Klassen mit Session-Scope ablegen?.....	506
10.7	... ein Objekt individuell serialisieren?	509
11	Sicherheit.....	512
11.1	... feststellen, ob es sich um eine sichere Verbindung handelt?	512
11.2	... die Bit-Stärke der SSL-Verschlüsselung	514
11.3	... die Email-Adresse des Benutzers verifizieren?	514
11.4	... ein Passwort erstellen?.....	515
11.5	... einen Hashcode erstellen?.....	520
11.6	... Forms Authentication mit Window-Benutzerdaten überprüfen?.....	522
11.7	... Impersonation mit Forms Authentication nutzen?.....	524
11.8	... alle Dateien mit Forms Authentication schützen?	526
11.9	... Download-Dateien vor dem direkten Zugriff schützen?.....	528
11.10	... Links von bestimmten Seiten verhindern?.....	531
11.11	... einen Stream oder eine Datei verschlüsseln?.....	535
11.12	... eine Email verschlüsseln?.....	540

12	Grafik.....	546
12.1	... ein Bild einladen und im Browser ausgeben?	546
12.2	... Bildformate konvertieren?	548
12.3	... die Größe und Auslösung eines Bildes ermitteln?	548
12.4	... automatisch Thumbnails von Bildern erzeugen?	549
12.5	... Thumbnails automatisch erstellen und zwischenspeichern?	550
12.6	... Bilder von Benutzern uploaden und ablegen?.....	553
12.7	... ein Bild rotieren?.....	555
12.8	... dynamisch Grafiken erstellen?.....	556
12.9	... grafischen Text ausgeben?	558
12.10	... grafischen Text formatieren?	561
12.11	... ein zufälliges Bild anzeigen?	565
13	System und Netzwerk	570
13.1	... eine COM-Komponente in ASP.NET verwenden?.....	570
13.2	... eine Win32 API-Funktion aufrufen?.....	573
13.3	... ein Programm auf dem Server starten?	576
13.4	... eine Übersicht der Server-Prozesse anzeigen?.....	578
13.5	... eine Übersicht der Server-Services anzeigen?	582
13.6	... einen Service starten oder beenden?	584
13.7	... das Event-Log im Browser anzeigen lassen?.....	589
13.8	... Ereignisse im Event-Log protokollieren?	595
13.9	... den Computernamen des Servers ermitteln?.....	596
13.10	... das Windows-Verzeichnis herausfinden?	596
13.11	... ermitteln, wie lange der Server schon läuft?	598
13.12	... eine Umgebungsvariable abfragen?	599
13.13	... die Version des Betriebssystems abfragen?	601
13.14	... die Version der Common Language Runtime (CLR) abfragen?.....	602
13.15	... die Version der verwendeten Internet Information Services abfragen?	603
13.16	... die IIS vor Viren und Eindringlingen schützen?	604
13.17	... festlegen, wann die ASP.NET-Engine neu gestartet wird?.....	606
13.18	... die ASP.NET-Engine (neu) registrieren?.....	607
13.19	... ASP.NET auf einem anderen Web-Server als den IIS verwenden?.....	608
13.20	... die IP-Adresse des Benutzers ermitteln?.....	609
13.21	... einen Host-Namen auflösen?	610
13.22	... eine Datei serverseitig von einem anderen Server herunterladen?.....	611
14	XML Web Services	616
14.1	... einen Web Service erstellen?	616
14.2	... einen Web Service mit Visual Studio .NET konsumieren?	619

14.3	... einen Web Service ohne Visual Studio .NET konsumieren?.....	622
14.4	... einen Web Service ohne Round Trip per JavaScript abfragen?.....	625
14.5	... einen Web Service mit Code Behind erzeugen?.....	627
14.6	... Session-Daten in einem Web Service verwenden?.....	628
14.7	... Sessions ohne Cookie verwenden?.....	630
14.8	... einen Counter als Web Service realisieren?.....	632
14.9	... einen mit Passwort geschützten Web Service erstellen?.....	635
14.10	... auf einen mit Passwort geschützten Web Service zugreifen?.....	637
14.11	... einen Web Service per HTTPS/SSL aufrufen?.....	639
14.12	... komplexe Datentypen in einem Web Service verwenden?.....	639
14.13	... binäre Daten mit einem Web Service übertragen?.....	647
14.14	... Bilder von einem Web Service generieren lassen?.....	649
14.15	... die automatische asmx-Hilfeseite verändern?	655
14.16	... einen Web Service im UDDI-Verzeichnis finden?.....	656

Teil II – Lösungen 661

15 Content-Management..... 664

15.1	... ein Excel-Sheet dynamisch erstellen?.....	664
15.2	... eine dynamisch erstellte PDF-Datei im Browser anzeigen?.....	670
15.3	... Inhalte anderer Seiten scrapen?	676
15.4	... eine Tabelle parsen?	679
15.5	... einen Linkzähler einrichten?.....	683

16 Community..... 688

16.1	... ein News-System entwickeln?.....	688
16.2	... neue Inhalte seit dem letzten Besuch markieren?.....	696
16.3	... eine Newsletter-An- und Abmeldung realisieren?.....	701
16.4	... eine Link-Sammlung erstellen?	719
16.5	... einen zufälligen Link anzeigen?	725
16.6	... eine elektronische Postkarte versenden?.....	727
16.7	... ein Gästebuch erstellen?	739
16.8	... ein Forum erstellen?	746

17 E-Commerce..... 764

17.1	... eine hierarchische Produktübersicht erzeugen?	764
17.2	... ermitteln, ob ein Land zur EU gehört?.....	769
17.3	... ermitteln, ob ein Land den Euro als Zahlungsmittel verwendet?.....	771
17.4	... ermitteln, ob ein Land umsatzsteuerpflichtig ist?	771
17.5	... Preise mit und ohne Umsatzsteuer anzeigen?	774

17.6	... eine Umsatzsteuer-ID syntaktisch überprüfen?.....	775
17.7	... eine Umsatzsteuer-ID online beim nationalen Finanzamt überprüfen?	779
17.8	... aktuelle Währungsinformationen erhalten?	785
17.9	... Preise in verschiedenen Währungen anzeigen?	791

Index		795



Vorwort

Obwohl .NET noch eine recht junge Technologie ist, erscheint eine Betrachtung ihrer Entwicklung seit dem „Making public“ vor knapp zwei Jahren schon jetzt interessant. Während in den verschiedenen Beta-Phasen fast ausschließlich wissenschaftlich mit der Thematik umgegangen wurde, sind seit dem Release im Januar 2002 nun tatsächlich erste für die Öffentlichkeit bestimmte Projekte in Arbeit. Das Vertrauen der Entwickler ist deutlich gestiegen, und auch zu den Entscheidungsträgern sind die neuen Möglichkeiten ganz offensichtlich vorgedrungen.

ASP.NET hat sich bezogen auf das gesamte Technologiespektrum von .NET zu einem klaren Vorreiter entwickelt. Die Gründe liegen auf der Hand. Während Desktop-Produkte davon abhängig sind, dass auf dem Zielsystem das Framework installiert ist, lassen sich Web-Applikationen autark entwickeln. Nur auf dem Server selbst muss das Framework vorhanden sein. Der ungleich größere Technologieschub, den die Internetentwicklung für sich verbuchen konnte, spielt vermutlich eine ebenso wichtige Rolle. Auch wenn Quick'n'Dirty-Lösungen mit alten ASP-Versionen zur Freude des Kunden schnell realisiert waren, können sie mit den Leistungen der neuen Version nicht im Ansatz Schritt halten.

Die offizielle deutsche ASP.NET-Newsgroup spiegelt diese Bild sehr deutlich wider. Längst ist die Überzahl der theoretischen Fragen den klar praxisrelevanten Detailthemen gewichen. Es geht nicht mehr darum, die Technologie nur zu verstehen, sondern diese im alltäglichen Arbeitsgebiet sinnvoll und effizient einzusetzen. Diesen Zustand belegen nicht nur die Fragen, sondern auch die qualifizierten Antworten.

Das vor Ihnen liegende Buch ist die logische Fortführung dieses Umschwungs. Es wurde in einem sehr engen und beständigen Kontakt zu den Entwicklern „an der Front“ geschrieben und beantwortet deren Fragen. Es geht dabei um eines und nur um eines: die Praxis. In weit über 250 Rezepten gibt es Antworten auf die meistgestellten Fragen und wiederkehrende Probleme, die sich einem Web-Entwickler bei der Verwendung von ASP.NET stellen. Jedes Rezept beginnt mit der Frage „Wie kann ich ...“ und endet mit einer differenzierten Lösung im Kontext von ASP.NET.

Dieses Buch wäre nicht ohne die direkte und indirekte, die bewusste und unbewusste Hilfe aller derer möglich gewesen, die ihre täglichen Anliegen zu ASP.NET in öffentlichen Foren formuliert haben. Es wäre nicht denkbar gewesen ohne die vielen direkten Mailwechsel zu einzelnen Themen. Und schließlich hätte es nicht so viel Spaß gemacht ohne das positive wie negative, immer aber gut gemeinte Feedback vieler Leser meiner bisherigen Bücher. Ich möchte mich bei all diesen Helfern bedanken!

Danke darüber hinaus an meinen Lektor Fernando Schneider, der meinen Heißhunger nach diesem Buch mit einem Augenzwinkern begrüßt hat und der mich immer aufgebaut hat, wenn Motivation Not tat. Danke an die Herstellerin Monika Kraus und an Brigitte Aurnhammer vom Carl Hanser Verlag. Danke an Sandra Gottmann, ohne deren Copy-Editing das Buch ein gefundenes Fressen für meinen ehemaligen Deutsch-LK-Lehrer geworden wäre. Danke an meinen bisherigen Arbeitgeber für die Möglichkeit, jahrelange Praxiserfahrungen mit ASP zu sammeln. Danke an meinen neuen Arbeitgeber für die Möglichkeit, diese Entwicklung auch in der Zukunft fortführen zu können.

Danke an meine Familie.

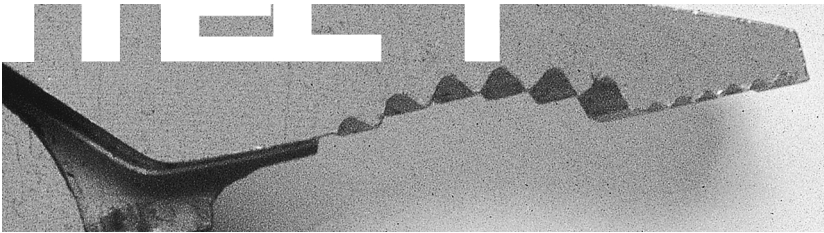
Viel Spaß beim Köcheln!

Patrick A. Lorenz

Ihringen, im August 2002

1

Einführung



Hier erfahren Sie alles, was Sie schon
immer über dieses Buch wissen wollten,
aber nie zu fragen gewagt haben.

1 Einführung

Hallo Welt! In diesem Kapitel finden Sie einführende Hinweise über das Wer, Wie und Was dieses Buches. Ich empfehle Ihnen, die folgenden Abschnitte unbedingt zu lesen; sie sind extra kurz gehalten, liefern jedoch einige wichtige Informationen über das Buch und den Umgang damit. Sollten Sie diesen Berg Papier gerade in der Buchhandlung Ihres Vertrauens unentschlossen in den Händen halten, lesen Sie bitte unbedingt zumindest die Überschriften eins bis drei!

1.1 Wer dieses Buch lesen sollte

Sie! Zumindest wenn Sie mit ASP.NET produktiv arbeiten und auf bestehende Erfahrungen im praktischen Einsatz der neuen Technologie zurückgreifen wollen. Sie sollten wenigstens theoretische Erfahrung mit ASP.NET gemacht haben, die einzelnen Objekte kennen und die allgemeinen Zusammenhänge verinnerlicht haben. Erfüllen Sie diese Voraussetzungen, können Sie ab sofort auf mehrere hundert Seiten absolutes Praxiswissen zurückgreifen und ASP.NET im Live-Einsatz erleben.

1.2 Was dieses Buch leisten kann

Das vor Ihnen liegende Werk enthält weit über 250 Rezepte. Sie erfahren in diesen zwar nicht, wie Sie ein Ei ohne Wasser, Topf und Strom hart kochen, dafür aber alles, was Sie über ASP.NET wissen sollten.

Jedes Rezept stellt in der Überschrift eine aus der Praxis stammende Frage in den Raum. Wie kann ich ...? Das Rezept beschreibt den Hintergrund der Problematik und stellt diese in den Kontext gängiger Projektarbeiten. Problem erkannt, Problem gebannt! Die sich anschließende Lösung betrachtet die Fragestellung zumeist aus unterschiedlichen Standpunkten und liefert dazu individuelle Ansatzpunkte. Jede Möglichkeit wird in Form eines direkt lauffähigen Beispiels vorgestellt und das Ergebnis wann immer sinnvoll abgebildet.

Viele Rezepte enthalten zusätzliche Tipps und Tricks aus der Praxis, die über den Tellerrand der aktuellen Fragestellung hinaus blicken und sich anschließend ergebende Stolpersteine aus dem Weg räumen. Weitergehende Empfehlungen runden die Praxistauglichkeit ab.

1.3 Was dieses Buch nicht leisten kann

Das häufigste in diesem Buch benutzte Wort lautet „Praxis“. Warum? Na ja, weil es hier eben um die Praxis geht. Folgerichtig hat die Theorie kaum Platz. Für das Verständnis der Beispiele benötigen Sie daher allgemeine Kenntnisse im Umgang mit ASP.NET. Ein Grundlagenkapitel und eine allgemeine Einführung in die Technologie hätte schlicht keinen Platz. Sollten Sie jedoch an einer solchen interessiert sein, verweise ich gerne auf zwei andere Bücher aus meiner Feder:

- ASP.NET Grundlagen und Profiwissen, 1248 Seiten, 59,90 €, Mai 2002, Hanser Verlag München Wien, ISBN 3-446-21943-9
- .net shortcut ASP.NET, 224 Seiten, 24,90 €, September 2002, Hanser Verlag München Wien, ISBN 3-446-22129-8

Während das erste Buch alle Aspekte rund um ASP.NET behandelt und alle Grundlagen erklärt, bietet das zweite eine Zusammenfassung der Technologie.

1.4 Wie Sie mit diesem Buch arbeiten

Dieses Kochbuch ist kein normales Buch und verfolgt nicht wie dieses einen roten Pfaden. Wenngleich die Rezepte im ersten Teil des Buches in logische Einheiten (Kapitel) unterteilt und in einer bewusst gewählten Reihenfolge abgedruckt werden, ist jedes doch in sich abgeschlossen und autark zu sehen. Der zweite Teil behandelt umfangreichere Rezepte, genannt Lösungen. Auch diese sind in sich abgeschlossen, stehen aber im Anwendungskontext des Kapitels.

Ich empfehle, speziell für Sie und Ihren Einsatz wichtige Kapitel zu überfliegen und die besonders interessanten Rezepte zu lesen und unbedingt auch auszuprobieren. Es macht einerseits Spaß, die Lösungen live zu erleben, und ist andererseits sehr lehrreich.

Einen guten Überblick über die Rezepte bietet auch das Inhaltsverzeichnis. Dieses sollte Ihr erster Anlaufpunkt sein, wenn Sie das Buch als Nachschlagewerk benutzen, wofür es explizit konzipiert wurde. Auch der Index hilft Ihnen, die passende Lösung zu finden.

1.5 Die mitgelieferten Beispiele

Zunächst einmal vorab: Alle Beispiele dieses Buches sind in der Sprache C# gehalten. Alle? Nicht alle! Im Kapitel „Sprachelemente“ haben sich durchaus einige VB. NET-Beispiele eingeschlichen, die insbesondere die wenigen tatsächlichen Unterschiede zwischen den beiden Sprachen aufzeigen sollen.

Warum C#? Möchte ich alle VB. NET-Entwickler vor den Kopf stoßen? Sicherlich nicht, denn ich bin selbst einer und komme ursprünglich aus der Welt der Desktop-Entwicklung. Visual Basic ist mir seit Version 3.0 ins Blut übergegangen. Wenngleich ich immer ein absoluter Verfechter der Sprache war, muss auch ich die schlichtweg bessere Syntax von C# zugeben. Hier erfährt die Sprache einen sehr deutlichen Vorteil gegenüber VB. NET.

Im Ergebnis unterscheiden sich die beiden Sprachen jedoch nicht, und daher habe ich auch darauf verzichtet, die Beispiele parallel in C# und VB. NET anzubieten. Aufgrund der Vielzahl der Beispiele hätte dies das Erscheinen des Buches enorm verzögert. Sollten Sie lieber mit VB. NET arbeiten, können Sie die Lösungen mit einfachen Mitteln übersetzen, der oftmals wichtige Layout-Bereich der Seite ist hiervon ohnehin nicht betroffen.

Sollte der Wunsch aufkommen, die Beispiele im Zuge einer Neuauflage in beiden Sprachen anzubieten, schreiben Sie mir bitte. Der Verlag und ich werden die Anzahl der Rückmeldungen maßgeblich in unsere Entscheidung einfließen lassen.

Installation der Beispiele

Sie finden die Quellcodes aller Beispiele auf der beiliegenden Buch-CD-ROM im Unterverzeichnis `Beispiele`. Um die Lösungen direkt auszuprobieren, kopieren Sie bitte den gesamten Inhalt des Ordners in das folgende – neu zu erstellende – Verzeichnis:

```
c:\inetpub\wwwroot\aspnet\
```

Anschließend sollten Sie den Schreibschutz global zurücksetzen. Dieses Verzeichnis ist insbesondere für die Verwendung der Datenbanken wichtig, da diese über absolute Pfade angesprochen werden.

1.6 Website zum Buch

Aktualisierte Informationen zu diesem und anderen Werken aus meiner Feder finden Sie auf der begleitenden Website zum Buch. Sie haben die Möglichkeit, die Listings ohne die Buch-CD-ROM herunterzuladen. Auch wird ein Newsletter mit aktuellen Informationen rund um die Website und ASP.NET angeboten. Um sich als Käufer zu authentifizieren, werden Sie zur Eingabe eines Passworts aufgefordert.

www.asp-buch.de

Passwort: vusewara

Das Passwort wurde übrigens mit Hilfe eines Rezepts aus diesem Buch erstellt. Wie das geht und warum mnemonische Passwörter so gut zu merken sind, erfahren Sie im Kapitel „Sicherheit“.

1.7 Kontakt zum Autor

Wie immer an dieser Stelle möchte ich mein Interesse an Ihrer Meinung betonen. Bitte schreiben Sie mir, wann immer Sie etwas zu diesem Buch zu sagen haben, sei es positiv oder auch negativ. Ihre Kritik ermöglicht es mir, weitere Projekte noch mehr an die Anforderungen meiner Leser anzupassen. Eine Bitte habe ich jedoch: Wenn Sie Kritik üben, seien Sie bitte konstruktiv, und zeigen Sie mir, wie ich es besser machen kann. Beleidigungen sind zum Glück selten und erreichen ohnehin nicht ihr Ziel – sie landen direkt im Papierkorb. Bitte schreiben Sie an folgende Adresse:

feedback@asp-buch.de

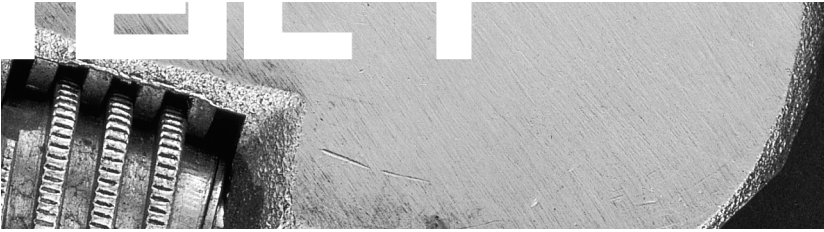
1.8 Die Zukunft dieses Buches ...

... liegt in Ihren Händen! Eine stark erweiterte Neuauflage ist bereits jetzt für das Jahr 2003 geplant. Wenn Sie Praxisfragen rund um ASP.NET haben, lassen Sie mich diese bitte wissen. Ich kann Ihnen nicht versprechen, jede individuell zu beantworten, werde deren Besprechung im Rahmen der Neuauflage jedoch in jedem Fall prüfen. Bitte richten Sie Ihre Ideen und Vorschläge an folgende Adresse:

suggestions@asp-buch.de

Und nun viel Spaß! :-)

Teil I – Rezepte



Der folgende erste und wichtigste Teil des Buches enthält Rezepte, Tipps und Tricks zum täglichen Umgang mit ASP.NET.

2

Basics



Wie kann ich ...

2 Basics

In diesem Kapitel finden Sie zahlreiche allgemeine Rezepte, Tipps und Tricks zum Umgang mit ASP.NET. Einige Ausnahmen kommen auch ganz ohne direkten Bezug zu ASP.NET aus, sehr nützlich sind die Rezepte dennoch. So erfahren Sie beispielsweise, wie Sie die Kommunikation zwischen Client und Server unter anderem zum Debugging belauschen können.

2.1 ... Debugging und Tracing nutzen?

Jeder macht mal Fehler, das gilt insbesondere auch für Entwickler bei der Arbeit. Anders als vorherige Versionen von ASP unterstützt Sie das neue ASP.NET außerordentlich gut bei der Fehlerfindung im Entwicklungsprozess. Es werden hierzu zwei Mechanismen angeboten.

Debugging hilft Ihnen im Falle eines Fehlers schnell, dessen Ursache zu finden und zu eliminieren. Hierzu wird statt der auszugebenden Seite ein Fehlerprotokoll angezeigt. Dabei wird zwischen Compiler-Fehlern während der ersten Übersetzung der Seite und Laufzeitfehlern unterschieden. Die Abbildung zeigt eine Division durch null, die bereits durch den Compiler erkannt wurde.

Das Debugging vom Laufzeitfehlern ist standardmäßig deaktiviert und muss folgerichtig explizit aktiviert werden. Dies geschieht über das Debug-Attribut der @Page-Direktive:

```
<% @Page Language="C#" Debug="true" %>
```

Alternativ können Sie das Debugging auch für die gesamte Web-Applikation aktivieren. Hierzu müssen Sie einen Eintrag in der Konfigurationsdatei `web.config` vornehmen.

Listing 2.1 web.config

```
<configuration>
  <system.web>
    <compilation debug="true"/>
  </system.web>
</configuration>
```

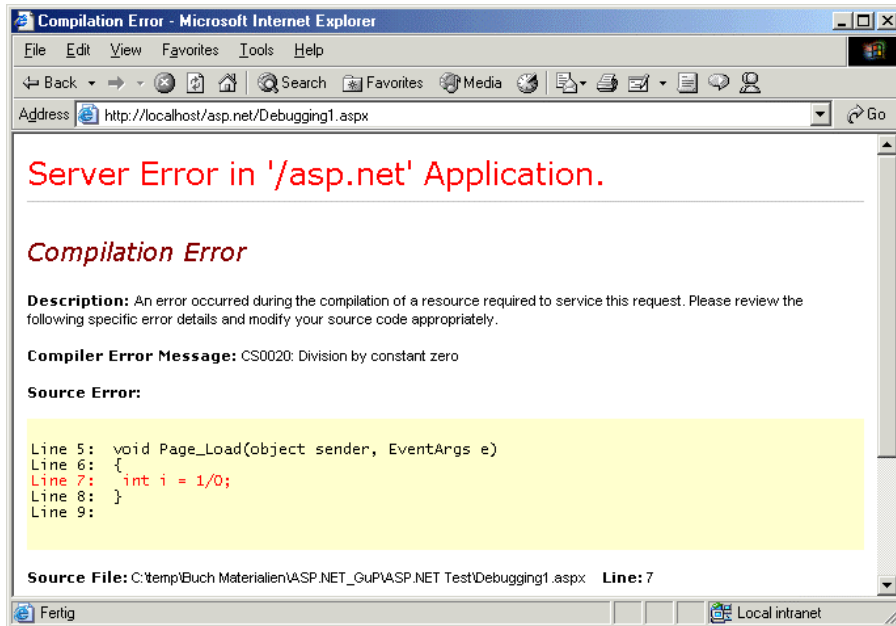


Abbildung 2.1 Der Compiler hat die ungültige Division erkannt.

Die zweite Möglichkeit, Fehlern auf die Spur zu kommen, ist das Tracing. Anders als das Debugging werden hier bei jedem Aufruf einer Seite zusätzliche Ausgaben angehängt. Hier sind beispielsweise die enthaltenen Cookies und die Server-Variablen enthalten. Auch lassen sich möglicherweise verwendete Server Controls sowie deren Speicherbedarf erkennen.

Auch das Tracing muss sinnigerweise explizit eingeschaltet werden. Es stehen zwei analoge Möglichkeiten zur Verfügung, die in der Regel zusätzlich und in Verbindung mit dem Debugging genutzt werden. Für die aktuelle Seite kann das Trace-Attribut der @Page-Direktive genutzt werden.

```
<% @Page Language="C#" Debug="true" Trace="true" %>
```

Für die gesamte Web-Applikation gilt auch hier die Verwendung der Konfigurationsdatei `web.config`:

Listing 2.2 `web.config`

```
<configuration>
<system.web>
  <trace enabled="true" pageOutput="true"/>
</system.web>
</configuration>
```

The screenshot shows a Microsoft Internet Explorer window with the address `http://localhost/asp.net/Tracing1.aspx`. The page displays two sections: **Request Details** and **Trace Information**.

Request Details:

Session Id:	v1z3ri45frbakcqqabew2h3w	Request Type:	GET
Time of Request:	29.04.2002 14:57:30	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

Trace Information:

Category	Message	From First(s)	From Last(s)
aspx.page	Begin Init		
aspx.page	End Init	0,000087	0,000087
aspx.page	Begin PreRender	0,000820	0,000732
aspx.page	End PreRender	0,000922	0,000103
aspx.page	Begin SaveViewState	0,010283	0,009360
aspx.page	End SaveViewState	0,010388	0,000106
aspx.page	Begin Render	0,010440	0,000052
aspx.page	End Render	0,195515	0,185074

Abbildung 2.2 Beim Tracing werden diese und andere nützliche Daten ausgegeben.



Um einen reibungslosen und schnellen Betrieb Ihrer Web-Applikation zu gewährleisten, sollten Sie Debugging und Tracing *unbedingt* ausschalten, bevor Sie Ihre Seite zum Produktiveinsatz veröffentlichen. Bleibt das Debugging hingegen eingeschaltet, so sind erhebliche Geschwindigkeitsverluste zu erwarten.

2.2 ... Debugging und Tracing mit Visual Studio .NET nutzen?

Die Visual Studio .NET-Entwicklungsumgebung gibt Ihnen weitere Freiheiten und Möglichkeiten, eine Web-Applikation zu debuggen. Neben einer netten Aufbereitung von Compiler-Fehlern in der Aufgabenliste ist insbesondere der Sprung in den Quellcode zu nennen. Sobald ein Laufzeitfehler auftritt, wird die Ausführung angehalten und die entsprechende Stelle im Code markiert.

Um das Debugging einzuschalten, wählen Sie die standardmäßig vorhandene Solution-Konfiguration „Debug“. Sollte dies nicht ausreichen, müssen Sie nachträglich das serverseitige ASP.NET-Debugging in den Projekteigenschaften Ihrer Web-Applikation aktivieren.

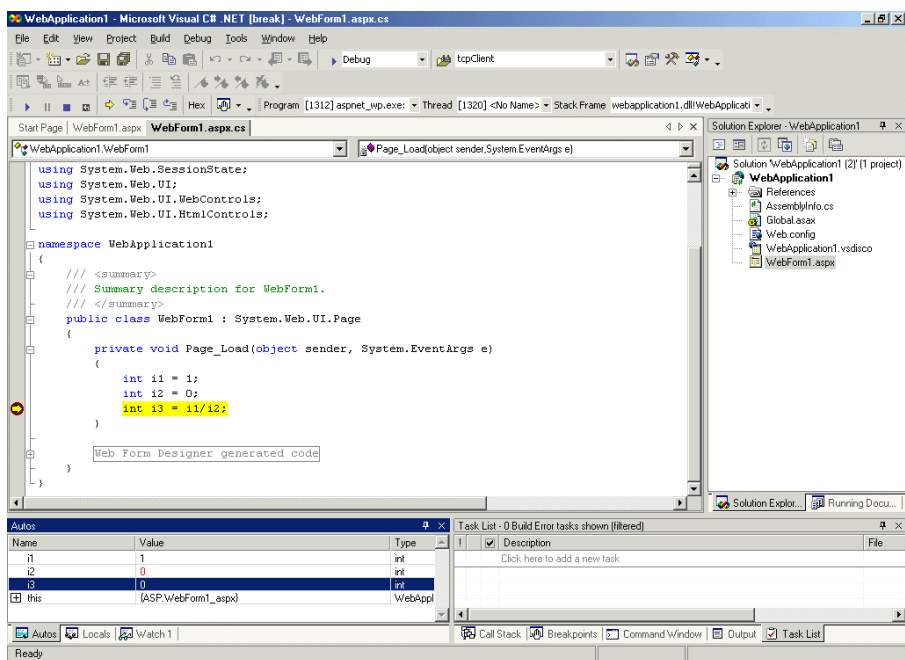


Abbildung 2.3 Die Ausführung der Web-Applikation wurde unterbrochen.

Eine hilfreiche Möglichkeit bieten Breakpoints. An beliebiger Stelle des Quellcodes können Sie einen Breakpoint setzen, indem Sie in der entsprechenden Zeile mit der linken Maustaste auf den grauen Bereich direkt links neben dem Quellcode-Eingabefenster klicken. Es erscheint ein roter Punkt, ein Breakpoint. Alternativ setzen Sie den Cursor in die entsprechende Zeile und betätigen die Taste F9. Sobald die Web-Applikation eine so markierte Position erreicht, wird die Ausführung unterbrochen, und Sie können sich einen Überblick über den aktuellen Programmsta-

tus verschaffen. Hierzu steht Ihnen insbesondere das mit „Autos“ umschriebene Variablenfenster zur Verfügung, das den Inhalt aller im Kontext stehenden Variablen anzeigt. Je nach Datentyp können Sie diesen Inhalt sogar verändern, indem Sie doppelt auf den Listeneintrag klicken und einen neuen Wert eingeben.

Über das Kontextmenü können Sie sich die Eigenschaften eines angelegten Breakpoints anzeigen lassen. Hier haben Sie mehrere Möglichkeiten, die Unterbrechung konditionell zu beschränken. So kann ein Breakpoint beispielsweise nur ein- oder normal ausgelöst oder vom Inhalt einer Variablen abhängig gemacht werden.

2.3 ... die Kommunikation zwischen Client und Server belauschen?

Das Protokoll HTTP basiert auf TCP/IP und besteht aus einfachen, standardisierten Textanweisungen. Dennoch ist das Protokoll, das jeder Abfrage einer Internetseite zugrunde liegt, für viele Web-Entwickler immer noch eine Blackbox. Was passiert wirklich zwischen Client und Server? Ein kleines Tool hilft, die Blackbox in eine Whitebox zu verwandeln und ist überdies für jeden Web-Entwickler eine gute Unterstützung beim Debuggen.

Das Programm proxyTrace wird als Proxy auf dem Client eingerichtet. Die Kommunikation zwischen dem Client und dem Browser wird hier durchgeschleust und protokolliert. Das System funktioniert daher sowohl mit dem Entwicklungsserver als auch mit beliebigen anderen Websites.

Nach dem Start des Programms werden Sie aufgefordert, eine Port-Nummer anzugeben. Der Standardwert 8080 ist in den meisten Fällen zu verwenden und sollte daher nicht geändert werden.

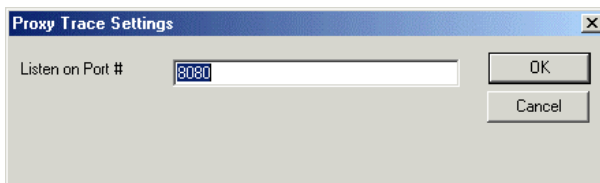


Abbildung 2.4 Die einfache Konfiguration des Programms

Nach dem Programm muss noch der Browser konfiguriert werden. Dieser muss alle Anfragen über den Proxy durchführen. Hierzu wird dieser in den LAN-Einstellungen des Internet Explorers hinterlegt. Die Abbildung zeigt die notwendige Konfiguration. Besonders wichtig ist das *Deaktivieren* der CheckBox „Proxy-Server für lokale Adressen umgehen“ am unteren Rand des Dialogs.

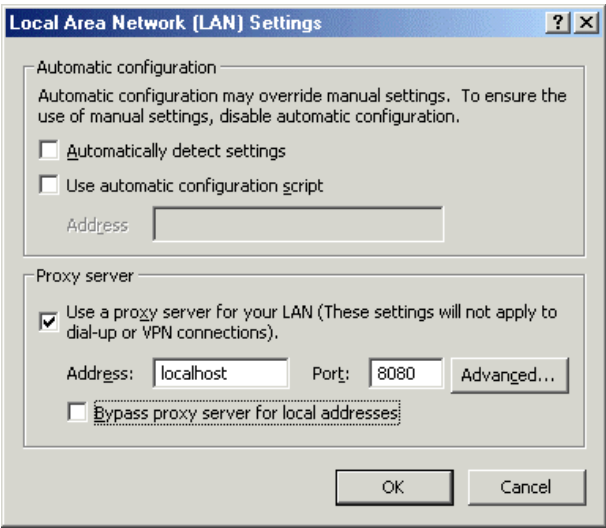


Abbildung 2.5 Der Internet Explorer muss den Proxy verwenden.

Sind Programm und Internet Explorer korrekt konfiguriert, können Sie wie gewohnt auf Ihrem Entwicklungsserver „surfen“. Alle Anfragen werden jedoch protokolliert und im Programmfenster angezeigt. Dieses ist dreigeteilt. Im linken Bereich sind alle Client-Anfragen mit einigen Zusatzinformationen aufgelistet.

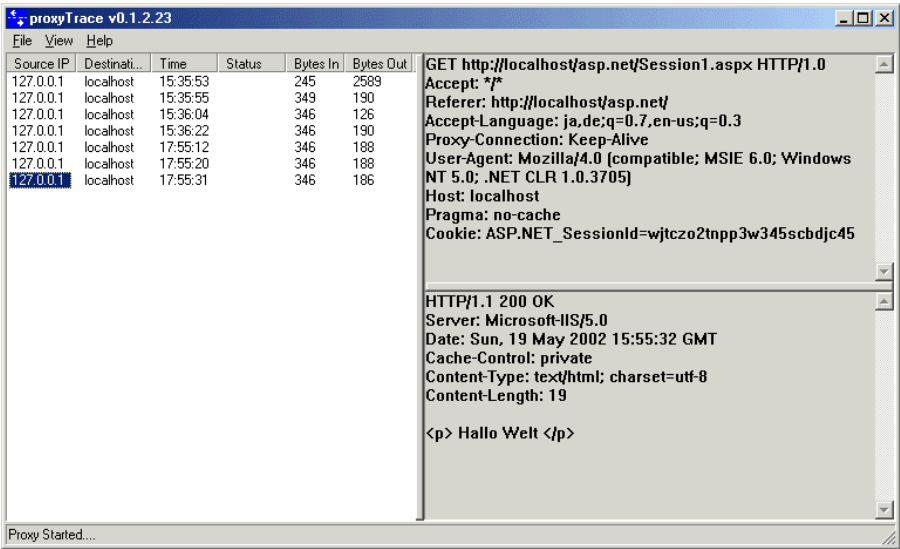


Abbildung 2.6 Mit proxyTrace wird das Protokoll HTTP zur Whitebox.

Der rechte Bereich ist aufgeteilt in die Client-Anfrage oben und die Antwort des Servers unten. Es handelt sich dabei jeweils um die reinen per HTTP übertragenen Informationen. Deutlich erkennbar sind die Kopfzeileneinträge und davon getrennt die eigentlichen Daten. Bei der Antwort des Servers handelt es sich um den darzustellenden HTML-Stream.

Die Abbildung zeigt eine einfache Client-Anfrage. Sie können hier deutlich sehen, welche Daten an den Server übertragen werden. Insbesondere interessant ist der ASP.NET-Session-Cookie.

Das Programm proxyTrace ist kostenlos als Freeware erhältlich. Es wird von den beiden Autoren über die folgende Website zum Download angeboten:

<http://www.pocketsoap.com/tcptrace/pt.asp>

2.4 ... alle vorhandenen Kopfzeilen und Server-Variablen ausgeben?

Gerade beim Debuggen von Seiten und Web-Applikationen ist es oftmals hilfreich, den Inhalt der vom Client übergebenen Kopfzeilendaten und die vom Server zur Verfügung gestellten Server-Variablen zu kennen. Im Trace-Modus werden zumindest Letztere bei jeder Anfrage im Browser mit ausgegeben.

Mit Hilfe des nachfolgenden Listings können Sie die Daten bei Bedarf jederzeit anfordern. Die beiden Collections `Request.Headers` und `Request.ServerVariables` werden als Datenquelle für zwei `DataList`-Controls verwendet.

Listing 2.3 ServerVariables1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    headers.DataSource = Request.Headers;
    servervars.DataSource = Request.ServerVariables;
    DataBind();
}

</script>

<h2>Kopfzeilen</h2>

<ASP:DataList id="headers" runat="server">

    <ItemTemplate>
```

```
<b><# Container.DataItem %>:</b>
<# Request.Headers[Container.DataItem.ToString()] %>
</ItemTemplate>

</ASP:DataList>

<h2>Server-Variablen</h2>

<ASP:DataList id="servervars" runat="server">

  <ItemTemplate>
    <b><# Container.DataItem %>:</b>
    <# Request.ServerVariables [Container.DataItem.ToString()] %>
  </ItemTemplate>

</ASP:DataList>
```

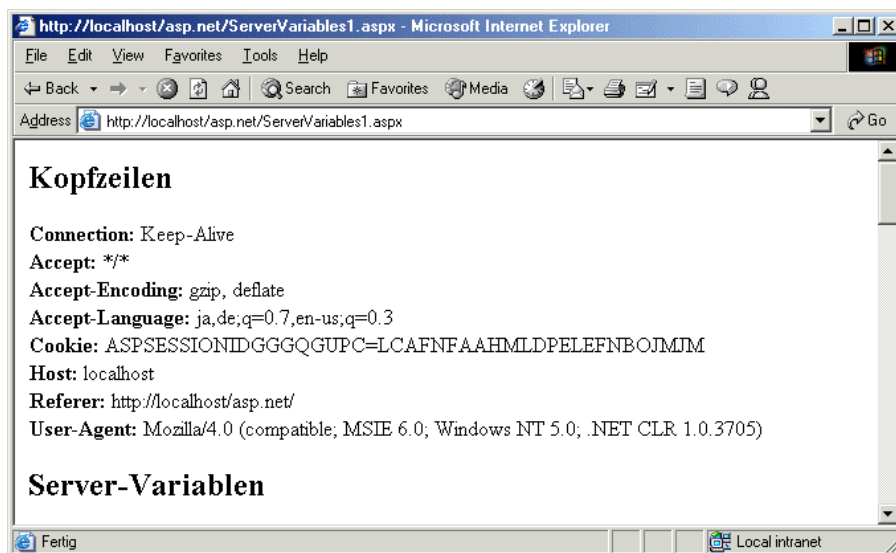


Abbildung 2.7 Alle Infos parat: Kopfzeileneinträge und Server-Variablen

2.5 ... eine umfangreiche Website strukturieren?

„Ordnung muss sein“, heißt es so schön. Eine kleinere Website kann durchaus einmal ein wenig chaotisch sein, doch bei größeren Projekten sollte eine gewisse Planung im Voraus nicht vernachlässigt werden. Erfahrungsgemäß hat sich eine Reihe von Regeln als äußerst praktisch und hilfreich erwiesen:

1. Ausgehend vom Hauptverzeichnis mit der Startseite sollte jeder logische Bereich in einem eigenen, physikalischen Verzeichnis abgelegt werden. So lassen sich die unterschiedlichen Funktionen auch optisch gliedern und für Außenstehende nachvollziehbar gestalten. Jedes Verzeichnis sollte mit einem passenden, nach Möglichkeit englischen, Namen versehen werden.
2. Wann immer sinnvoll, sollten hierarchische Strukturen nachgebildet und verschachtelte Bereiche erstellt werden. So könnte ein allgemeiner Produktbereich wiederum Bereiche mit Detailinformationen zu den Produkten enthalten.
3. Jeder logische Bereich, also jedes Verzeichnis, sollte eine Einstiegsseite enthalten, die eine Übersicht über den gewählten Bereich anbietet. Diese Seite sollte angezeigt werden, wenn das Verzeichnis direkt im Browser angefordert wird. Hierzu ist es erforderlich, dass die Seite den Namen `default.aspx` trägt. Der oberste Bereich ist für die Startseite reserviert.
4. Die einzelnen Seiten sollten sich von einer Code Behind-Datei im aktuellen Verzeichnis ableiten. Diese sollte mit einem standardisierten Namen versehen werden, beispielsweise `_page.cs`. Die Seite nimmt bereichsspezifische Einstellungen vor und leitet sich ihrerseits von einer zentralen Code Behind-Klasse ab, die als DLL im bin-Verzeichnis existiert.
5. Jeder Bereich erhält eine thematische Navigation. Die notwendigen Informationen werden entweder direkt in der lokalen Code Behind-Datei oder aber in einer zusätzlichen Navigationsdatei untergebracht. In diesem Fall sollte ebenfalls ein standardisierter Name gewählt werden, beispielsweise `_nav.cs` beziehungsweise `_nav.xml`, wenn die Daten im XML-Format vorliegen. Ist dies im Kontext nicht sinnvoll, erbt der Bereich die Navigation des darüber liegenden.
6. In jedem Verzeichnis sollten ausschließlich die benutzten ASP.NET-Dateien abgelegt werden. Andere Formate sollten in separaten Unterverzeichnissen mit standardisierten Namen abgelegt werden. So können Grafiken beispielsweise in einem Ordner `grafix` und herunterzuladende Dateien in `files` abgelegt werden. Die Verzeichnisse sollten autark strukturiert sein, so dass die Dateien auch wirklich nur von diesem Bereich benötigt werden.
7. Globale Dateien und Grafiken sollten unterhalb eines entsprechenden Ordners abgelegt werden. Hierzu zählt zum Beispiel das Logo der Website beziehungsweise des Unternehmens. Derartige Grafiken könnten im Ordner `/global/grafix/` untergebracht werden.

Das Umsetzen dieser Regeln ist zunächst mit ein klein wenig Arbeit verbunden. Bereits nach kurzer Zeit macht sich diese Investition jedoch in Form von verbesserter Übersicht und Struktur bemerkbar. Ich habe dieses System bei unterschiedlichen Seiten mit 50, aber auch mit einer Seitenanzahl im vierstelligen Bereich erfolgreich eingesetzt. Die nachfolgenden Rezepte liefern Ihnen hierzu weitere Ideen.

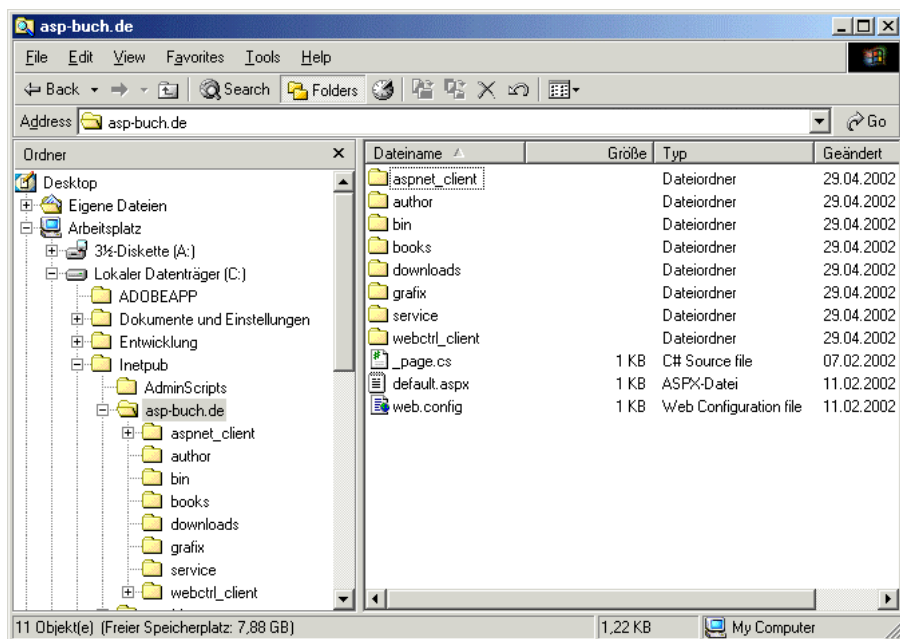


Abbildung 2.8 Je strukturierter, desto übersichtlicher bleibt eine Website.

2.6 ... eine globale Seitenvorlage erstellen?

Das Layout einer Website in jeder einzelnen Unterseite neu abzulegen ist wirklich nicht mehr standesgemäß. Mit älteren ASP-Versionen hatten Sie die Möglichkeit, globale Layoutdateien serverseitig einzubinden. Die Include-Direktiven werden zwar weiterhin unterstützt, sind bei ASP.NET jedoch nur ein Überbleibsel aus Kompatibilitätsgründen. Die neue Version bietet bessere Möglichkeiten, die in Verbindung mit Code Behind genutzt werden können. Im Folgenden stelle ich Ihnen eines der denkbaren Systeme vor, ein – wie ich denke – gleichermaßen sinnvoller wie einfacher und effektiver Ansatz, das Layout einer Seite zentral vorzuhalten.

Die zentrale Seitenvorlage

Die Vorlage für alle Unterseiten wird in Form eines User Controls abgelegt. Dies bietet gegenüber einer Code Behind-Datei den Vorteil, dass Sie die benötigten HTML-Anweisungen in gewohnter Form ganz einfach niederschreiben können. Die Stelle, an der später der eigentliche Inhalt der Seite abgelegt werden soll, wird mit einem Placeholder-Control versehen.

Listing 2.4 Page.ascx

```
<% @Control Language="C#"
    Inherits="PAL.Projects.AspNetKochbuch.PageTemplate" %>
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<html>
<head>
    <title>
        <%# (Title != null) ? Title+" - " : "" %>ASP.NET Kochbuch
    </title>
</head>

<body leftmargin="10" topmargin="10">

<form runat="server">

<p><i>Hier könnte eine globale Navigation eingefügt werden.</i></p>

<table width="100%" border="1" cellpadding="5" cellspacing="0">
    <tr>
        <td width="25%" valign="top">
            <p><i>Hier könnte eine Bereichsnavigation eingefügt werden.</i></p>
        </td>
        <td width="75%" valign="top">
            <p><i>Hier kommt der eigentliche Inhalt der Seite hin:</i></p>
            <ASP:Placeholder id="Main" runat="server"/>
        </td>
    </tr>
</table>

<!-- // Footer // -->
<p class="small" align="center">
    Copyright © ...<br>
    Alle Rechte vorbehalten.
</p>

</form>

</body>
</html>
```

Im Listing können Sie die spätere Struktur der Seiten bereits recht gut nachvollziehen. Es existiert ein Bereich für eine globale Navigation und mit Hilfe einer Tabelle abgetrennt eine Bereichsnavigation und der Inhalt der Seite. Am Ende ist Platz für einen Fußtext, beispielsweise für den Copyright-Hinweis oder Disclaimer.

Die zentrale Code Behind-Datei

Das User Control leitet sich von einer Code Behind-Klasse ab. Diese befindet sich in einer zentralen Quellcodedatei, die als DLL kompiliert im bin-Verzeichnis der Web-Applikation abgelegt wird.

Die Basisklasse des User Controls hört auf den Namen `PageTemplate`. Die Klassendefinition enthält lediglich zwei öffentliche Felder. Es handelt sich um eine Zeichenkette mit einem optionalen Seitentitel und ein `Placeholder-Control`. Dieses wird von dem abgeleiteten User Control überschrieben. Sinn dieser Definition ist der Zugriff auf die beiden Objekte innerhalb der globalen Seitenvorlage.

```
public class PageTemplate : UserControl
{
    public Placeholder Main;
    public string Title;
}
```

Diese Vorlage ist in derselben Code Behind-Datei definiert und nennt sich `GlobalPageTemplate`. Ein globales als `protected` markiertes Feld nimmt eine Instanz der eben gezeigten Klasse auf. Die Zuweisung erfolgt innerhalb der überschriebenen Methode `Construct`. Diese wird für derartige Funktionalitäten durch die Basisklasse `Page` angeboten. Hier wird mittels `LoadControl` das User Control eingeladen und nach einer Typenkonvertierung in dem globalen Feld abgelegt. Außerdem wird das Control der `Controls-Collection` der Seite angefügt.

```
public class GlobalPageTemplate : Page
{
    protected PageTemplate Template;

    protected override void Construct()
    {
        this.Template = (PageTemplate) this.LoadControl("page.ascx");
        this.Controls.Add(this.Template);
    }
    protected override void AddParsedSubObject(object obj)
    {
        this.Template.Main.Controls.Add((Control) obj);
    }
}
```


Richtig spannend wird's im zweiten Schritt. Hier wird die Methode `AddParsedSubObject` überschrieben. Diese wird von der `Page`-Klasse verwendet, um die einzelnen Objekte der Seite der `Controls-Collection` hinzuzufügen. Der Trick besteht nun darin, statt der lokalen `Collection` die des `Placeholder-Controls` im `User Control` zu verwenden. Auf diese Weise werden alle Elemente der Seite an der markierten Stelle im `User Control` platziert.

Nun muss die Quellcodedatei kompiliert werden. Die so erzeugte `DLL` wird im `bin`-Verzeichnis der `Web-Applikation` abgelegt. Der Aufruf des `Kommandozeilenprogramms` sieht in etwa wie folgt aus:

```
csc /t:library /r:System.dll,System.Web.dll global.cs
```

Die lokale Code Behind-Datei

Um eine individuelle Anpassung und Ergänzung der Seiten im lokalen Bereich beziehungsweise Verzeichnis vornehmen zu können, leiten sich die einzelnen Seiten nicht direkt von der eben gezeigten `Code Behind-Klasse` ab. Vielmehr ist pro Verzeichnis eine zusätzliche Datei zwischengeschaltet. Im Rohzustand sieht diese wie folgt aus:

Listing 2.5 _page.cs

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace PAL.Projects.AspNetKochbuch
{
    public class LocalPageTemplate : GlobalPageTemplate
    {
        protected override void Construct()
        {
            {
                base.Construct();
                // Einstellungen für den aktuellen Bereich
            }
        }
    }
}
```

Die einzelnen Seiten

Jede einzelne Seite muss nun lediglich von der gezeigten Code Behind-Datei abgeleitet und mit dem gewünschten Inhalt versehen werden – nichts einfacher als das.

Listing 2.6 default.aspx

```
<% @Page Language="C#" Src="_page.cs"
    Inherits="PAL.Projects.AspNetKochbuch.LocalPageTemplate"
    Debug="true" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Template.Title = "Startseite";
    Template.DataBind();
}

</script>

<h1> Willkommen auf meiner Website </h1>

<p> Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blind-
text Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext </p>
```

Das Ergebnis

Die Abbildung zeigt das Ergebnis des Systems. Die @Page-Direktive in der Datei default.aspx sorgt dafür, dass das hinterlegte zentrale Layout auf den Inhalt angewandt wird. Das System steht, Erweiterungen sind willkommen und einfach zu implementieren.

Eine weitaus schönere Möglichkeit, Ausnahmen abzufangen, bietet das Ereignis `Error` der Klasse `Page`. Sie können dieses nicht nur in der jeweiligen Seite, sondern selbstverständlich auch in einer zentralen Code Behind-Klasse behandeln. Die folgende Klasse zeigt genau das. Sobald ein Fehler auftritt, wird in diesem Beispiel eine serverseitige (!) Umleitung durchgeführt.

Listing 2.8 `Page_Error.cs`

```
using System;
using System.Web;
using System.Web.UI;

public class MyPage : Page
{
    protected override void OnError(EventArgs e)
    {
        Server.Transfer("errorpage.aspx");
    }
}
```

Durch die serverseitige Umleitung kann die Zielseite auf die ursprünglich aufgetretene Ausnahme zugreifen. Diese steht über die Methode `Server.GetLastError` zur Verfügung. Folgerichtig ist es nun auch möglich, eine genaue Fehlerbeschreibung auszugeben und diese möglicherweise per Email direkt an den zuständigen Administrator zu senden. Die folgende Seite zeigt ein Beispiel für eine derartige Fehlermeldung.

Listing 2.9 `ErrorPage.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DataBind();
}

</script>

<h1>Ups...</h1>

<p> Bei der Bearbeitung der Anfrage ist leider folgender Fehler aufgetreten:</p>

<p> <i> %# Server.GetLastError().Message %</i> </p>

<p> Bitte wenden Sie sich an den Administrator! </p>
```

Um die Ausgabe der Fehlerseite zu provozieren, benötigen Sie jetzt nur noch eine Seite, die eine unbehandelte Ausnahme zur Folge hat. Im folgenden Listing wird eine Division durch 0 durchgeführt, was bekanntermaßen nicht möglich ist. Damit das Beispiel funktioniert, muss die Seite selbstverständlich von der erstellten Code Behind-Klasse abgeleitet werden.

Listing 2.10 Page1.aspx

```
<% @Page Language="C#" Debug="true"
    Src="Page_Error.cs" Inherits="MyPage" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    int x = 0;
    int y = 1/x;
}

</script>
```

Die Abbildung zeigt das Ergebnis im Browserfenster. Eine übersichtliche Fehlermeldung wird angezeigt. Selbstverständlich lassen sich auch die übrigen Daten der Exception-Klasse wie Quelle, mögliche innere Ausnahme und so weiter auswerten.



Abbildung 2.10 Die globale Fehlerbehandlung funktioniert.

2.8 ... überprüfen, ob Cookies akzeptiert werden?

Sofern Sie in Ihrer Web-Applikation beispielsweise für den Session-State Cookies verwenden, sollten Sie überprüfen, ob der Client diese auch akzeptiert. Dank Filterprogrammen wie WebWasher und restriktiver Sicherheitspolitik erlauben viele Browser die Annahme von Cookies *nicht*, so dass die darauf basierenden Teile Web-Applikation nicht funktionieren würden.

Das Listing zeigt die Implementierung eines einfachen Cookie-Checks in Form einer booleschen Methode. Beim ersten Aufruf der Seite wird diese Methode aufgerufen. Es wird ein temporärer Cookie angelegt und ein Redirect auf die gleiche Seite ausgeführt. Existiert der Cookie beim zweiten Aufruf der Seite noch, so akzeptiert der Browser die Anlage, ansonsten leider nicht. In diesem Fall wird ein erneuter Redirect durchgeführt, diesmal auf eine Hilfe-Seite mit entsprechenden Erklärungen.

Listing 2.11 CheckCookie1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        if(!CookiesEnabled())
            Response.Redirect("nocookies.aspx", true);
    }
}

bool CookiesEnabled()
{
    if(Request.QueryString["check"] == null)
    {
        Response.Cookies.Add(new HttpCookie("check", "1"));
        string redir = string.Format("{0}?check=1", Request.Path);
        Response.Redirect(redir, true);
        return(false);
    }
    else
    {
        return(Request.Cookies["check"] != null);
    }
}

</script>

<p> Cookies sind aktiviert...! :-)</p>
```

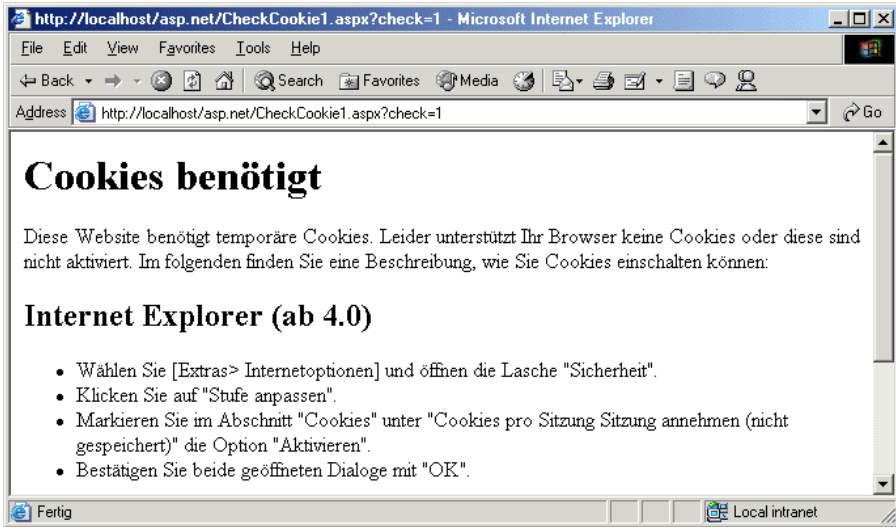


Abbildung 2.11 Cookies sind auf diesem Rechner leider deaktiviert.

2.9 ... überprüfen, ob JavaScript aktiviert ist?

Eine sichere Überprüfung auf JavaScript können Sie nur auf eine Weise durchführen: indem Sie es benutzen. Beim ersten Aufruf einer Seite führen Sie einen Redirect auf eine zweite durch. Diese enthält den JavaScript-Quellcode, der zum Aufruf der ersten notwendig ist. Wird diese clientseitige Umleitung nicht durchgeführt, wird die Seite mitsamt einer Fehlerbeschreibung im Browser angezeigt. Ansonsten wird erneut die erste Seite aufgerufen, und es steht fest, dass JavaScript aktiviert ist.

Listing 2.12 test.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        CheckJSEnabled();
}

void CheckJSEnabled()
{
    if (Request.QueryString["jscheck"] == null)
    {
        string redir = string.Format("jscheck.aspx?returnurl={0}",
            Request.Path);
    }
}
```

```
    Response.Redirect(redir, true);
}
}

</script>

<p> JavaScript ist aktiviert ...! :-) </p>
```

Listing 2.13 jscheck.aspx

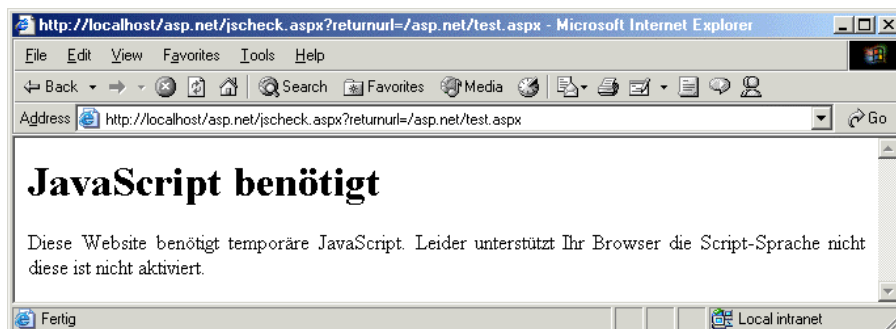
```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string returnUrl = Request.QueryString["returnurl"];
    string script = "<script language=\"JavaScript\">";
    script += "window.location.href=\"";
    script += returnUrl + "?jscheck=1\";"/>";
    script += "<script>";

    if(!IsStartupScriptRegistered("js"))
        this.RegisterStartupScript("js", script);
}

</script>

<form runat="server">
<h1> JavaScript benötigt </h1>
<p align="justify">Diese Website benötigt JavaScript. Leider unter-
stützt Ihr Browser die Script-Sprache nicht oder diese ist nicht
aktiviert. </p>
</form>
```

**Abbildung 2.12** Ohne JavaScript geht's nicht weiter.

2.10 ... die Möglichkeiten des Client-Browsers ermitteln?

Auch wenn die Features der gängigen Browser in den vergangenen Jahren immer weiter angeglichen wurden, so gibt es doch weiterhin kleine, aber feine Unterschiede. Wie die Server-Statistiken frequentierter Seiten zudem beweisen, ist immer noch ein gewisser Prozentsatz ziemlich alter Browser-Versionen im Einsatz der Surfer. Gerade größere Firmen, Institutionen und Behörden trennen sich nur sehr schweren Herzens von einmal eingeführter Soft- und Hardware.

Über die Kopfzeile `UserAgent` des Protokolls HTTP liefert der Client-Browser Informationen über sich. In der standardisierten Zeichenkette ist beispielsweise der Name und die Version des Browsers sowie des Betriebssystems enthalten. Sie können die Kopfzeile über die gleichnamige Eigenschaft der Klasse `HttpRequest` abfragen und ausgeben.

Listing 2.14 UserAgent1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Response.Write(Request.UserAgent);
}

</script>
```

Im Falle meiner Windows 2000-Installation mit Internet Explorer Version 6.0 und dem .NET Framework Release 1 (Version 1.0.3705) sieht die Ausgabe so aus:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)
```

Sie können die gelieferten Informationen benutzen, um individuell auf die Eigenheiten eines bestimmten Browsertyps oder einer -version zu reagieren.

Eine Alternative hierzu bietet die Eigenschaft `Browser` der Klasse `HttpRequest`. Sie erhalten hier Zugriff auf eine Instanz der Klasse `HttpBrowserCapabilities`. Diese fragt eine Datenbank nach den übertragenen Informationen ab und ordnet diese so einer hinterlegten Browserkonfiguration zu. Diese können Sie über die Eigenschaften der Klasse abfragen. Beispielsweise können Sie so ermitteln, ob der Browser Frames unterstützt.

Listing 2.15 BrowserCapabilities1.aspx

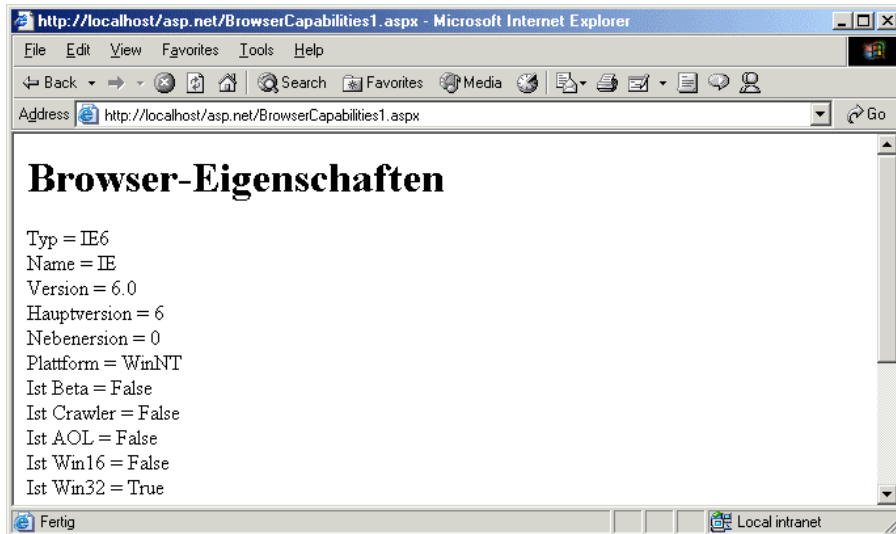
```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<h1>Browser-Eigenschaften</h1>");

    HttpBrowserCapabilities bc = Request.Browser;

    Response.Write("Typ = " + bc.Type + "<br>");
    Response.Write("Name = " + bc.Browser + "<br>");
    Response.Write("Version = " + bc.Version + "<br>");
    Response.Write("Hauptversion = " + bc.MajorVersion + "<br>");
    Response.Write("Nebenversion = " + bc.MinorVersion + "<br>");
    Response.Write("Plattform = " + bc.Platform + "<br>");
    Response.Write("Ist Beta = " + bc.Beta + "<br>");
    Response.Write("Ist Crawler = " + bc.Crawler + "<br>");
    Response.Write("Ist AOL = " + bc.AOL + "<br>");
    Response.Write("Ist Win16 = " + bc.Win16 + "<br>");
    Response.Write("Ist Win32 = " + bc.Win32 + "<br>");
    ...
}

</script>
```

**Abbildung 2.13** Was kann der Browser theoretisch?

Das Listing zeigt die Abfrage einiger Informationen; die Abbildung zeigt das Ergebnis beim Aufruf der Seite. Da in der beschriebenen Kopfzeile theoretisch beliebige Daten übertragen werden können, können Sie sich nicht zu 100% auf die Angaben verlassen. Zudem kann es sein, dass der Browser ein Feature wie VBScript zwar prinzipiell unterstützt, dies aber für die aktuelle Sicherheitszone deaktiviert wurde oder gar von einer Firewall unterdrückt wird.

Die Datenquelle für die Zuordnung der UserAgent-Kopfzeile und der Features der Browser befindet sich auf dem lokalen Web-Server. Um immer auf dem aktuellen Stand zu sein und auch aktuelle Browser zu unterstützen, sollten Sie die Daten in regelmäßigen Abständen aktualisieren. Informationen hierzu finden Sie auf der folgenden Website:

<http://www.cyscape.com/browsercaps>

2.11 ... Seiten für unterschiedliche Browser optimieren?

Über die Browser-Capabilities haben Sie die Möglichkeit, Ihren Quellcode an die jeweiligen unterstützten Features des vom Benutzer verwendeten Browsers anzupassen. Es gibt dazu verschiedene Ansätze. Die sicherlich einfachste Variante ist die Verwendung mehrerer Placeholder-Controls. Diese werden in Abhängigkeit vom Browser ein- beziehungsweise ausgeschaltet und enthalten den jeweils spezifischen Quellcode.

Das Listing zeigt den Ansatz. Bei Verwendung einer Version des Internet Explorers wird der eine Text ausgegeben, ansonsten der andere.

Listing 2.16 BrowserOptim1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DataBind();
}

</script>

<ASP:Placeholder runat="server"
    visible=<%= Request.Browser.Browser=="IE" %>>
<p> Sie verwenden einen Internet Explorer </p>
</ASP:Placeholder>
```

```
<ASP:Placeholder runat="server"
  visible=<## Request.Browser.Browser!="IE" %>>
<p> Sie verwenden einen anderen Browser als den Internet Explorer </p>
</ASP:Placeholder>
```

Für statische Seiten ist dieser Ansatz geeignet. Sind jedoch weitere, serverseitige Controls enthalten, die interaktiv angesprochen werden sollen, wird diese Implementierung unübersichtlich. Wann immer Sie auf eines der Controls zugreifen wollen, müssen Sie die gezeigte Abfrage erneut durchführen, um das richtige Objekt zu erreichen. Es empfiehlt sich daher, derartige Funktionalitäten beispielsweise in ein User Control zu kapseln.

Das folgende Listing zeigt die Implementierung eines User Controls, das in Abhängigkeit vom verwendeten Browser den Inhalt der Eigenschaft `Text` ausgibt. Die aufrufende Seite im zweiten Listing bekommt von dieser Unterscheidung nichts mit und greift in jedem Fall über dieselbe Eigenschaft zu.

Listing 2.17 BrowserOptim2.ascx

```
<script runat="server">

protected string m_Text;

public string Text
{
  get { return(m_Text); }
  set { m_Text = value; }
}

</script>

<ASP:Placeholder runat="server"
  visible=<## Request.Browser.Browser=="IE" %>>
  <p>
    Sie verwenden einen Internet Explorer. Text: <## this.Text %>
  </p>
</ASP:Placeholder>

<ASP:Placeholder runat="server"
  visible=<## Request.Browser.Browser!="IE" %>>
  <p>
    Sie verwenden einen anderen Browser als den Internet Explorer.
    Text: <## this.Text %>
  </p>
</ASP:Placeholder>
```

Listing 2.18 BrowserOptim2.aspx

```
<% @Register tagprefix="PAL" tagname="BrowserOptim"
    src="BrowserOptim2.ascx" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    bo.Text = "Hallo Welt";
    DataBind();
}

</script>

<PAL:BrowserOptim id="bo" runat="server"/>
```

Die in ASP.NET enthaltenen Server Controls passen sich automatisch dem verwendeten Browsertypus an. Hierbei wird zwischen den unterstützten HTML-Versionen unterschieden. Mögliche Werte sind dabei 4.0 und 3.2 beziehungsweise eine unbekannte Version.

Wenn Sie eigene Custom Controls entwickeln, nutzen Sie die Umschaltung automatisch, sofern Sie innerhalb der überschriebenen Methode `Render` beziehungsweise `RenderContents` die übergebene `HtmlTextWriter`-Instanz verwenden. Handelt es sich um einen Browser mit HTML Version 3.2 oder geringer, wird die abgeleitete Klasse `Html32TextWriter` übergeben. Diese sorgt dafür, dass bestimmte Tags und Attribute der Version 4.0 mittels der äquivalenten Tags der alten Version abgebildet werden.

2.12 ... Frames in ASP.NET verwenden?

Die Verwendung von Frames unterscheidet sich bei ASP.NET nicht von statischem HTML. Sie können eine `aspx`-Seite anlegen und hier die üblichen `frameset`-Tags angeben. Ob die Datei die Endung `aspx` oder `html` trägt, ist dabei relativ egal. Im folgenden Listing sehen Sie eine `Frameset`-Definition ohne `Script`-Anweisungen.

Listing 2.19 frameset1.aspx

```
<% @Page Language="C#" Debug="true" %>

<frameset cols="50%,50%">
    <frame src="left.aspx"/>
    <frame src="right.aspx"/>
</frameset>
```

Das Ergebnis der Seite sieht wie gewöhnlich aus. In diesem Fall handelt es sich um eine vertikal getrennte Seite mit zwei Frames.



Abbildung 2.14 Frames funktionieren bei ASP.NET mit normalen HTML-Anweisungen.

Theoretisch wäre es natürlich denkbar, ein Server-Control zu entwickeln, über das Frames programmatisch definiert werden können. Aber mal ehrlich, braucht man das? Sie meinen ja? Dann können Sie das `HtmlGenericControl` verwenden und die benötigten Angaben über die `Attributes-Collection` übergeben – kein Problem!

Listing 2.20 frameset2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    HtmlGenericControl frameset = new HtmlGenericControl("frameset");
    frameset.Attributes.Add("cols", "50%,50%");
    this.Controls.Add(frameset);

    HtmlGenericControl frameleft = new HtmlGenericControl("frame");
    frameleft.Attributes.Add("src", "left.aspx");
    frameset.Controls.Add(frameleft);

    HtmlGenericControl frameright = new HtmlGenericControl("frame");
    frameright.Attributes.Add("src", "right.aspx");
    frameset.Controls.Add(frameright);
}

</script>
```

Das HTML-Ergebnis des zweiten Listings sieht gegenüber der manuellen Variante minimal anders aus, zumal Sie ohne „Punkt“ und „Komma“ (gemeint ist ein Umbruch) daherkommt. Die Visualisierung ist jedoch vollkommen identisch.

```
<frameset cols="50%,50%"><frame src="left.aspx"></frame>
<frame src="right.aspx"></frame></frameset>
```

2.13 ... einen Redirect durchführen?

Ein Redirect ist eine Umleitung auf Ebene des Protokolls HTTP. Die aufgerufene Seite meldet in den zurückgesandten Kopfzeilen die Adresse einer neuen Seite. Der Client ruft daraufhin die neue Seite ab. Ein Redirect wird also clientseitig durch den Browser ausgeführt. Folgerichtig kann die Zielseite eine beliebige Adresse im Internet sein und muss *nicht* auf dem gleichen Server liegen. Eine typische Verwendung sieht wie folgt aus:

Listing 2.21 Redirect1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Response.Redirect("zweiteseite.aspx", true);
}

</script>
```

Im Browser wird nun statt der angeforderten Seite die neue Adresse `zweiteseite.aspx` angezeigt.

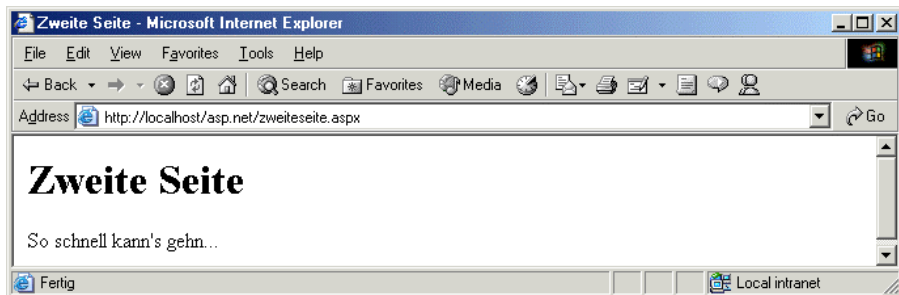


Abbildung 2.15 Die Seite wurde umgeleitet.

Viele ASP-Entwickler wissen nicht um die Hintergründe eines Redirects. Als Folge werden in den diversen Newsgroups und Mailing-Listen immer wieder die gleichen Fragen gestellt. Um ein Verständnis für die letztlich simple Technik zu schaffen, zeigt das nachfolgende Listing die manuelle Erstellung eines Redirects.

Listing 2.22 Redirect2.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Response.StatusCode = 302;
    Response.AppendHeader("Location", "zweiteseite.aspx");
    Response.End();
}

</script>
```

Beim Aufruf der Redirect-Methode werden intern zwei Änderungen an den Kopfzeilen vorgenommen. Zum einen wird der Statuscode auf 302 („Object moved“) gesetzt, und zum anderen wird über den Kopfzeileneintrag „Location“ die neue Seite angegeben.



Beachten Sie bei beiden gezeigten Ansätzen des Redirects bitte immer, dass es sich um eine Änderung der Kopfzeilendaten handelt. Folgerichtig dürfen die Kopfzeilen noch nicht an den Client übertragen worden sein. Ansonsten kann der Redirect nicht durchgeführt werden, und eine entsprechende Ausnahme wird geworfen.

2.14 ... einen Redirect für einen anderen Frame durchführen?

Wie im vorherigen Rezept beschrieben, handelt es sich bei einem Redirect um eine Protokollfunktionalität. Aus diesem Grund ist beispielsweise die Angabe eines Frames nicht möglich. Das Protokoll HTTP „weiß“ ja gar nicht, was da übertragen und im Browser dargestellt wird. Ein Redirect mit Frames ist also prinzipiell nicht möglich.

Einen Ansatz, das gewünschte Ergebnis dennoch zu erreichen, bietet JavaScript. Über das reguläre clientseitige Objektmodell kann der Inhalt eines anderen Frames neu geladen werden. Das nachfolgende Listing kommt innerhalb eines Framesets zum Einsatz. Über ein LinkButton-Control lässt sich ein Quasi-Redirect in einem anderen Frame durchführen. Die beiden Abbildungen zeigen das Browserfenster vor und nach dem Mausklick.

Listing 2.23 left.aspx

```

<script runat="server">

void Link_Click(object sender, EventArgs e)
{
    FrameRedirect("right2.aspx", "right");
}

void FrameRedirect(string url, string target)
{
    string script = "<script language=\"JavaScript\">";
    script += "parent.frames[\"" + target + "\"].location.href=\"" + url + "\";";
    script += "</script>";

    if(!IsStartupScriptRegistered("redirect"))
        this.RegisterStartupScript("redirect", script);
}

</script>

<form runat="server">

    <ASP:LinkButton runat="server"
        Text="Rechten Frame wechseln"
        OnClick="Link_Click"/>

</form>

```

**Abbildung 2.16** Der anfängliche Zustand der beiden Frames



Abbildung 2.17 Nach dem Mausklick wurde die rechte Seite per JavaScript aktualisiert.

2.15 ... einen Redirect serverseitig durchführen?

Alternativ zum clientseitigen Redirect mittels des Protokolls HTTP können Sie auch eine serverseitige Umleitung vornehmen. Die Methode `Server.Transfer` verhält sich analog zum Redirect, lässt den Client die Umleitung jedoch nicht merken.

Listing 2.24 transfer1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Server.Transfer("zweiteseite.aspx");
}
```

Neben einem kleinen Geschwindigkeitsvorteil durch Wegfall eines zweiten Verbindungsaufbaus durch den Client hat diese Variante einen weiteren Vorteil. Sie können auf Informationen der ursprünglich aufgerufenen Seite zugreifen. Dies ist beispielsweise dann interessant, wenn dort Eingaben durch den Benutzer vorgenommen wurden.

Die Eigenschaft `Context.Handler` liefert die ursprünglich aufgerufene Seite. Sind hier beispielsweise öffentliche Eigenschaften implementiert, können diese zum Austausch von Informationen genutzt werden. Die beiden Listings zeigen dies anhand eines Eingabefeldes auf der ersten Seite. Ein Klick auf den Button führt eine Umleitung auf die zweite Seite durch. Die Eingabe des Benutzers steht dank der beschriebenen Möglichkeiten zur Verfügung.

Listing 2.25 Handler1.aspx

```

<script runat=server>

public string UserInput
{
    get { return(tb_input.Text); }
}

void myButton_Click(object sender, EventArgs e)
{
    Server.Transfer("handler1b.aspx");
}

</script>

<form runat="server">

<ASP:TextBox id="tb_input" runat="server"/>

<ASP:Button id="myButton" runat="server"
    Text="OK"
    OnClick="myButton_Click"/>

</form>

```

Damit die zweite Seite die Informationen aus der ersten auslesen kann, muss eine entsprechende Referenz eingefügt werden. So kann die notwendige Typenkonvertierung durchgeführt werden. Alternativ könnte per Reflection zugegriffen werden.

Listing 2.26 Handler1b.aspx

```

<% @Reference page="handler1.aspx" %>

<script runat=server>

void Page_Load(object sender, EventArgs e)
{
    Handler1Page p = (Handler1Page) Context.Handler;
    lb_input.Text = p.UserInput;
}

</script>

<p> Sie haben eingegeben:
    <ASP:Label id="lb_input" runat="server"/>
</p>

```

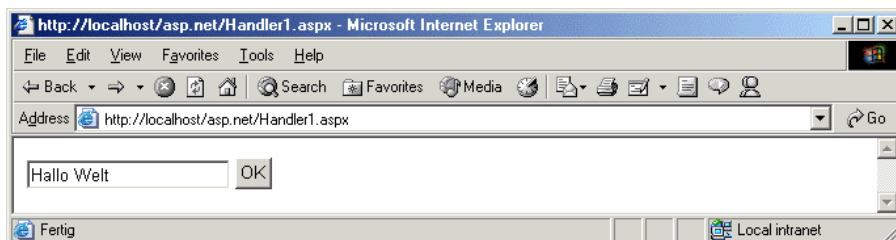


Abbildung 2.18 Die Eingabe eines Textes erfolgt auf der ersten Seite.

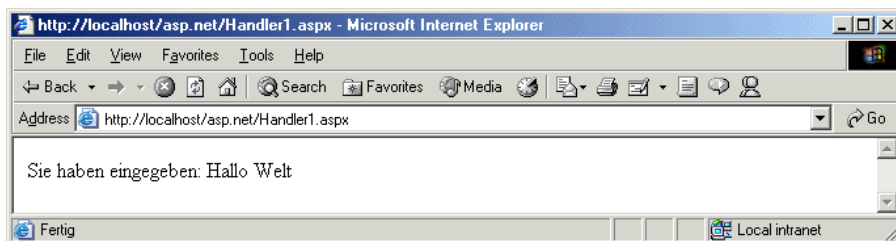


Abbildung 2.19 Innerhalb der zweiten Seite wird der Text im Browser ausgegeben.

Eine weitere Methode zur Umleitung lautet `Server.Execute`. Anders als bei dem vorgestellten `Transfer` wird hier die erste Seite nach Beenden der zweiten weiter ausgeführt. Durch Einführung von Web Controls ist die Methode jedoch faktisch obsolet.

2.16 ... den Namen und die Adresse der aktuellen Seite ermitteln?

Die Klasse `HttpRequest` bietet diverse Möglichkeiten, auf den virtuellen und physikalischen Pfad der aufgerufenen Seite zuzugreifen. Auf diese Weise können Sie ermitteln, welche Seite aufgerufen wurde, und beispielsweise das Datum der letzten Änderungen ausgeben (vergleiche Rezept „... das Datum der letzten Änderung ausgeben“ im Kapitel „Mathematik und Berechnung“). Das Listing zeigt eine Übersicht der zur Verfügung stehenden Eigenschaften.

Listing 2.27 url1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<h1>Arbeiten mit URLs</h1>");
}
```

```

Response.Write("RawUrl = " + Request.RawUrl + "<br>");
Response.Write("Path = " + Request.Path + "<br>");
Response.Write("FilePath = " + Request.FilePath + "<br>");
Response.Write("PathInfo = " + Request.PathInfo + "<br>");
Response.Write("PhysicalPath = " + Request.PhysicalPath + "<br>");
Response.Write("ApplicationPath = " + Request.ApplicationPath +
    "<br>");
Response.Write("PhysicalApplicationPath = " +
    Request.PhysicalApplicationPath + "<br>");

Response.Write("<h2>Url</h2>");
Response.Write("AbsoluteUri = " + Request.Url.AbsoluteUri + "<br>");
Response.Write("AbsolutePath = " + Request.Url.AbsolutePath +
    "<br>");
Response.Write("Host = " + Request.Url.Host + "<br>");
Response.Write("Port = " + Request.Url.Port + "<br>");
}
</script>

```

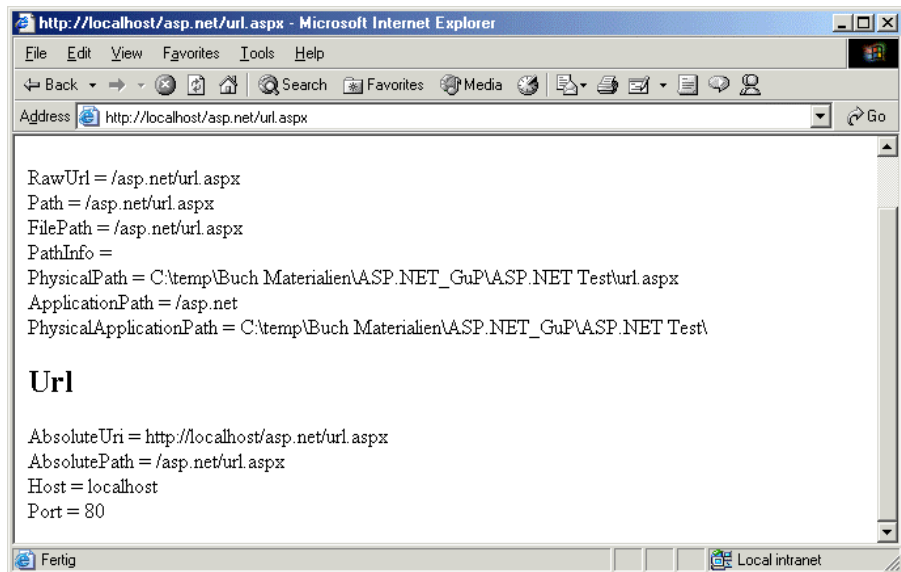


Abbildung 2.20 Detaillierte Informationen über die aufgerufene URL

2.17 ... die zuvor aufgerufene Seite ermitteln

Es ist mitunter sehr nützlich, die zuvor aufgerufene Seite zu ermitteln. So können Sie beispielsweise einen einfachen Rücksprung ohne JavaScript realisieren oder einfach nur eine Statistik erstellen.

Das Protokoll HTTP definiert eine Kopfzeile `referer`. Folgt der Benutzer einem Link, so wird die ursprüngliche Adresse beim Aufruf in dieser Kopfzeile übertragen. Sie können den Wert über die Eigenschaft `UrlReferrer` der Klasse `HttpRequest` abfragen. Zurückgeliefert wird eine Instanz der Klasse `Uri`. Diese liefert die einzelnen Segmente der URL als Eigenschaften. `AbsoluteUri` liefert die gesamte Adresse in Form einer Zeichenkette.

Das Listing zeigt die Abfrage der zuvor aufgerufenen Seite. Deren Adresse wird im Browserfenster ausgegeben und bildet die Basis für ein serverseitiges HyperLink-Control zum Rücksprung. Das Beispiel kommt mit einer aufrufenden Seite `referrer1.aspx`, die lediglich einen clientseitigen Link auf die zweite Seite enthält.

Listing 2.28 `referrer2.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("");
    lb.Text = Request.UrlReferrer.AbsoluteUri;
    link.NavigateUrl = Request.UrlReferrer.AbsoluteUri;
}

</script>

<p> Vor dieser Seite haben Sie folgende aufgerufen: </p>
<p> <ASP:Label id="lb" runat="server"/> </p>
<p> <ASP:HyperLink id="link" runat="server" text="Zurück"/> </p>
```



Double R or not? So ganz einig sind sich die Amerikaner nicht. Das Protokoll HTTP definiert den beschriebenen Kopfzeileneintrag als `referer`, also mit einem R. Das .NET Framework bietet den Zugriff auf diesen Wert jedoch über eine Eigenschaft `UrlReferrer`, also mit zwei R. Na ja, so ist es halt – einfach nicht verwirren lassen!



Beachten Sie, dass die Übergabe eines Referrers optional ist. Nicht jeder Browser kümmert sich um diese Kopfzeile, und selbst wenn, muss diese nicht unbedingt richtig sein. Ruft der Benutzer eine Seite direkt auf, ist der Referrer ohnehin leer. Möglicherweise liefert die Eigenschaft `UrlReferrer` daher `null` zurück. Vor der Verwendung sollten Sie also unbedingt auf `null` überprüfen!

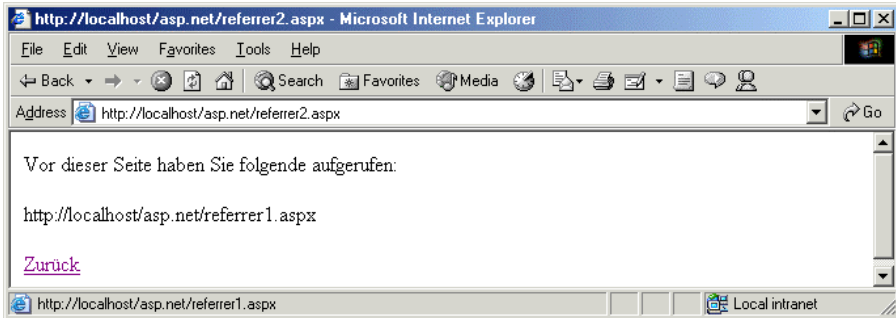


Abbildung 2.21 Über den Referrer können Sie ganz einfach zurückspringen.

2.18 ... die Ausführung einer Seite verzögern?

GMX macht es vor, alle anderen machen es nach. Um das böswillige Ausprobieren von Passwörtern zu erschweren, wird die Ausgabe der Statusseite bei Eingabe eines fehlerhaften Passwortes um mehrere Sekunden verzögert. Und auch in anderen Situationen kann die verzögerte Ausgabe durchaus sinnvoll sein.

Mit ASP.NET ist die kurze Unterbrechung ganz einfach. Die Basis bildet die Klasse `Thread` aus dem Namespace `System.Threading`. Diese repräsentiert den aktuellen Thread, also die Befehlsabfolge, in der die aktuelle Seite ausgeführt wird. Mit Hilfe der statischen Methode `Sleep` können Sie die Ausführung für eine angegebene Anzahl von Millisekunden anhalten.

Das Listing zeigt die Verwendung der Methode, die nach Einbindung des Namenspaces zur Verfügung steht. Vor der Unterbrechung wird ein Statustext ausgegeben. Der Aufruf der Methode `Response.Flush` sorgt dafür, dass dieser Text auch wirklich an den Browser übergeben wird, bevor die Ausführung unterbrochen wird.

Listing 2.29 Sleep1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("Bitte warten ...<br>");
    Response.Flush();
    Thread.Sleep(2000);
    Response.Write("Danke fürs Warten!<br>");
}

</script>
```

Ruft man die Seite im Browser auf, erscheint zunächst ein „Bitte warten ...“. Zwei Sekunden später wird die Abarbeitung der Seite abgeschlossen und zusätzlich der Text „Danke fürs Warten!“ ausgegeben.

2.19 ... eine Seite automatisch neu laden?

Eine Seite mittels JavaScript nach einer festgelegten Zeitspanne neu zu laden ist relativ einfach. Doch was tun, wenn der Benutzer möglicherweise kein JavaScript hat oder dieses deaktiviert ist? Das Protokoll HTTP kennt eine Alternative, und Sie können diese mittels ASP.NET sehr einfach nutzen. Es handelt sich um den Kopfzeileintrag `Refresh`. Dieser kann zusammen mit dem gewünschten Zeitintervall in Sekunden an den Client zurückgesendet werden.

Das Listing zeigt die Verwendung. Die Kopfzeile wird mit Hilfe der Methode `Response.AppendHeader` an die Server-Antwort angehängt. Im Beispiel wird die Seite alle fünf Sekunden aktualisiert. Erkennbar ist dies durch die Ausgabe der Uhrzeit, die so alle fünf Sekunden auf den aktuellen Stand gebracht wird.

Listing 2.30 Reload1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.AppendHeader("Refresh", "5");
}

</script>

<%= DateTime.Now.ToString() %>
```

Statt nur die aktuelle Seite neu einzuladen, können Sie auch eine alternative Adresse angeben, die nach Ablauf der Zeit automatisch aufgerufen wird. Auch diese Adresse wird innerhalb der Kopfzeile `Refresh` angegeben. Die Notation sieht wie folgt aus:

```
<Sekunden>;URL=<Adresse>
```

Das zweite Beispiel ruft nach zehn Sekunden die Website zum Buch, `www.aspbuch.de`, auf.

Listing 2.31 Reload2.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.AppendHeader("Refresh", "10;URL=http://www.asp-buch.de/");
}

</script>

<p> Gleich geht's weiter ... </p>

```



Das vorgestellte System kommt ohne JavaScript aus und wird von nahezu jedem Browser unterstützt. Es ist äquivalent zum Meta-Tag „Refresh“. Wie bei diesem Tag liefert der Kopfzeileneintrag keine Garantie, dass die Seite aktualisiert oder neu umgeleitet wird. Vielmehr kann der Benutzer dies vor Ablauf der Zeitspanne mittels Escape oder einem Klick auf den Stop-Button des Browsers verhindern.

2.20 ... einen beliebigen Inhalt an einer festgelegten Position ausgeben?

Sicher kennen Sie das Problem, an einer festgelegten Position innerhalb Ihrer ASP.NET-Seite einen bestimmten Inhalt ausgeben zu müssen. In Newsgroups wurde dies beispielsweise in Verbindung mit der dynamischen Ausgabe von HTML-Meta-Tags benannt. Das in alten ASP-Versionen gebräuchliche `Response.Write` hilft leider nicht weiter. Das Listing zeigt, warum.

Listing 2.32 Literal1.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<p>Hallo Welt</p>");
}

</script>

<h1> Überschrift </h1>

<p> Seitenfuß </p>

```

Statt den Text „Hallo Welt“ zwischen Überschrift und Fuß der Seite auszugeben, wird dieser an deren Beginn platziert. Die Abbildung zeigt dies. Die Ursache liegt in der Architektur von ASP.NET, da die Write-Methode die Daten direkt in den Puffer schreibt, die eigentliche Seite aber erst später gerendert wird.

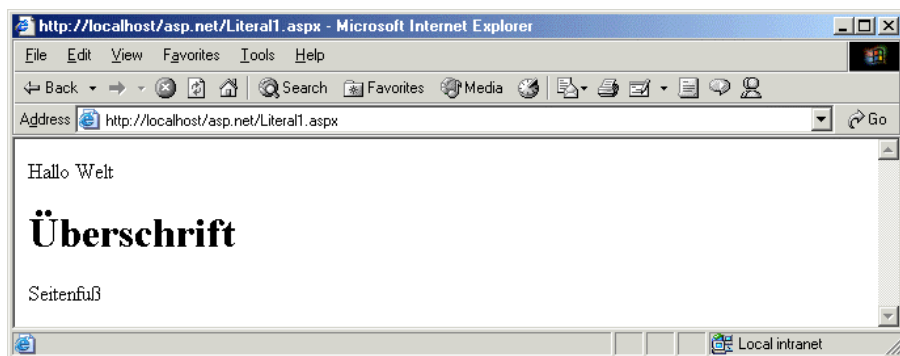


Abbildung 2.22 Die Ausgabe erfolgt an der falschen Position.

Es gibt unterschiedliche Ansätze, den Text an der richtigen Position auszugeben. Manche verwenden ein User Control, andere ein Placeholder, und wieder andere nutzen einfach das Label. Alle drei Varianten sind nicht zweifelsfrei zu empfehlen. So benötigt das User Control beispielsweise eine zusätzliche Datei, dem Placeholder müssen umständlich andere Unterobjekte angefügt werden, und das Label wird innerhalb eines unnötigen span-Tags ausgegeben.

Eine elegante Variante bietet das kaum direkt verwendete Literal-Control. Dieses wird immer dann implizit von ASP.NET genutzt, wenn statischer Text im Layout-Bereich einer Seite hinterlegt ist (was in annähernd 100 Prozent der Fall sein sollte). Sie können dieses Control jedoch auch direkt in der Seite ablegen und den Inhalt dynamisch über die Eigenschaft Text zuweisen.

Listing 2.33 Literal2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    lit.Text = "<p>Hallo Welt</p>";
}

</script>

<h1> Überschrift </h1>
<ASP:Literal id="lit" runat="server"/>
<p> Seitenfuß </p>
```

Auf diese Weise können Sie die Position bestimmen und müssen die beschriebenen Nachteile der anderen Ansatzmöglichkeiten nicht in Kauf nehmen. Die Abbildung zeigt das nun korrekte Ergebnis.

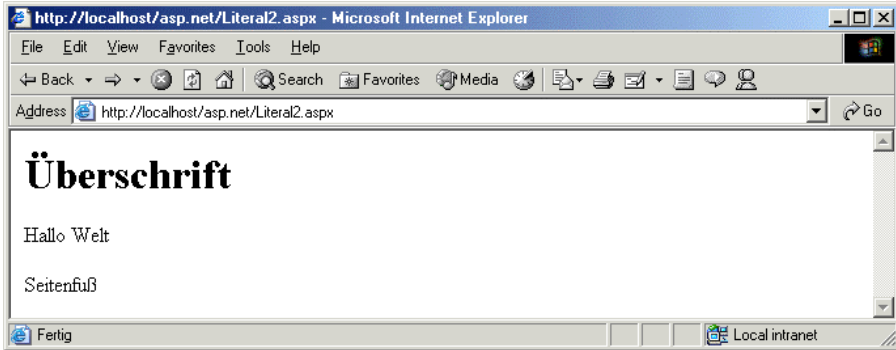


Abbildung 2.23 Der Text wird jetzt an der richtigen Position angezeigt.

2.21 ... Meta-Tags dynamisch übergeben?

Es ist sinnvoll, je nach Art des Besuchers und der Seite unterschiedliche Meta-Tags mit angepasstem Inhalt auszugeben. Auf diese Weise kann beispielsweise individuell auf Suchmaschinen und Spider reagiert werden.

Im Rezept „... einen beliebigen Inhalt an einer festgelegten Position ausgeben?“ haben Sie über eine einfache Möglichkeit gelesen, Inhalte an einer festen Position an den Client zu übergeben. Diesen Ansatz können Sie sich zur dynamischen Ausgabe von Meta-Tags zunutze machen. Das folgende Beispiel zeigt dies in einfacher Form.

Listing 2.34 MetaTag1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    meta1.Text = "<meta name=\"description\" content=\"Beschreibung\"/>";
}

</script>

<html>
<head>
```

```
<title>Meta-Tags</title>

<ASP:Literal id="metal" runat="server"/>

<head>

<body>
  <h1> Meta-Tags </h1>
</body>
</html>
```

Im Browser wird das Meta-Tag „description“ an der korrekten Stelle im HTML-Quelltext ausgegeben:

```
<html>
<head>

  <title>META-Tags</title>

  <meta name="description" content="Beschreibung"/>

</head>
...
```

Auf der anderen Seite ist eine derartige Übergabe der Meta-Tags gerade bei einer größeren Anzahl vielleicht etwas unkomfortabel. Ich habe daher ein kleines Custom Control geschrieben, das die Verwendung vereinfacht. Nachfolgend sehen Sie die Implementierung, die in erster Linie aus zwei öffentlichen Eigenschaften `Name` und `Content` sowie einer überschriebenen Methode `Render` zum Ausgeben des Controls besteht. Es wird die übergebene `HtmlTextWriter`-Instanz genutzt, um das HTML-Tag `meta` mitsamt den zwei Attributen auszugeben.

Listing 2.35 MetaTag.cs

```
using System;
using System.Web;
using System.Web.UI;

namespace PAL.Projects.AspNetKochbuch
{
  public class MetaTag : Control
  {
    public MetaTag() {}

    public MetaTag(string name)
    {
```

```
        this.Name = name;
    }

    public MetaTag(string name, string content)
    {
        this.Name = name;
        this.Content = content;
    }

    public string Name
    {
        get { return((string) ViewState["Name"]); }
        set { ViewState["Name"] = value; }
    }

    public string Content
    {
        get { return((string) ViewState["Content"]); }
        set { ViewState["Content"] = value; }
    }

    protected override void Render(HtmlTextWriter writer)
    {
        writer.AddAttribute("name", this.Name);
        writer.AddAttribute("description", this.Content);
        writer.RenderBeginTag("meta");
        writer.RenderEndTag();
    }
}
```

Nachdem Sie die Quellcode-Datei mittels `csc.exe` kompiliert und die so erzeugte DLL im `bin`-Verzeichnis der Web-Applikation abgelegt haben, können Sie das Control in Ihren Seiten verwenden. Das Listing zeigt den Einsatz. Es werden zwei Instanzen des Controls platziert. Die eine wird dynamisch bestückt, die andere enthält statisch vorgegebene Inhalte.

Listing 2.36 MetaTag2.aspx

```
<% @Register TagPrefix="PAL"
    Namespace="PAL.Projects.AspNetKochbuch" Assembly="MetaTag" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    meta1.Name = "description";
    meta1.Content = "Beschreibung";
```

```
}

</script>

<html>
<head>

    <title>Meta-Tags</title>

    <PAL:MetaTag id="meta1" runat="server"/>

    <PAL:MetaTag id="meta2" runat="server"
        Name="keywords" Content="ASP.NET, Meta, Meta-Tag"/>

</head>

<body>
    <h1> Meta-Tags </h1>
</body>
</html>
```

Die Ausgabe des Beispiels im Browser sieht wie folgt aus:

```
<html>
<head>

    <title>Meta-Tags</title>

    <meta name="description" description="Beschreibung" />
    <meta name="keywords" description="ASP.NET, Meta, Meta-Tag" />

</head>
...
```

Die wahre Stärke spielt das Control jedoch erst bei der vollkommen dynamischen Erzeugung der Meta-Tags aus. Im folgenden Beispiel habe ich dazu ein Placeholder-Control im Kopfbereich der Seite hinterlegt. Diesem werden dynamisch neu angelegte Instanzen der Klasse `MetaTag` übergeben. Die bei der Realisierung des Controls programmierten Konstruktorüberladungen vereinfachen die Anlage.

Listing 2.37 MetaTag3.aspx

```
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
```

```
{
    // Variante I
    MetaTag metatag = new MetaTag();
    metatag.Name = "description";
    metatag.Content = "Beschreibung";
    metatags.Controls.Add(metatag);

    // Variante II
    for(int i=0; i<5; i++)
        metatags.Controls.Add(new MetaTag("Tag Nummer " + i.ToString(), ":-
    ))"));
}

</script>

<html>
<head>

    <title>Meta-Tags</title>

    <ASP:PlaceHolder id="metatags" runat="server"/>

</head>

<body>
    <h1> META-Tags </h1>
</body>
</html>
```

Die Ausgabe im Browser wurde durch die Schleife ein wenig erweitert. Die Meta-Tags wurden dynamisch angelegt – eine Suchmaschine könnte mit den Ausgaben im Beispiel allerdings wenig anfangen.

```
<html>
<head>

    <title>Meta-Tags</title>

    <meta name="description" description="Beschreibung" />
    <meta name="Tag Nummer 0" description=":-))" />
    <meta name="Tag Nummer 1" description=":-))" />
    <meta name="Tag Nummer 2" description=":-))" />
    <meta name="Tag Nummer 3" description=":-))" />
    <meta name="Tag Nummer 4" description=":-))" />

</head>
```



Die korrekte Verwendung von Meta-Tags ist für den Erfolg einer Website nicht ganz unwichtig. Wie viele andere Elemente werden auch diese von Spidern und Suchmaschinen genau unter die Lupe genommen. Eine Optimierung ist daher mehr als sinnvoll. Weitere Informationen finden Sie beim Online-Magazin Dr. Web unter folgender Adresse:

<http://www.drweb.de>

2.22 ... das Euro-Zeichen € im Browser ausgeben?

Seit dem 01.01.2002 ist der Euro offizielles Zahlungsmittel in Deutschland. Aller spätestens seit diesem Termin sollten sämtliche Preisauszeichnungen auf einer Website in Euro erfolgen. Besonders schön sieht das Währungszeichen des Euros aus. Doch wie lässt sich das „€“ im Browser ausgeben?

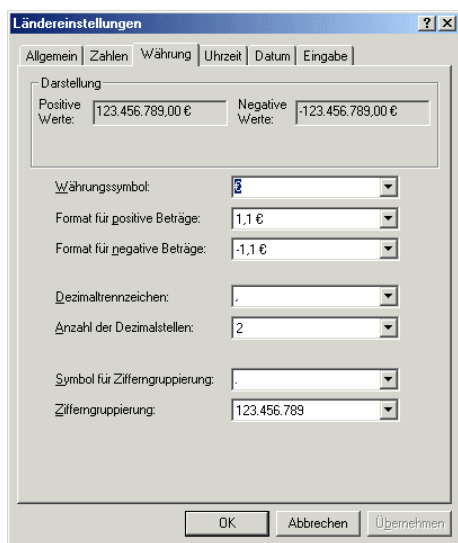


Abbildung 2.24 Die Ländereinstellungen

Bei VBScript in älteren ASP-Versionen stand die Funktion `FormatCurrency` zur Verfügung. Sofern keine andere Locale-ID explizit gesetzt wurde, wurden die Einstellungen des Systems und somit das dort hinterlegte Währungszeichen verwendet.

Listing 2.38 Euro1.asp

```
<% = FormatCurrency(49.9) %>
```


Über den Dialog “Ländereinstellungen” der Systemsteuerung kann das gewünschte Währungszeichen angegeben werden. Die Abbildung zeigt den Dialog. Anders als bei ASP 3.0 schien sich diese Einstellung jedoch in einigen Tests mit ASP.NET nicht auszuwirken. Hier erfolgt die Formatierung einer Währung wie im zweiten Listing gezeigt. Die Ausgabe erfolgt teilweise jedoch weiterhin mit „DM“.

Listing 2.39 Euro2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    double price = 49.9;
    Response.Write(string.Format("{0:C}", price));
}

</script>
```

Um das Währungszeichen gegebenenfalls manuell zu setzen, verwenden Sie die Klassen `CultureInfo` und `FormatNumberInfo`. `CultureInfo` aus dem Namespace `System.Globalization` repräsentiert alle länderspezifischen Einstellungen wie zum Beispiel den Kalender, aber auch die Währungsformatierung.

Eine neue Instanz der Klasse `CultureInfo` mit Vorgaben für ein bestimmtes Land kann mittels der statischen Methode `CultureInfo.CreateSpecificCulture` angelegt werden. Übergeben wird die Locale-ID des Landes, bei Deutschland „de“ (Österreich: „de-at“, Schweiz: „de-ch“). Über die Eigenschaft `NumberFormat.CurrencySymbol` kann das Währungszeichen nun individuell verändert werden. Damit sich die Formatierung auswirkt, wird die Instanz der Methode `string.Format` übergeben.

Listing 2.40 Euro3.aspx

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

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    CultureInfo culture = CultureInfo.CreateSpecificCulture("de");
    culture.NumberFormat.CurrencySymbol = "€";
    double price = 49.9;
    Response.Write(string.Format(culture, "{0:C}", price));
}

</script>
```

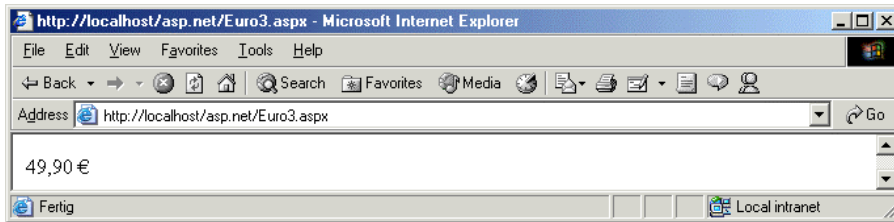


Abbildung 2.25 Die Ausgabe erfolgt nun mit dem Euro-Symbol.

Eine weitere Möglichkeit, die zu verwendenden Ländereinstellungen vorzugeben, bietet die `@Page`-Direktive. Hier kann die Locale-ID über das `Culture`-Attribut zugewiesen werden:

Listing 2.41 Euro5.aspx

```
<% @Page Language="C#" Debug="true" Culture="de-de" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    double price = 49.9;
    Response.Write(string.Format("{0:C}", price));
}

</script>
```

Natürlich können Sie das Euro-Zeichen auch ohne Verwendung der entsprechenden Formatierungsanweisungen ausgeben. Hierzu können Sie das Zeichen direkt im Editor eingeben. Sofern die Besucher Ihrer Website jedoch ein anderes Encoding eingestellt haben, wird das Zeichen mit großer Wahrscheinlichkeit durch ein anderes Sonderzeichen ersetzt. Sie sollten daher unbedingt die im folgenden Listing gezeigte Repräsentation des Zeichens verwenden.

Listing 2.42 Euro4.aspx

```
<p> Der Euro wird durch das Symbol &#8364; dargestellt. </p>

<p> Wird das €-Zeichen direkt eingegeben, ist die Darstellung unter
Umständen falsch. </p>
```

Das Listing zeigt neben der Repräsentation über den Unicode-Hexwert auch die direkte Eingabe im Editor. Die Abbildung zeigt das Ergebnis des Listings bei Auswahl eines griechischen Encoding. Während das erste Zeichen korrekt dargestellt wird, wurde das zweite durch Sonderzeichen ersetzt.

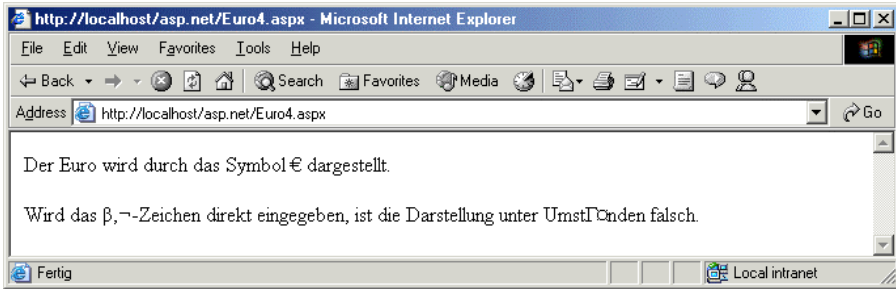


Abbildung 2.26 Das direkt eingegebene Symbol wird nicht korrekt angezeigt.

2.23 ... die Sprache des Besuchers erkennen?

Das Protokoll HTTP liefert mit dem Kopfzeileneintrag `Accept-Language` Informationen über die vom Benutzer gewählte Sprache. Im Internet Explorer können Sie diese beispielsweise in den Internet-Optionen konfigurieren. Die hinterlegten Werte werden Ihnen als string-Array über die Eigenschaft `Request.UserLanguages` angeboten.

Auf Basis des übertragenen ISO-Sprachcodes können Sie entscheiden, welche Sprache der Benutzer spricht. In der Regel werden Sie in Abhängigkeit von dieser einen Redirect auf das lokalisierte Internet-Angebot durchführen. Das Listing zeigt dies für die Sprachen Deutsch und Englisch. Sollte keine der beiden Sprachen zutreffen, können Sie eine manuelle Auswahl anschließen.

Listing 2.43 Language1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    foreach(string str in Request.UserLanguages)
    {
        if(str.IndexOf("de") != -1)
            Response.Redirect("/de/", true);
        else if(str.IndexOf("en") != -1)
            Response.Redirect("/en/", true);
    }
}

</script>

<h1>Bitte wählen Sie Ihre Sprache aus:</h1>
...
```

Das .NET Framework bietet im Namespace `System.Globalization` umfangreiche Möglichkeiten zur Unterstützung einer Lokalisierung an. So repräsentiert die Klasse `CultureInfo` beispielsweise alle spezifischen Landeseinstellungen einer Kultur. Hierzu gehören insbesondere auch Formatierungsanweisungen für `Dati`, `Zahlen` und `Währungen`.

Sie können die Klasse `CultureInfo` benutzen, um diese spezifischen Formatierungen zu unterstützen. Zur Instanziierung übergeben Sie der statischen Methode `CreateSpecificCulture` den gewünschten ISO-Sprachcode. Die Klasse unterstützt die Schnittstelle `IFormatProvider` und kann daher beispielsweise direkt zur Formatierung von Zeichenketten herangezogen werden.

Im Listing sehen Sie die Verwendung der Klasse. In Abhängigkeit von den Einstellungen im Client-Browser werden das aktuelle Datum, eine Zahl und ein Währungsbetrag ausgegeben. Leider wird der Euro bisher noch nicht direkt unterstützt. Hier ist auf eine Aktualisierung des Betriebssystems zu warten. Um das Beispiel auszuprobieren, können Sie einfach die Einstellung des Internet Explorers unter `Extras > Internet Optionen > Sprachen verändern`.

Listing 2.44 Language2.aspx

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

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    CultureInfo culture;

    try
    {
        culture = CultureInfo.CreateSpecificCulture(
            Request.UserLanguages[0]);
    }
    catch
    {
        culture = CultureInfo.CreateSpecificCulture("de");
    }

    if(culture.NumberFormat.CurrencySymbol == "DM")
        culture.NumberFormat.CurrencySymbol = "€";
    Response.Write(culture.DisplayName + "<br>");
    Response.Write(string.Format(culture, "{0:D}<br>", DateTime.Now));
    Response.Write(string.Format(culture, "{0:N}<br>", 123.45));
    Response.Write(string.Format(culture, "{0:C}<br>", 123.45));
}

</script>
```

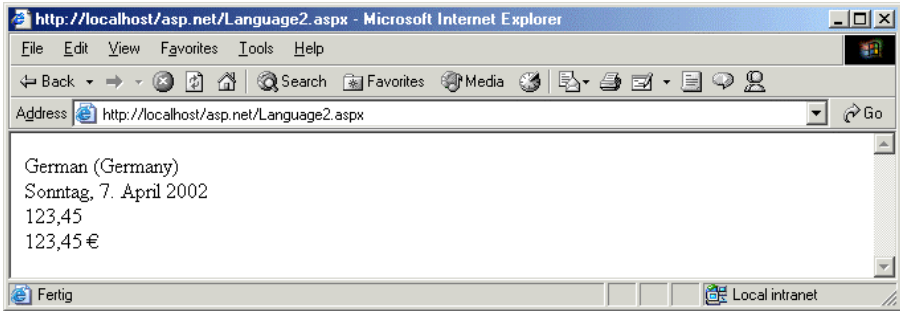


Abbildung 2.27 Als Sprache wurde Deutsch ausgewählt ...

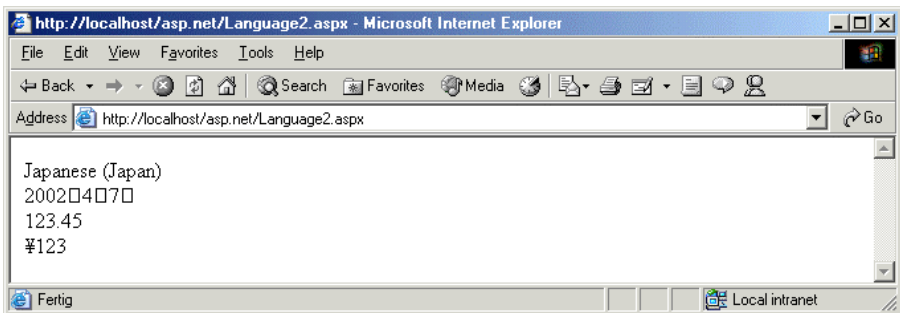


Abbildung 2.28 ... und nun Japanisch.

Da die Einstellung abhängig vom Benutzer ist, bietet es sich an, die so erzeugte Instanz der Klasse `CultureInfo` in einer Session-Variablen abzulegen. Dies ist auch in Verbindung mit dem `SessionState-Service` sowie dem `SQL-Server` möglich, da die Klasse als `serialisierbar` gekennzeichnet ist (Attribut `Serializable`).



Eine Übersicht der verfügbaren ISO-Sprachcodes finden Sie in der Beschreibung der Klasse `CultureInfo` innerhalb der Microsoft .NET Framework-Dokumentation.

2.24 ... eine eindeutige ID erstellen?

Eine eindeutige Kennzeichnung wird bei Datenbanken zur Identifizierung eines Datensatzes verwendet. Dort kann die Datenbank-Engine selbst die Unique-ID erzeugen. Doch auch an anderen Stellen im Programmverlauf wird oftmals eine Kennzeichnung benötigt.

Das Betriebssystem verwendet zur Identifizierung beispielsweise von Komponenten einen so genannten global unique Identifier, kurz GUID. Es handelt sich um einen 128-Bit-int-Wert (16 Byte), der weltweit eindeutig ist. Dies wird durch eine komplexe Berechnung unter Einbeziehung der aktuellen Zeit sowie der eindeutigen MAC-Adresse der Netzwerkkarte erreicht.

Das .NET Framework bietet eine Klasse `Guid` an, über deren statische Methode `NewGuid` Sie eine neue, eindeutige `Guid` anlegen können. Das Beispiel zeigt die Verwendung der Klasse sowie deren Ausgabe als Zeichenkette.

Listing 2.45 GUID1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Guid guid = Guid.NewGuid();
    Response.Write(guid.ToString("N"));
}

</script>
```

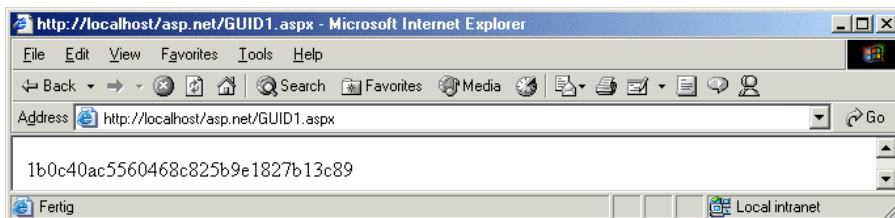


Abbildung 2.29 Eine neue GUID

Die Abbildung zeigt eine neu erstellte GUID in Form einer Zeichenkette. Natürlich können Sie auf dieser Basis auch einen eindeutigen int-Wert erstellen. Hierzu verwenden Sie die von `object` geerbte und überschriebene Methode `GetHashCode`:

Listing 2.46 GUID2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Guid guid = Guid.NewGuid();
    Response.Write(guid.GetHashCode().ToString());
}

</script>
```

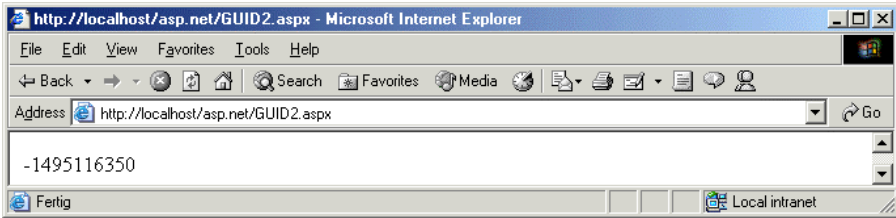


Abbildung 2.30 Auch ein eindeutiger int-Wert lässt sich erzeugen.

2.25 ... feststellen, ob der Benutzer noch verbunden ist?

Wenn Sie langwierige Operationen durchführen, können Sie von Zeit zu Zeit abfragen, ob der Benutzer überhaupt auf ein Ergebnis wartet. Die Eigenschaft `Response.IsClientConnected` gibt an, ob der Client auf Protokollebene (TCP/IP) noch mit dem Server verbunden ist. Ist dies nicht mehr der Fall, hat sich die Operation möglicherweise bereits erledigt.

Das Listing zeigt die Verwendung der Eigenschaft. Es werden so lange Daten an den Client ausgegeben, wie dieser „lauscht“. Die Ausgabe lässt sich erst durch Beenden der Verbindung am Browser stoppen.

Listing 2.47 `IsClientConnected1.aspx`

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<h1>IsClientConnected?</h1>");

    while(Response.IsClientConnected)
    {
        Response.Write(". ");
        Response.Flush();
    }
}

</script>
```

2.26 ... einen Broken Link abfangen?

Der Statuscode 404 des Protokolls HTTP ist nahezu legendär. Wann immer die vom Client angeforderte Seite vom Server nicht gefunden wird, gibt dieser den genannten Code zurück. Im Browser wird dann oftmals eine unschöne Fehlermeldung angezeigt. Sie haben die Möglichkeit, dies abzufangen und dem Benutzer stattdessen eine übersichtliche Seite im Corporate Design mit dem Fehlerhinweis anzubieten. Hier können Sie ihm auch Möglichkeiten anbieten, wie er weiter vorgehen soll, eine Sitemap anzeigen oder Ähnliches.

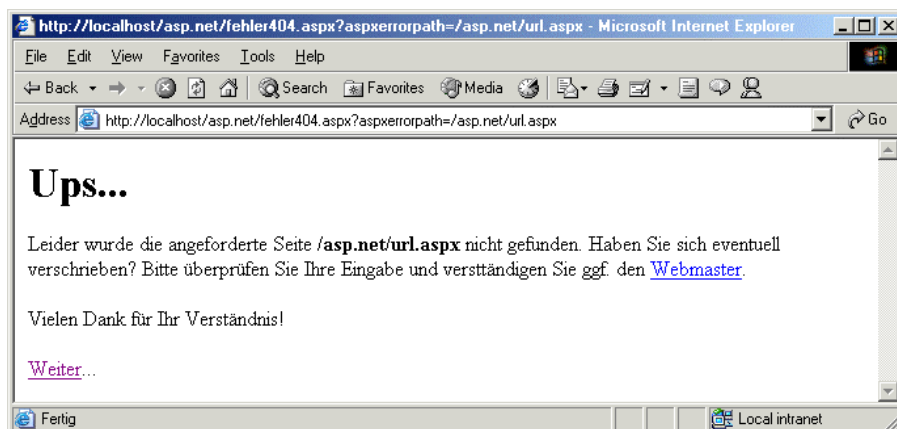


Abbildung 2.31 Die gewünschte Seite wurde nicht gefunden.

Der ASP.NET-Weg

Bei ASP.NET definieren Sie in der Konfigurationsdatei `web.config` eine andere Fehlerseite, die aufgerufen werden soll, sobald die angeforderte Seite nicht gefunden wird. Hierzu wird ein Redirect auf die angegebene URL durchgeführt. Die Konfiguration sieht wie folgt aus:

Listing 2.48 web.config

```
<configuration>
  <system.web>
    <customErrors mode="On">
      <error statusCode="404" redirect="/fehler404.aspx"/>
    </customErrors>
  </system.web>
</configuration>
```


Um in der Fehlerseite zu ermitteln, welche Seite ursprünglich aufgerufen wurde, wird deren URL als Query-String-Variable „aspxerrorpath“ übergeben. Sie können über die übliche `NameValueCollection` auf den Wert zugreifen:

```
errpage = Request.QueryString["aspxerrorpath"];
```

Die Abbildung zeigt das konfigurierte Fehlermanagement im Einsatz.

Der universelle Weg

Der gezeigte Weg hat einen Haken. Er funktioniert nur in Verbindung mit ASP.NET und Dateieendungen, die mit der ASP.NET-Engine verknüpft sind. Würde beispielsweise eine .html-Datei angefordert, so würde wieder die übliche Fehlermeldung der Internet Information Services beziehungsweise des Browsers angezeigt werden.

Um die Fehlerseite für alle beliebigen Dateien zu registrieren, müssen Sie diese in der Konfiguration der IIS für den Statuscode 404 als „Benutzerdefinierte Fehler“ angeben. Auch hier wird die ursprünglich aufgerufene Seite im Query-String übergeben, allerdings leicht anders. Die Fehlerseite sollte daher tolerant reagieren, um beide Fälle zu unterstützen.

```
errpage = Request.QueryString["aspxerrorpath"];  
  
if(errpage == null)  
{  
    errpage = Request.ServerVariables["QUERY_STRING"];  
    errpage = errpage.Replace("404;", " ");  
}
```

Ist die Fehlerseite in der Konfiguration der IIS angegeben, so wird diese ab sofort für sämtliche nicht gefundenen Dateien angezeigt. Die Abbildung zeigt den Konfigurationsdialog.

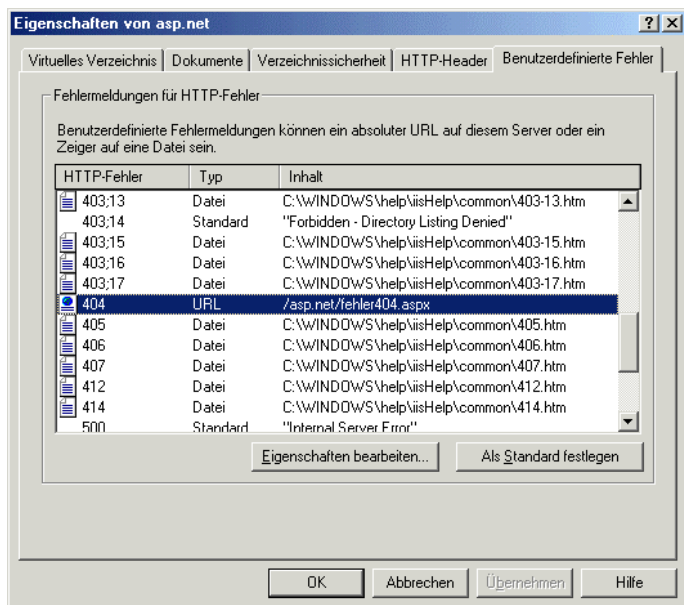


Abbildung 2.32 Die Fehlerseite muss in der IIS-Konfiguration hinterlegt werden.

2.27 ... Broken-Links finden und beseitigen?

Einen auftretenden Fehler 404 ein wenig zu entschärfen ist eine Sache. Besser ist jedoch, wenn derartige Fehler nach Möglichkeit gar nicht erst auftreten.

Broken Links passiv finden

Um so genannte Broken Links dauerhaft zu eliminieren, können Sie sich beispielsweise eine Email senden, sobald die oben gezeigte Fehlerseite aufgerufen wird. Allerdings sollten Sie eine Selektion vornehmen, denn nur wenn der Besucher von einer Ihrer Seiten kommt, handelt es sich auch um einen Broken Link innerhalb Ihres Angebots. Ansonsten hat sich der Besucher vielleicht bei der Eingabe im Browser vertippt.

Um innerhalb der Fehlerseite abzufragen, ob der Besucher von einer Seite innerhalb des Angebots kommt, können Sie die Server-Variable „HTTP_REFERER“ auswerten. Enthält diese Ihren Domain-Namen, versenden Sie die Email, ansonsten nicht.

Listing 2.49 fehler404.aspx

```
...  
  
string referer = Request.ServerVariables["HTTP_REFERER"];  
if(referer != null && referer.ToLower().IndexOf("localhost") != -1)  
{  
    MailMessage mail = new MailMessage();  
    mail.From = "webmaster@domain.tld";  
    mail.To = "webmaster@domain.tld";  
    mail.Subject = "Wichtig: Broken Link entdeckt!";  
    mail.Body = string.Format("URL: {0}, Referer: {1}",  
        errpage, referer);  
    SmtpMail.Send(mail);  
}  
  
...
```

Wird ein falscher Link innerhalb des Internet-Angebotes entdeckt, erhalten Sie nun eine Email und können den Link kurzfristig korrigieren.

Broken Links aktiv finden

Noch besser als eine schnelle Reaktion ist das aktive Vorbeugen gegen Broken Links. Das hat prinzipiell nichts mit ASP.NET zu tun, ist aber in diesem Zusammenhang dennoch interessant. Es gibt unterschiedliche Software-Programme, die beginnend bei der Startseite eines Angebots alle Unterseiten wie ein Spider rekursiv durchlaufen und alle enthaltenen Links prüfen. Broken Links werden erkannt und können direkt behoben werden.

Eine derartige Software ist auch Xenu Link Sleuth, das von einem deutschen Entwickler kostenlos angeboten wird. Die englischsprachige Software geht wie beschrieben vor und erstellt einen übersichtlichen Report.

Die jeweils aktuelle Version der Software erhalten Sie kostenlos auf der Website des Autoren. Sie finden diese unter folgender Adresse:

```
http://home.snafu.de/tilman/xenulink.html
```

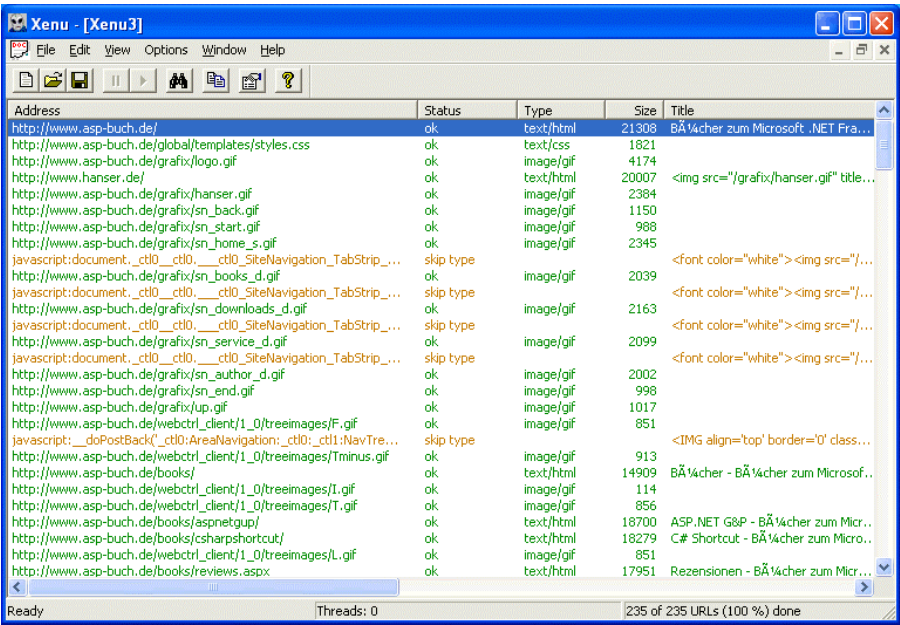


Abbildung 2.33 Glück gehabt, keine Broken Links!

2.28 ... Broken Links bei Bildern verhindern?

Ist ein eingebundenes Bild auf dem Server nicht vorhanden, zeigt der Browser meist ein eher hässliches Kreuz an. Schöner wäre es, wenn stattdessen eine Standardgrafik mit einem Hinweis eingeblendet werden würde. Mit ASP.NET kein Problem. Hierzu muss lediglich das Beispiel aus dem Rezept „... einen Broken Link abfangen?“ ein klein wenig erweitert werden. Dabei ist es unbedingt erforderlich, dass die Seite – wie im Rezept beschrieben – in der Konfiguration der Internet Information Services als Fehlerseite 404 hinterlegt ist.

Die Seite muss um eine spezielle Behandlung von Grafiken erweitert werden. Dazu muss die Endung der vom Client angeforderten, aber nicht vorhandenen Datei abgefragt werden. Handelt es sich um ein GIF- oder JPEG-Bild, wird eine hinterlegte Grafik zurückgesandt, ansonsten erfolgt die Ausgabe des üblichen Textes.

Listing 2.50 fehler404.aspx

```

<% @Import Namespace="System.IO" %>

<script runat="server">

string errpage;

void Page_Load(object sender, EventArgs e)
{
    errpage = Request.QueryString["aspxerrorpath"];

    if(errpage == null)
    {
        errpage = Request.ServerVariables["QUERY_STRING"];
        errpage = errpage.Replace("404;", " ");
    }

    string extension = Path.GetExtension(errpage);
    if(extension == ".gif" || extension == ".jpg")
    {
        Response.ContentType = "image/gif";
        Response.WriteFile("404.gif");
        Response.End();
    }

    DataBind();
}

</script>

<h1> Ups... </h1>

<p>
    Leider wurde die angeforderte Seite <b><%# errpage %></b> nicht
    gefunden. Haben Sie sich eventuell verschrieben? Bitte überprüfen
    Sie Ihre Eingabe, und verständigen Sie ggf. den
    <a href="mailto:webmaster@here.com">Webmaster</a>.
</p>

<p> Vielen Dank für Ihr Verständnis! </p>

<p> <a href="/">Weiter</a>... </p>

```

Die Abbildung zeigt den Einsatz der 404-Grafik. Die ASP.NET-Seite bindet eine Grafik ein, die nicht existiert. Statt derer wird die hinterlegte Warngrafik angezeigt. Natürlich können Sie sich auch hier automatisch per Email informieren lassen, sobald eine fehlende Grafik gefunden wurde. Informationen hierzu finden Sie im Rezept „... Broken-Links finden und beseitigen?“.

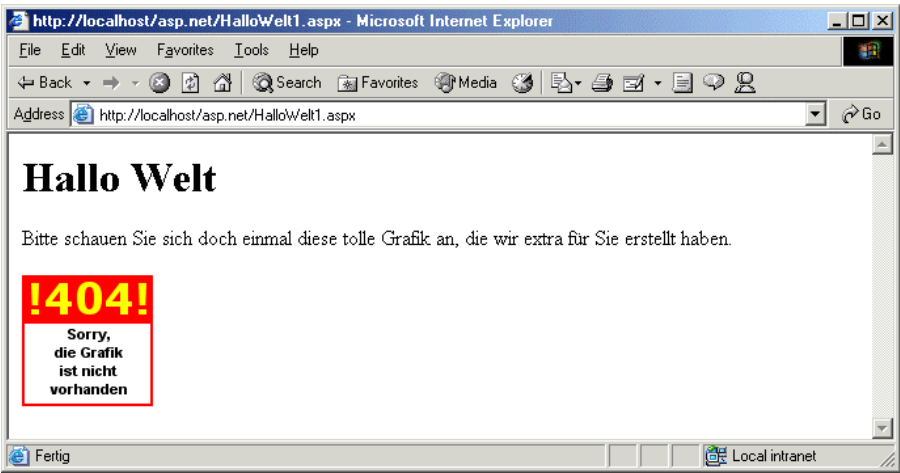


Abbildung 2.34 Die Grafik ist leider nicht vorhanden.

2.29 ... eine Liste aller Content-Types finden?

Content-Types sind sehr wichtig, denn sie informieren den Browser über die Art der empfangenen Daten. Über die Kennzeichnung muss dieser letztlich entscheiden, ob es sich um eine HTML-Seite, eine Grafik, ein PDF oder ein anderes Format handelt. Jeder Content-Type besteht aus einem Haupttyp wie zum Beispiel „text“ und einem mittels Slash getrennten Untertypen wie beispielsweise „html“. Der Content-Type für HTML-Daten lautet somit „text/html“. Es wird dabei prinzipiell die Kleinschreibung verwendet.

Eine ständig aktualisierte Liste der von der IANA festgelegten Typen finden Sie im Internet unter folgender Adresse:

```
ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types
```

Die folgende Liste zeigt eine definitiv unvollständige Aufzählung einiger wichtiger Typen, die Ihnen häufiger begegnen werden. Die Tabelle zeigt jeweils auch eine kurze Beschreibung des Typs.

Tabelle 2.1 Wichtige Content-Typen in der Übersicht

Haupttyp	Untertyp	Beschreibung
text	plain	Purer Text (ohne Formatierungen)
	html	HTML-Seite
	css	Cascading Style Sheet

Haupttyp	Untertyp	Beschreibung
	xml	XML-Daten
	rtf	RTF-Text
application	pdf	PDF-Datei
	zip	ZIP-Datei
	msword	Microsoft Word-Datei (doc)
image	jpeg	JPEG-Grafik
	gif	GIF-Grafik
	tiff	TIFF-Grafik
	png	PNG-Grafik
audio	mpeg	MPEG-kodiertes Audiosignal
	mp3	MP3
	wave	WAVE-Format
video	mpeg	MPEG-Video
	quicktime	QuickTime

Diese Typenangaben werden Sie insbesondere benötigen, wenn Sie andere als HTML-Daten zurückliefern wollen. Soll eine Seite beispielsweise dynamisch eine Grafik erstellen und diese an den Browser senden, so müssen Sie den entsprechenden Content-Type explizit über die gleichnamige Eigenschaft der Response-Klasse setzen, und zwar bevor Sie die Daten an den Client senden. Dies ist erforderlich, da über die Eigenschaft der korrespondierende Kopfzeileneintrag der Server-Antwort gesetzt wird, und diese Informationen werden bekanntermaßen immer vor den eigentlichen Daten übertragen.

Das folgende Listing zeigt das einfache Zurücksenden einer Grafik an den Client. Im vorliegenden Fall wird einfach eine lokale Datei geöffnet und mittels `WriteFile` übertragen. Selbstverständlich könnten Sie die Grafik oder ein anderes Format auch on the fly erstellen. Diverse Beispiele hierzu finden Sie im Kapitel „Grafik“ und auch in anderen Rezepten dieses Buches.

Listing 2.51 ContentType1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.ContentType = "image/gif";
    Response.WriteFile("vivendi-universal.gif");
}

</script>
```

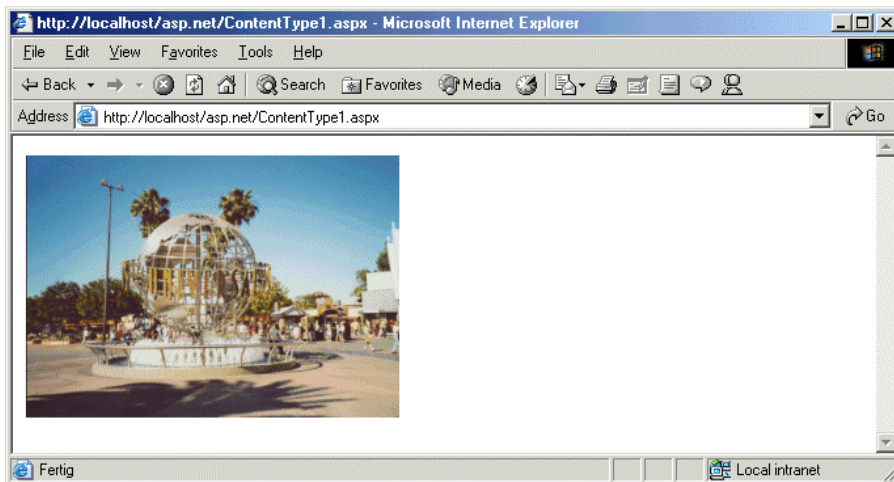


Abbildung 2.35 Die zurückgelieferten Daten werden als Bild angezeigt.

Die Abbildung zeigt, dass die vom Server übertragenen Daten mit Hilfe des korrekten Content-Types als Bild dargestellt werden.

2.30 ... die Standard-Programmiersprache ändern?

Nach der Installation von ASP.NET ist Visual Basic .NET als Standardsprache für alle Seiten gewählt. Davon abweichende Werte müssen auf jeder Seite manuell innerhalb der `@Page`-Direktive angegeben werden. Aber muss das wirklich sein? Nein, denn Sie können die Sprache auch global oder aber nur spezifisch für eine ausgewählte Web-Applikation ändern.

Standardsprache global ändern

Die Standardsprache ist in der zentralen Konfigurationsdatei `machine.config` hinterlegt. Sie können diese einfach mit einem Editor manipulieren und eine beliebige andere Sprache eintragen. Fortan wird diese Sprache als Standard verwendet.

Öffnen Sie die Datei aus dem folgenden Verzeichnis:

```
<WINDIR>\Microsoft.NET\Framework\<Version>\config
```

Suchen Sie nun den Abschnitt `compilation`. Unter dem `defaultLanguage` hinterlegen Sie nun die gewünschte Sprache. Mögliche Werte sind beispielsweise folgende:

- CS für C#
- VB für Visual Basic .NET
- JS für JScript .NET

```
<compilation
  debug="false"
  explicit="true"
  defaultLanguage="CS">
```

Daneben können Sie auch das Kürzel beliebiger anderer Sprachen eintragen, die im untergeordneten Abschnitt `compilers` registriert wurden. Hier wird eine von `CodeDomProvider` abgeleitete Klasse und die entsprechende Assembly aus dem Global Assembly Cache (GAC) angegeben. Möchten Sie weitere Sprachen hinzufügen, können Sie diese hier registrieren.



Die Änderung der Standardsprache ist wirklich einfach und schnell durchgeführt. Bedenken Sie jedoch immer die Konsequenzen. Befinden sich bereits bestehende ASP.NET-Seiten auf dem System, bei denen die Sprache nicht explizit angegeben wurde, so werden diese Seiten nicht mehr funktionieren. Das ist nur logisch, schließlich können Visual Basic .NET-Anweisungen nur schwerlich mit dem C#-Kompiler übersetzt werden. Wenn Sie die globale Änderung scheuen, können Sie die Sprache auch individuell für jede Web-Applikation ändern.

Standardsprache für eine Web-Applikation ändern

Die Änderung der Sprache separat für jede Web-Applikation wird in der Konfigurationsdatei `web.config` vorgenommen. Ähnlich der objektorientierten Programmierung erbt diese die globalen Einstellungen der `machine.config` und überschreibt die gewünschten Werte nur noch. Ergo wird die Konfiguration dort ganz analog vorgenommen.

Listing 2.52 web.config

```
<configuration>
  <system.web>
    <compilation
      debug="true"
      defaultLanguage="CS"/>
    </system.web>
  </configuration>
```

Wie Sie sehen, können Sie hier auch sehr bequem den Debug-Modus für die gesamte Web-Applikation einschalten. Auch dies ist global möglich, Sie sollten dies aber tunlichst vermeiden.

2.31 ... mehrere Sprachen innerhalb einer ASP.NET-Seite verwenden?

Gar nicht! Tut mir leid, aber das ist aufgrund der Architektur leider ausgeschlossen. Beim Aufruf einer ASP.NET-Seite werden der enthaltene Quellcode sowie die HTML-Anweisungen in eine Quellcode-Datei umgewandelt. Sie können dies sehr schön sehen, indem Sie sich unterhalb des Verzeichnisses ...

```
<WINDIR>\Microsoft.NET\Framework\<Version>\Temporary ASP.NET Files
```

... die temporär erstellten Dateien anschauen. Jede ASP.NET-Seite landet letztlich hier und wird beim ersten Aufruf in eine DLL kopiert. Alle Anfragen werden bis zur Änderung der Seite aus eben dieser DLL bedient. Riskieren Sie ruhig einmal einen Blick, es lohnt sich.

Wie es doch geht ...

Auch wenn Sie innerhalb einer Seite direkt nur eine Sprache verwenden können, gibt es natürlich Möglichkeiten, sprachübergreifende Web-Applikationen zu realisieren. Um genau zu sein, sind es drei an der Zahl:

- Sie können eine Code Behind-Datei in einer Sprache erstellen und die davon abgeleitete aspx-Datei in einer anderen verfassen.
- Sie können innerhalb Ihrer Seite User Controls verwenden, die Quellcode in einer anderen Sprache enthalten. Diese wird dort über das Attribut `Language` der `@Control`-Direktive festgelegt.
- Sie können innerhalb Ihrer Seite Klassen aus hinterlegten Assemblies verwenden, die in einer beliebigen .NET-Sprache entwickelt wurden. Die Assemblies können beispielsweise im bin-Verzeichnis abgelegt sein oder über die `@Assembly`-Direktive referenziert werden.

2.32 ... eine Code Behind-Datei kompilieren?

Oft werden Code Behind-Dateien als Quellcode-Dateien abgelegt und automatisch von der ASP.NET-Engine mitsamt der eigentlichen Seite kompiliert. Verantwortlich hierfür ist das Src-Attribut der @Page-Direktive. Alternativ können Sie eine oder mehrere Quellcode-Dateien auch manuell kompilieren und als DLL im bin-Verzeichnis Ihrer Web-Applikation ablegen. Mitunter kann dies einfacher sein als der andere Weg, da die dort abgelegten Assemblies automatisch geladen werden und somit in jeder Seite zur Verfügung stehen.

Das Kompilieren erfolgt mit Hilfe des sprachenabhängigen Kommandozeilenprogramms. Bei C# benutzen Sie `csc.exe`, bei Visual Basic .NET `vbc.exe`. Die wesentlichen Parameter wurden analog implementiert, so dass Sie lediglich im einen Fall die `.cs`- und im anderen die `.vb`-Dateien übergeben. Die nachfolgenden Beschreibungen beziehen sich der Einfachheit halber auf C#, die jeweiligen Parameter lassen sich jedoch eins zu eins auf den Visual Basic .NET-Kompiler anwenden.

Kompilieren einer einzelnen Quellcode-Datei

Die beiden nachfolgenden Listings zeigen eine ASP.NET-Seite mit einem Button sowie eine entsprechende Code Behind-Datei. Hier ist eine Ereignisbehandlung für den Button implementiert, die in der Ausgabe eines kurzen Textes resultiert.

Listing 2.53 test.aspx

```
<% @Page Language="C#" Debug="true" Inherits="TestPage" %>

<form runat="server">

  <ASP:Button id="bt" runat="server"
    text="Klick mich!"
    onclick="bt_click"/>

</form>
```

Listing 2.54 test.aspx.cs

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public class TestPage : Page
{
    protected Button bt;
```

```
protected void bt_click(object sender, EventArgs e)
{
    Response.Write("Danke!");
}
}
```

Beachten Sie bitte, dass in der `@Page`-Direktive zwar der Name der Code Behind-Klasse notiert ist (Attribut `Inherits`), nicht aber die entsprechende Quellcode-Datei (Attribut `Src`). Diese muss daher vor dem Aufruf des Beispiels kompiliert werden. Da es sich um eine C#-Klasse handelt, kommt das Kommandozeilenprogramm `csc.exe` zum Einsatz.

Der Compiler erstellt standardmäßig `exe`-Dateien. Für ASP.NET wird jedoch eine DLL benötigt, so dass über den Parameter `/t` das Ziel `library` angegeben werden muss. Anschließend wird der Name der zu kompilierenden Datei übergeben:

```
csc /t:library test.aspx.cs
```

Wechseln Sie zur Eingabe der Zeile zunächst in die Eingabeaufforderung `cmd.exe` und hier mittels `cd` in das physikalische Verzeichnis Ihrer Web-Applikation. Die Abbildung zeigt den Ablauf des Compilers.

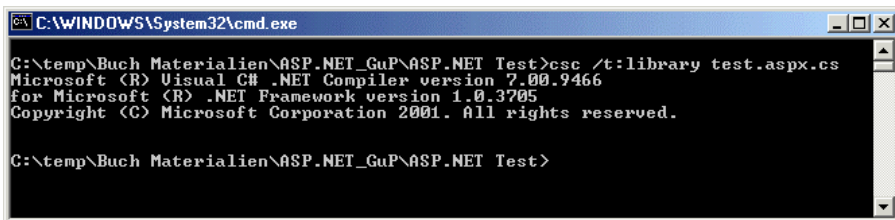


Abbildung 2.36 Das Programm `csc.exe` kompiliert die Quellcode-Datei.

Sie erhalten im aktuellen Verzeichnis eine DLL mit dem Namen der Quellcode-Datei, in diesem Fall `test.aspx.dll`. Verschieben Sie diese Datei in das Unterverzeichnis `bin`. Sollte dieses noch nicht existieren, legen Sie es neu an. Beachten Sie dabei, dass Sie sich *unbedingt* im Hauptverzeichnis Ihrer Web-Applikation und *nicht* in einem Unterverzeichnis befinden müssen. Anschließend können Sie die oben erstellte ASP.NET-Seite aufrufen.

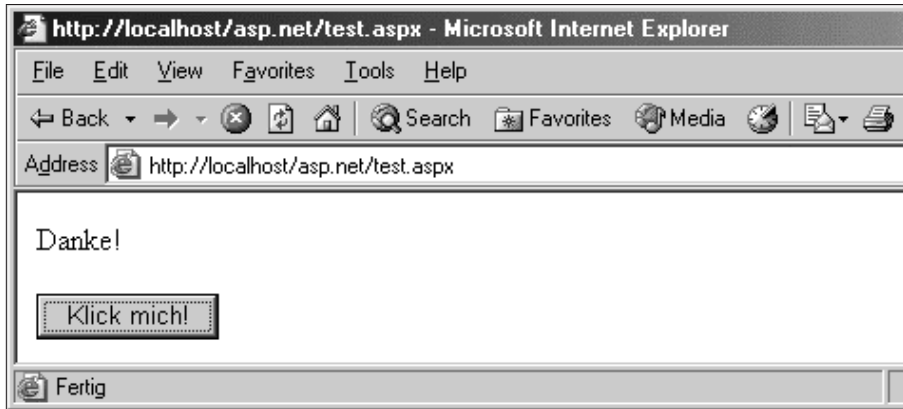


Abbildung 2.37 Die Seite leitet sich von der kompilierten Code Behind-Klasse ab.



Sollte beim Aufruf des Kompilers eine Fehlermeldung erscheinen, dass der Befehl `csc` nicht gefunden wurde, so befindet sich das Programm nicht im Pfad. In diesem Fall müssen Sie die Umgebungsvariable `Path` manuell erweitern. Öffnen Sie dazu in der Systemsteuerung den Dialog „System“. Wechseln Sie auf die Lasche „Erweitert“, und klicken Sie den Button „Umgebungsvariablen“ an. Aus der oberen Liste wählen Sie den genannten Eintrag und bearbeiten diesen über den gleichnamigen Button. Fügen Sie getrennt mit einem Semikolon den auf Ihr System angepassten Pfad hinzu:

```
<windir>\Microsoft.NET\Framework\<version>\
```

Anschließend können Sie die Eingabeaufforderung beenden und neu starten. Das Kommandozeilenprogramm sollte nun zur Verfügung stehen. Die Abbildung zeigt noch einmal die beschriebenen Dialoge.

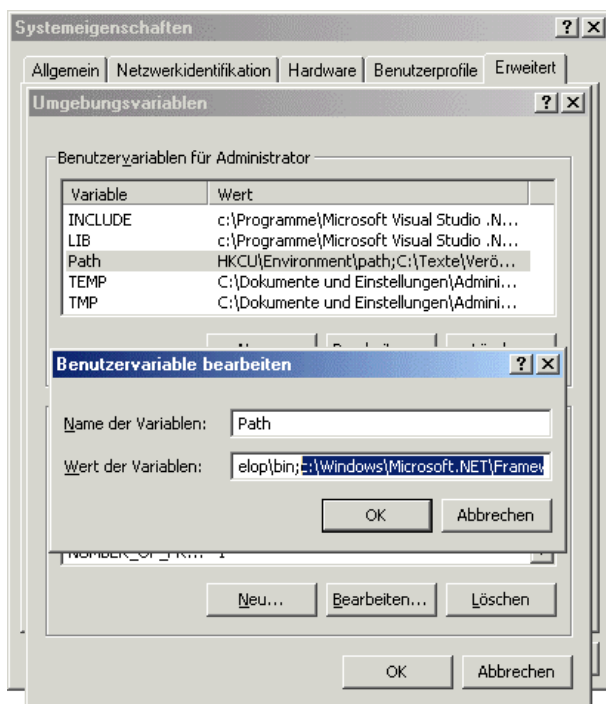


Abbildung 2.38 Die Umgebungsvariable Path muss erweitert werden.

Kompilieren mehrerer Quellcode-Dateien

Sofern Sie über mehrere Quellcode-Dateien verfügen, müssen Sie diese nicht einzeln kompilieren, sondern können sie in eine einzelne, handliche DLL verpacken. Geben Sie die Dateien hierzu einfach der Reihe nach an:

```
csc /t:library test.aspx.cs test2.aspx.cs test3.aspx.cs ...
```

Alternativ können Sie auch mit Platzhaltern arbeiten. Möchten Sie beispielsweise alle C#-Dateien im aktuellen Verzeichnis kompilieren, verwenden Sie folgenden Aufruf:

```
csc /t:library test.aspx.cs *.aspx.cs
```

Sollten auch Dateien in Unterverzeichnissen rekursiv einbezogen werden, geben Sie folgende Zeile an:

```
csc /t:library test.aspx.cs /resurse:*.aspx.cs
```

Der Name der erzeugten DLL ergibt sich in jedem der vorgestellten Fälle aus dem Namen der ersten Quellcode-Datei. Möchten Sie einen bestimmten Namen explizit vorgeben, verwenden Sie den out-Parameter:

```
csc /t:library test.aspx.cs /out:mydll.dll /resource:*.aspx.cs
```

Weitere Kompileroptionen

Die beiden Kommandozeilencompiler `csc.exe` und `vbc.exe` kennen zahlreiche Optionen. Eine Kurzübersicht können Sie sich über den Parameter `/?` anzeigen lassen.

```
csc /?
```

Selbstverständlich existiert auch eine umfangreiche Beschreibung innerhalb der .NET Framework SDK-Dokumentation. Nachfolgend möchte Ihnen nur noch zwei wichtige Parameter vorstellen.

Oftmals werden zusätzliche DLLs und Komponenten innerhalb einer Web-Applikation genutzt. Sofern Sie diese Objekte im Global Assembly Cache oder im bin-Verzeichnis abgelegt haben, können Sie nach Einbindung des entsprechenden Namespaces direkt innerhalb Ihrer Seiten darauf zugreifen. Anders sieht es aus, wenn Sie eine Code Behind-Datei kompilieren. Hier müssen Sie die externe Referenz explizit angeben. Dies geschieht über den Parameter `/r` wie Reference, mit Hilfe dessen die gewünschte DLL inklusive Dateiendung angegeben wird. Mehrere DLLs werden mit Kommata getrennt übergeben:

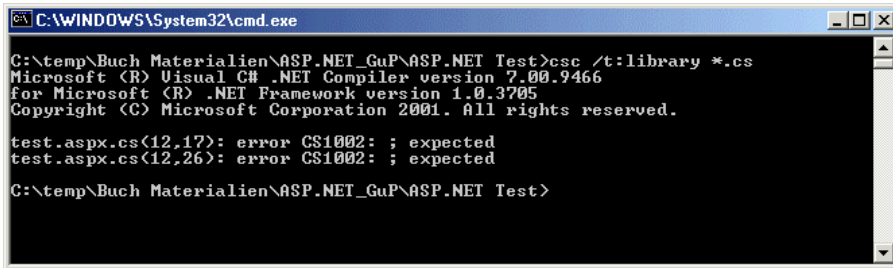
```
csc /t:library /r:System.dll,System.Web.dll *.cs
```

Normalerweise wird eine DLL im Release-Modus kompiliert. Während der Entwicklungszeit ist es jedoch durchaus hilfreich, Debug-Ausgaben und fehlerhafte Quellcode-Zeilen direkt angezeigt zu bekommen. Wollen Sie den Debug-Modus aktivieren, so verwenden Sie beim Aufruf den gleichnamigen Parameter:

```
csc /t:library /debug *.cs
```

Was tun bei Kompilierungsfehlern?

Zum Glück ist der Kompiler nicht ganz unintelligent und erkennt viele Fehler, bevor diese im Realbetrieb zu Problemen führen. Beim automatischen Kompilieren von ASP.NET-Seiten werden Ihnen die Fehler direkt im Browser angezeigt. Bei den Kommandozeilenprogrammen erfolgen ein wenig kryptischere Ausgaben in der Konsole. Die Abbildung zeigt ein Beispiel dafür.

A screenshot of a Windows command prompt window titled "C:\WINDOWS\System32\cmd.exe". The window shows the output of a C# compilation command. The text displayed is:

```
C:\temp\Buch Materialien\ASP.NET_GuP\ASP.NET Test>csc /t:library *.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

test.aspx.cs(12,17): error CS1002: ; expected
test.aspx.cs(12,26): error CS1002: ; expected

C:\temp\Buch Materialien\ASP.NET_GuP\ASP.NET Test>
```

Abbildung 2.39 Der Kompiler meldet einen Fehler.

Die Ausgabe liefert Ihnen Informationen, wo welcher Fehler auftritt. Dabei werden zunächst der Name der Quellcode-Datei und dahinter in Klammern die Zeile und die Spalte des Fehlers angegeben. Während die Spalte meist wenig aufschlussreich ist, liefert die Zeile wertvolle Informationen und führt zusammen mit der angegebenen Fehlermeldung meist zum Erfolg.

Aufruf per Batch-Datei

Ich persönlich bin vermutlich aufgrund der vielen Schreiberei kein Fan von unnötiger Tipparbeit bei Listings oder Befehlsaufrufen. Als Überbleibsel aus der DOS-Zeit erfreue auch ich mich immer wieder einmal an den Vorzügen von Batch-Dateien.

Zum Kompilieren von Code Behind-Dateien verwende ich das nachfolgend gezeigte Batch. Sie können die Datei im Entwicklungsverzeichnis ablegen. Wenn Sie im Explorer eine Datei per Drag & Drop auf das Batch ziehen, wird diese mittels `csc.exe` kompiliert. Rufen Sie das Batch hingegen ohne Parameter auf, werden alle Dateien im aktuellen Verzeichnis zu einer DLL zusammengefasst. Um eventuelle Fehler zu Gesicht zu bekommen, ist ein Aufruf von `pause` integriert. Zu guter Letzt werden alle DLLs im aktuellen Verzeichnis in das gegebenenfalls anzupassende bin-Verzeichnis verschoben.

Listing 2.55 compile.bat

```
@ECHO OFF

IF '%1' == '' GOTO ALL
csc /t:library %1
GOTO END

:ALL
csc /t:library *.cs

:END
pause
move *.dll "c:\inetpub\wwwroot\bin"
```

Sie finden das Batch selbstverständlich auf der begleitenden Buch-CD-ROM. Dort ist auch eine analoge Datei `compile_vb.bat` für Visual Basic .NET abgelegt.

2.33 ... alle Seiten automatisch von einer zentralen Code Behind-Klasse ableiten?

Im Rezept „... eine globale Seitenvorlage erstellen?“ haben Sie eine Möglichkeit kennen gelernt, eine zentrale Code Behind-Datei als Basis für ein übergreifendes Grundlayout einer Seite einzurichten. Um dieses nutzen zu können, war in jeder Seite die explizite Angabe des Inherits-Attributs der `@Page`-Direktive notwendig. Insbesondere bei umfangreicheren Seiten ist dies ein wenig lästig.

ASP.NET bietet die Möglichkeit, die standardmäßig zu verwendende Basisklasse auszutauschen. Hierzu müssen Sie diese zunächst in Form einer DLL im bin-Verzeichnis ablegen. Ich kompiliere dazu die Datei `main.cs` aus dem genannten Rezept und lege die so erzeugte `main.dll` im bin-Verzeichnis ab.

```
csc /t:library main.cs
```

Nun reicht ein einfacher Eintrag in der Konfigurationsdatei `web.config` aus, um die hinterlegte Vorlage zu verwenden. Diese trägt den Klassennamen `GlobalPageTemplate` und ist im Namespace `PAL.Projects.AspNetKochbuch` abgelegt. Beides muss dem Attribut `pageBaseType` des Konfigurationselements `pages` zugewiesen werden. Mit einem Komma separiert muss zudem der Name der entsprechenden Assembly, in diesem Fall `main`, ohne Dateiendung notiert werden.

Listing 2.56 web.config

```
<configuration>
  <system.web>

    <pages
      pageBaseType="PAL.Projects.AspNetKochbuch.GlobalPageTemplate,
        main"/>

    </system.web>
  </configuration>
```

Jede Seite innerhalb der Web-Applikation ohne explizite Angabe einer Code Behind-Klasse verwendet nun automatisch die hinterlegte Seitenvorlage. Ein Beispiel?!

Listing 2.57 Test1.aspx

```
<% @Page Language="C#" Debug="true" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
  Template.Title = "Startseite";
  Template.DataBind();
}

</script>

<h1> Willkommen auf meiner Website </h1>

<p> Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blind-
text Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext </p>
```

Die Abbildung zeigt deutlich den Erfolg, denn der eigentliche Inhalt der Seite wurde um einige zusätzliche Elemente wie Kopf, Navigation und Fuß ergänzt.

Listing 2.58 Test2.aspx

```
<% @Page Language="C#" Debug="true" Inherits="System.Web.UI.Page" %>

<h1> Willkommen auf meiner Website </h1>

<p> Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blind-
text Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext </p>
```

Analog können Sie übrigens auch eine alternative Code Behind-Vorlage für User Controls registrieren. In diesem Fall weisen Sie die von der Basis `UserControl` abgeleitete Klasse innerhalb der Konfigurationsdatei `web.config` dem Attribut `userControlBaseType` zu.

Listing 2.59 web.config

```
<configuration>
  <system.web>

    <pages
      userControlBaseType="<Klasse>, <Assembly>" />

    </system.web>
  </configuration>
```

2.34 ... die standardmäßig importierten Namespaces erweitern?

Eine Reihe von Namespaces wird automatisch importiert und steht daher beim impliziten Kompilieren einer ASP.NET-Seite zur Verfügung. Es handelt sich um folgende Namespaces:

- `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`

C#

Sollen darüber hinaus weitere Namespaces genutzt werden, so müssen Sie für jeden innerhalb jeder Seite die `@Import`-Direktive verwenden. Leider habe ich bei Verwendung von C# bis jetzt keine Möglichkeit gefunden, beispielsweise über die Konfigurationsdatei `web.config` automatisch weitere Namespaces zu importieren. Sollte Ihnen eine Möglichkeit auffallen, geben Sie mir bitte Bescheid.

Visual Basic .NET

Sofern Sie Visual Basic .NET als Programmiersprache verwenden, existiert eine Möglichkeit zur globalen Einbindung weiterer Namespaces. Diese ist ein klein wenig umständlich, funktioniert jedoch tadellos. Sie müssen die Datei `machine.config` im folgenden Verzeichnis editieren:

```
<windir>\Microsoft.NET\Framework\<version>\config\
```

Suchen Sie nach dem Abschnitt `<compilers>` und darunter nach dem Eintrag für Visual Basic .NET. Fügen Sie nun ein neues Attribut `compilerOptions` ein, und weisen Sie diesem die gewünschten Optionen zu. Um beispielsweise weitere Namespaces zu importieren, geben Sie diese hinter der Option `/imports:` an. Das Beispiel zeigt den zusätzlichen Import des Namespaces `System.Net`:

Listing 2.60 machine.config

```
...

<compilers>
  <compiler
    language="vb;vbs;visualbasic;vbscript"
    extension=".vb"
    type="Microsoft.VisualBasic.VBCodeProvider, System, .."
    compilerOptions="/imports:System.Net" />
  ...
```

Sollen mehrere Namespaces importiert werden, geben Sie diese mit Kommata separiert an. Dass dieser Import wirklich funktioniert, zeigt das folgende Listing. Die dort verwendeten Klassen stammen aus dem Namespace `System.Net`, der ansonsten explizit über die `@Import`-Direktive referenziert werden müsste.

Listing 2.61 Imports1.aspx

```
<% @Page Language="VB" Debug="true" %>

<script runat="server">

Sub Page_Load(Sender As Object, E As EventArgs)

    Dim RemoteHost AS String = Request.ServerVariables("REMOTE_HOST")
    Dim host As IPHostEntry = Dns.Resolve(RemoteHost)
    Response.Write("Ihr Host-Name: " & host.HostName.ToString())

End Sub

</script>
```

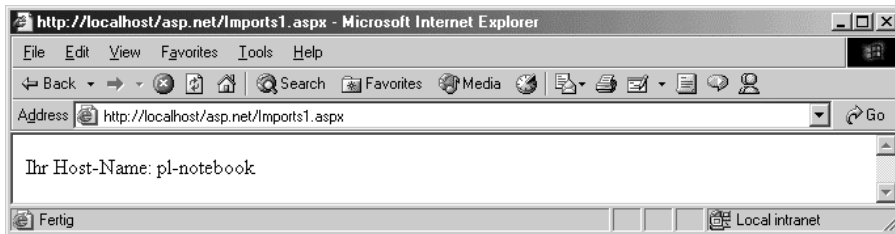


Abbildung 2.42 Der globale Import des Namespaces ist geglückt.

Einbindung eigener Klassen

Möchten Sie eigene Klassen nutzen, ohne den Namespace explizit einzubinden, notieren Sie diese einfach ohne Namespace, und legen Sie sie im bin-Verzeichnis als DLL ab. Ein Beispiel hierfür finden Sie im Rezept „... Hilfsfunktionen global hinterlegen?“ im Kapitel „Sprachelemente“.

2.35 ... die standardmäßig gesetzten Optionen beim Kompilieren verändern?

Beim Kompilieren über das Kommandozeilenprogramm `csc.exe` müssen vermutlich auch Sie immer wieder die gleichen Optionen angeben. So ist beispielsweise beim manuellen Kompilieren von Code Behind-Dateien die Angabe von `/t:library` notwendig, damit eine DLL und keine EXE erzeugt wird. Mitunter ist das ein wenig lästig, und daher gibt es eine schöne Möglichkeit, die explizite Angabe zu verhindern.

Der C#-Kompiler kann Konfigurationsdateien verarbeiten, die die Standardwerte für die angebotenen Optionen verändern. Eine globale Version dieser Datei befindet sich unter dem Namen `csc.rsp` im folgenden Verzeichnis:

```
<windir>\Microsoft.NET\Framework\<version>\
```

Der Inhalt der Datei sieht in etwa so aus:

Listing 2.62 csc.rsp

```
# This file contains command-line options that the C#
# command line compiler (CSC) will process as part
# of every compilation, unless the "/noconfig" option
# is specified.

# Reference the common Framework libraries
/r:Accessibility.dll
/r:Microsoft.Vsa.dll
/r:System.Configuration.Install.dll
/r:System.Data.dll
/r:System.Design.dll
/r:System.DirectoryServices.dll
/r:System.dll
/r:System.Drawing.Design.dll
/r:System.Drawing.dll
/r:System.EnterpriseServices.dll
/r:System.Management.dll
/r:System.Messaging.dll
/r:System.Runtime.Remoting.dll
/r:System.Runtime.Serialization.Formatters.Soap.dll
/r:System.Security.dll
/r:System.ServiceProcess.dll
/r:System.Web.dll
/r:System.Web.RegularExpressions.dll
/r:System.Web.Services.dll
/r:System.Windows.Forms.dll
/r:System.XML.dll
```

Ganz offensichtlich werden hier einige Assemblies standardmäßig importiert. Ebenfalls offensichtlich ist das Format der Konfigurationsdatei. Hier werden die Optionen so hinterlegt, wie sie auch an das Kommandozeilenprogramm übergeben werden würden.

Um beispielsweise die Angabe von `/t:library` zu sparen, können Sie der Datei diesen Aufruf einfach in einer neuen Zeile anhängen. Ich rate jedoch davon ab, denn dies würde jeglichen Kompilervorgang mittels des Programms auf dem System beeinflussen. Es empfiehlt sich hingegen die Anlage einer lokalen Version der Datei. Diese muss sich in dem Verzeichnis befinden, von dem aus der Compiler aufgerufen wird. Der Name ist auch hier `csc.rsp`:

Listing 2.63 `csc.rsp`

```
/t:library
```

Nun können Sie Ihre Code Behind-Dateien aus diesem Verzeichnis ohne explizite Angabe des „targets“ kompilieren; das Ergebnis ist nun immer eine DLL.

```
csc.exe test.cs
```

Natürlich können Sie auch beliebige weitere Optionen des Kommandozeilenprogramms über die Konfigurationsdatei vorbelegen. Das Programm arbeitet diese in der folgenden Reihenfolge durch:

1. Globale Datei `csc.rsp`
2. Lokale Datei `csc.rsp`
3. Kommandozeilenparameter

Mehrfach angegebene Optionen werden in dieser Reihenfolge überschrieben. Setzen Sie beispielsweise in der globalen Konfigurationsdatei eine Option, können Sie diese beim Aufruf über die Kommandozeilenparameter trotzdem verändern. Sollen die beiden Konfigurationsdateien überhaupt nicht ausgewertet werden, so fügen Sie dem Aufruf des Programms den Parameter `/noconfig` an:

```
csc /noconfig ...
```

2.36 ... eine Assembly aus dem Global Assembly Cache einbinden?

Externe Assemblies werden Sie in aller Regel im `bin`-Verzeichnis Ihrer Web-Applikation ablegen. Alle DLLs aus diesem Verzeichnis werden automatisch geladen und können direkt verwendet werden. Anders sieht es bei den global hinterleg-

ten DLLs im Global Assembly Cache (GAC) aus. Diese müssen explizit eingebunden werden. Bei den gebräuchlichsten ist dies bereits durch die Standardeinstellung der globalen Konfigurationsdatei `machine.config` gegeben. Vom Standard abweichende DLLs müssen jedoch manuell in der `web.config` referenziert werden.

Das folgende Beispiel zeigt die Einbindung der Assembly `System.Design.dll` aus dem GAC.

Listing 2.64 `web.config`

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="System.Design,
          Version=1.0.3300.0,
          Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```



Wenn Sie eine Assembly auf diese Weise einbinden, erhalten Sie unter Umständen eine Fehlermeldung, dass die DLL nicht gefunden werden konnte. In diesem Fall haben Sie die Datei vermutlich wie oben gezeigt eingebunden. Dies führt jedoch zu Problemen, da ein Umbruch enthalten ist. Seien Sie nett zu .NET, und notieren Sie den Inhalt des `add`-Elements innerhalb einer Zeile:

```
<add assembly="System.Design, Version=1.0.3300.0, Culture=neutral, Pub
licKeyToken=b03f5f7f11d50a3a"/>
```

Ist die Assembly erfolgreich eingebunden, kann sie wie andere auch verwendet werden. Die Frage, die sich Ihnen nun vermutlich stellt: Wie komme ich an die benötigten Informationen zur Referenzierung der DLL? Das ist ganz einfach. Öffnen Sie im Explorer das folgende Verzeichnis:

```
<windir>\Assembly\
```

Sie erhalten eine Übersicht aller im Global Assembly Cache registrierten DLLs. Aus dem Kontextmenü eines Eintrags können Sie das Eigenschaftsfenster öffnen. Hier finden Sie alle benötigten Informationen wie Namen, die Sprache, die Version und den öffentlichen Schlüssel. Kopieren Sie diese Informationen nun einfach über die Zwischenablage in die Konfigurationsdatei.



Abbildung 2.43 Die Eigenschaften der Assembly System.Design



Verwenden Sie als Versionsangabe bitte unbedingt immer die auf der *ersten* Registerlasche genannte Nummer und *nicht* die auf der *zweiten* Lasche „Version“ gemachte Angabe.

Sofern sinnvoll, können Sie die Assembly auch global allen Web-Applikationen zur Verfügung stellen. Nehmen Sie die beschriebenen Änderungen bitte in diesem Fall an der globalen Konfigurationsdatei `machine.config` im folgenden Verzeichnis vor:

```
<windir>\Microsoft.NET\Framework\<version>\config\
```

2.37 ... eine DLL im Global Assembly Cache ablegen?

Im Rezept „... eine Assembly aus dem Global Assembly Cache einbinden?“ haben Sie gelesen, wie Sie eine im Global Assembly Cache (GAC) bestehende Assembly in Ihre Web-Applikation einbinden können. Alternativ ist auch eine Einbindung auf

globaler Ebene über die Konfigurationsdatei `machine.config` möglich. Die Assembly steht dann automatisch in jeder Web-Applikation zur Verfügung, muss aber dennoch nur einmal auf dem System abgelegt werden.

Damit Assemblies auf diese Weise global zur Verfügung stehen, müssen sie zunächst einmal im GAC abgelegt werden. Sie können die gewünschte DLL dazu einfach per Drag & Drop in das GAC-Verzeichnis ziehen. Sie finden dieses im folgenden Ordner:

```
<windir>\Assembly\
```

Alternativ können Sie das Kommandozeilenprogramm `gacutil.exe` verwenden. Nach dem Parameter `/i` folgt der Dateiname der abzulegenden DLL:

```
gacutil /i MeineAssembly.DLL
```

Möglicherweise erhalten Sie beim Ablegen der Komponente eine Fehlermeldung mit dem Hinweis, dasss die Datei nicht „strongly named“ sei. Dies ist eine Voraussetzung für die Aufnahme in den GAC, doch was bedeutet das?

Damit eine Assembly im GAC abgelegt werden kann, muss diese eindeutig identifizierbar sein. Reichte früher der Name aus, ist jetzt zusätzlich die Sprache, die Version und ein eindeutiger Schlüssel notwendig. Auf diese Weise wird der wohl bekannten DLL-Hölle vorgebeugt, und es ist beispielsweise möglich, zwei unterschiedliche Versionen von der gleichen Assembly im GAC abzulegen und parallel zu verwenden.

Die eindeutige Kennzeichnung über Name, Sprache, Version und Schlüssel wird „starker Name“ oder auch „strong name“ genannt, da dieser eine sehr sichere Identifizierung ermöglicht. Ist eine DLL nicht strong named, kann diese nicht im GAC abgelegt werden. Stammt die Assembly nicht von Ihnen, haben Sie leider Pech gehabt. Handelt es sich hingegen um eine eigene DLL, so können Sie diese in wenigen Schritten mit einem starken Namen ausrüsten.

Einen starken Namen vergeben

Die Signierung mit einem starken Namen ist recht einfach durchzuführen. Zunächst möchte ich Ihnen das Opfer vorstellen. Es handelt sich um die Klasse `Helper`, die ich dem Rezept „... Hilfsfunktionen global hinterlegen?“ aus dem Kapitel „Sprachelement“ entwendet habe. Die einzige statische Methode `GetGreeting` liefert eine tageszeitabhängige Begrüßung.

Listing 2.65 HelperMethods1.cs

```
using System;
using System.Web;

namespace PAL.Projects.AspNetKochbuch
{
    public class Helper
    {
        public static string GetGreeting()
        {
            int hour = DateTime.Now.Hour;

            if(hour <= 10) return("Guten Morgen");
            if(hour <= 17) return("Guten Tag");
            return("Guten Abend");
        }
    }
}
```

Im ersten Schritt muss eine Schlüsseldatei erzeugt werden. Diese enthält einen privaten und einen öffentlichen Schlüssel, der später für die Einbindung benötigt wird. Die Arbeit übernimmt das Kommandozeilenprogramm `sn.exe`. Nach dem Parameter `-k` wird der Name der zu erzeugenden Schlüsseldatei angegeben, zum Beispiel so:

```
sn -k helpermethods1.snk
```

Die Schlüsseldatei ist zunächst unabhängig von der DLL und muss dieser nun zugeordnet werden. Hierzu importieren Sie den Namespace `System.Reflection`. Anschließend verwenden Sie die zwei Assembly-weiten Attribute `AssemblyVersion` und `AssemblyKeyFile`. Während mit Hilfe des ersten Attributs die aktuelle Version der Assembly angegeben wird, nimmt das zweite den vollen Dateinamen der erstellten Schlüsseldatei entgegen. Die Quellcode-Datei sieht anschließend so aus:

Listing 2.66 HelperMethods1.cs

```
using System;
using System.Web;
using System.Reflection;

[assembly: AssemblyVersion("1.0.1.*")]
[assembly: AssemblyKeyFile(@"<pfad>\helpermethods1.snk")]

namespace PAL.Projects.AspNetKochbuch
{
    ...
    ...
}
```

Nun können Sie die Datei wie gewohnt mittels `csc.exe` kompilieren. Die so erzeugte DLL ist strongly named und kann im GAC abgelegt werden. Dies geschieht – wie beschrieben – entweder per Drag & Drop im Explorer und mittels `gacutil.exe`. Die Abbildung zeigt die erzeugte DLL im GAC.

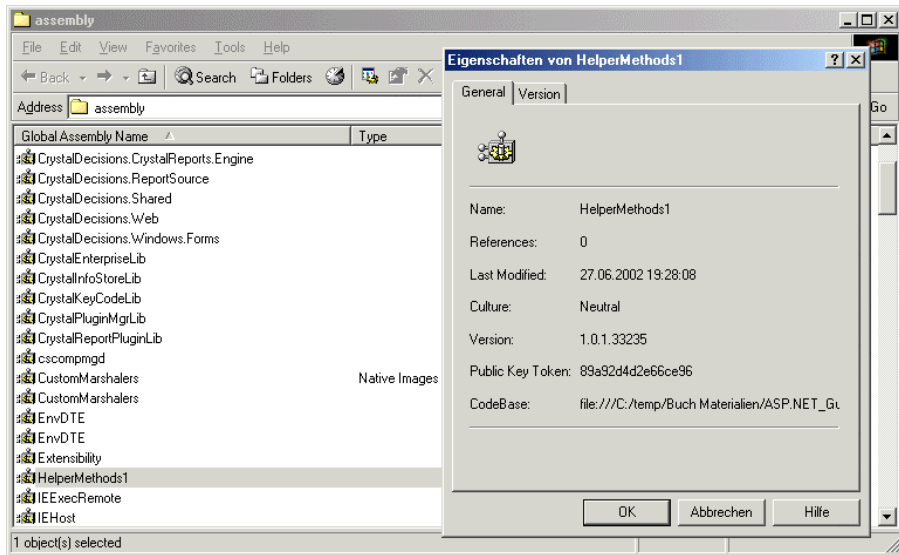


Abbildung 2.44 Die Assembly wurde erfolgreich im GAC abgelegt.

Ist die Assembly erfolgreich abgelegt, können Sie diese – wie im oben genannten Rezept beschrieben – über die Konfigurationsdatei `web.config` referenzieren. Für den vorliegenden Fall sieht die notwendige Konfiguration wie folgt aus (die Umbrüche dienen der Übersicht und dürfen nicht verwendet werden):

Listing 2.67 `web.config`

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="HelperMethods1,
          Version=1.0.1.33235,
          Culture=neutral,
          PublicKeyToken=89a92d4d2e66ce96" />
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Ist die Assembly referenziert, kann sie innerhalb jeder Seite der Web-Applikation genutzt werden.

Listing 2.68 Greeting1.aspx

```
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write(Helper.GetGreeting());
}

</script>
```

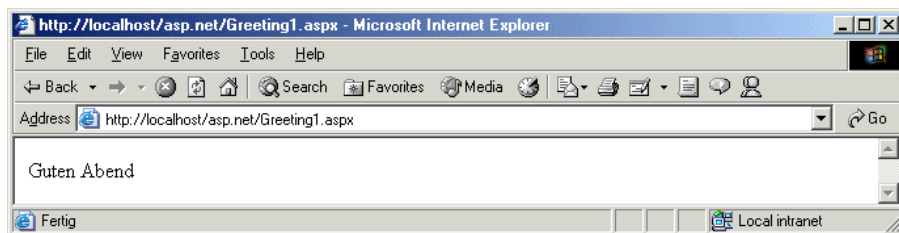


Abbildung 2.45 Die Assembly wird aus dem Global Assembly Cache aufgerufen.



Sinn und Zweck des Global Assembly Cache ist die Verhinderung einer neuen DLL-Hölle. Statt derer ist es nun möglich, mehrere Versionen einer DLL parallel zu verwenden. Damit dieses System funktioniert, ist es unbedingt erforderlich, dass Sie vor dem Kompilieren einer neuen Version – und seien die Änderungen noch so minimal – die Version hochzählen und eine neue Schlüsseldatei mittels `sn.exe` erzeugen. Nur so ist eine Differenzierung der verschiedenen Versionen möglich.

Assembly aus dem GAC löschen

Selbstverständlich können Sie eine DLL auch wieder aus dem GAC entfernen. Drücken Sie dazu die Entfernen-Taste im Explorer, oder rufen Sie das Kommandozeilenprogramm `gacutil.exe` wie folgt auf:

```
gacutil /u helpermethods1
```

2.38 ... den gesamten Query-String abfragen?

Der Query-String stellt eine Art Dictionary dar, bei dem Werte in Verbindung mit einem eindeutigen Schlüssel hinterlegt werden:

```
meineseite.aspx?var1=wert1&var2=wert2
```

Um auf die einzelnen Elemente zuzugreifen, verwenden Sie die `NameValueCollection`, die von der Eigenschaft `Request.QueryString` geliefert wird. Doch was, wenn Sie auf den gesamten Query-String zugreifen wollen, beispielsweise weil Sie Daten ohne Schlüsselzuordnung übergeben haben, wie die folgende Zeile zeigt ...

```
meineseite.aspx?diesistderquerystring
```

In diesem Fall können Sie die übergebenen Daten über die Server-Variable „`QUERY_STRING`“ abfragen. Das Listing zeigt, wie's geht.

Listing 2.69 QueryString1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string qs = Request.ServerVariables["QUERY_STRING"];
    Response.Write(qs);
}

</script>
```

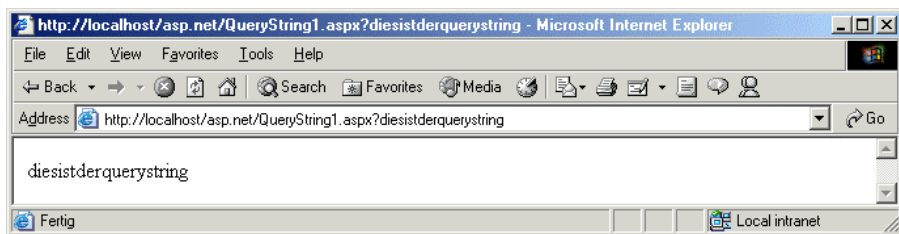


Abbildung 2.46 Auch der gesamte Query-String lässt sich abfragen.

3

Sprachelemente



Wie kann ich ...

3 Sprachelemente

Die Programmiersprache ist zwar nicht das Ziel der Entwicklung, aber sie ist dessen Weg. In diesem Kapitel erfahren Sie einige Details, die Ihnen den Umgang mit C# und Visual Basic .NET angenehm erleichtern werden.

3.1 ... eine sichere Typenkonvertierung durchführen?

Die explizite Typenkonvertierung ist sehr schnell, bietet aber keinerlei Überprüfungsmöglichkeiten, ob die Konvertierung auch tatsächlich möglich ist. C# bietet zwei einfache Möglichkeiten, eine sichere Typenkonvertierung durchführen. Es handelt sich um die Operatoren `is` und `as`. Während `is` lediglich eine Überprüfung durchführt und deren Erfolg mit Hilfe eines booleschen Wertes zurückliefert, liefert `as` direkt den konvertierten Inhalt beziehungsweise `null`, wenn die Konvertierung nicht möglich war.

Listing 3.1 `is_as1.aspx`

```
void Page_Load(object sender, EventArgs e)
{
    object obj = new Hashtable();

    if(obj is Hashtable)
    {
        Hashtable ht1 = (Hashtable) obj;
        ht1.Add("hallo", "welt");
    }

    Hashtable ht2 = (obj as Hashtable);
    if(ht2 != null)
        ht2.Add("welt", "hallo");

    foreach(DictionaryEntry de in (Hashtable) obj)
        Response.Write(de.Value.ToString() + "<br>");
}
```

3.2 ... den Typ einer Variablen ermitteln?

Die Operatoren `is` und `as` lassen sich verwenden, um eine sichere Typenkonvertierung durchzuführen und dabei zu überprüfen, ob ein Objekt einem bestimmten Datentyp entspricht. Mehr hierzu erfahren Sie im Rezept „... eine sichere Typenkonvertierung durchführen?“ in diesem Kapitel.

Erhalten Sie jedoch ein Ihnen unbekanntes Objekt, so möchten Sie eventuell den Namen des zugrunde liegenden Datentyps ermitteln. Hierzu können Sie die von `object` geerbte Methode `GetType` verwenden. Diese liefert eine Instanz der Methode `Type`, über deren Eigenschaft `Name` der Name des Typs abgefragt werden kann.

Listing 3.2 TypeName1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Type t = Request.GetType();
    string name = t.Name;
    Response.Write("Request-Klasse: " + name);
}

</script>
```

Das Listing zeigt die Abfrage des Datentyps des `Request`-Objekts. Die Abbildung zeigt, dass die Klasse den Namen `HttpRequest` trägt. Möchten Sie zusätzlich auch den vollständigen Namespace erhalten, verwenden Sie die alternative Eigenschaft `FullName`.

Listing 3.3 TypeName2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Type t = Request.GetType();
    string name = t.FullName;
    Response.Write("Request-Klasse: " + name);
}

</script>
```

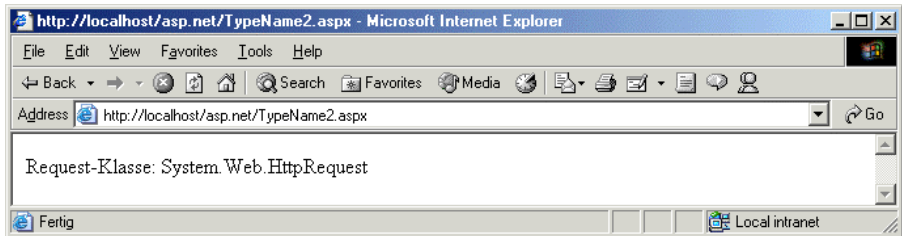


Abbildung 3.1 Der volle Name der Klasse inklusive dem Namespace

3.3 ... abfragen, ob ein Wert in einem Array vorhanden ist?

Oftmals soll überprüft werden, ob ein bestimmter Wert (bereits) in einem Array vorhanden ist. Sofern Sie ein dynamisches Array, also eine Instanz der Klasse `ArrayList` verwenden, ist dies sehr einfach abzufragen. Hier existiert eine Methode `Contains`, der das zu prüfende Element übergeben wird. Der boolesche Rückgabewert informiert, ob das Element enthalten ist oder nicht. Diese Methode wird durch die Schnittstelle `ICollection` vorgegeben. Alle Klassen, die diese Schnittstelle unterstützen, bieten somit auch eine Methode `Contains` an.

ArrayList

Im folgenden Beispiel wird eine neue `ArrayList` instanziiert. Es werden einige Buchstaben (Datentyp `char`) angefügt und anschließend überprüft, ob ein bestimmtes Zeichen enthalten ist. Wie nicht anders zu erwarten ist, liefert die Methode `Contains` den Rückgabewert `true`.

Listing 3.4 Contains1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    ArrayList al = new ArrayList();
    al.Add('a');
    al.Add('b');
    al.Add('c');

    if(al.Contains('b'))
    {
        Response.Write("\"b\" ist enthalten.");
    }
}
```

```
else
{
    Response.Write("\"b\" ist nicht enthalten.");
}
}

</script>
```

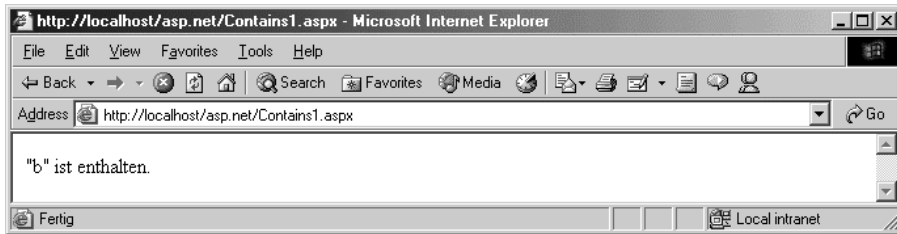


Abbildung 3.2 Der Buchstabe „b“ ist in der ArrayList enthalten.

Array

Die Kurzschreibweise eines Arrays darf nicht über deren Ursprung täuschen. Letztlich handelt es sich lediglich um eine Instanz der Klasse `Array`. Schaut man sich deren Mitglieder an, so entdeckt man leider keine analoge Methode `Contains`. Das verwundert, denn schließlich implementiert die Klasse die Schnittstelle `ICollection`, und diese enthält wie oben beschrieben eine derartige Methode. Des Rätsels Lösung ist schnell gefunden, bei `Contains` handelt es sich in diesem Fall um eine explizite Schnittstellenimplementierung. Um auf die Methode zuzugreifen, müssen Sie zunächst eine Typenkonvertierung zu `ICollection` durchführen.

Das Listing zeigt den Zugriff auf die scheinbar nicht vorhandene Methode, hier anhand eines `char`-Arrays.

Listing 3.5 Contains2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    char[] chars = {'a', 'b', 'c'};

    ICollection list = (ICollection) chars;

    if(list.Contains('b'))
        Response.Write("\"b\" ist enthalten.");
}
```

```
else
    Response.Write("\"b\" ist nicht enthalten.");
}

</script>
```

Sofern Sie die Typenkonvertierung und den damit verbundenen Aufwand scheuen, so können Sie alternativ auch die statische Methode `IndexOf` verwenden. Diese liefert den ersten Index eines übergebenen Elements in einem Array. Ist dieses gar nicht vorhanden, so wird `-1` zurückgeliefert.

Listing 3.6 Contains3.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    char[] chars = {'a', 'b', 'c'};

    if(Array.IndexOf(chars, 'b')>-1)
        Response.Write("\"b\" ist enthalten.");
    else
        Response.Write("\"b\" ist nicht enthalten.");
}

</script>
```

`IList` ist nicht umsonst eine beliebte und wichtige Schnittstelle des .NET Frameworks. Viele Klassen unterstützen die darüber angebotenen Methoden und Eigenschaften. Eine (unvollständige) Liste dieser finden Sie in der SDK-Dokumentation.



Auch die Schnittstelle `IDictionary` bietet eine Methode `Contains` an. Demzufolge implementieren auch Dictionaries eine entsprechende Möglichkeit. Bekanntester Vertreter der Schnittstelle ist die Klasse `HashTable`.

3.4 ... ein beliebiges Array sortieren?

Die Klasse `Array` besitzt eine statische Methode `Sort`, mit Hilfe derer Sie ein beliebiges Array sortieren können.

Listing 3.7 Array_Sort1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
```

```
{  
    string[] names = {"Elvis", "Aaron", "Lisa", "Marie"};  
    Array.Sort(names);  
  
    foreach(string name in names)  
        Response.Write(name + "<br>");  
}  
  
</script>
```

Voraussetzung für die Sortierung ist, dass die enthaltenen Daten Standardtypen sind oder sie die Schnittstelle `IComparable` unterstützen. Ist dies nicht der Fall, so kann die Methode die Objekte nicht miteinander vergleichen und somit nicht sortieren. Möchten Sie ein Array mit eigenen Klassen sortieren, so sollten diese die genannte Schnittstelle unterstützen.

Das Listing zeigt dies anhand der Klasse `Point`. Die Schnittstelle `IComparable` enthält eine einzige Methode `CompareTo`. Dieser wird das mit der aktuellen Instanz zu vergleichende Objekt übergeben. Der numerische Rückgabewert entscheidet, welches Objekt größer ist.

Listing 3.8 Array_Sort2.aspx

```
<script runat="server">  
  
void Page_Load(object sender, EventArgs e)  
{  
    Point[] points = {new Point(25,255),  
                      new Point(25,45),  
                      new Point(145,9),  
                      new Point(1,2)};  
  
    Array.Sort(points);  
  
    foreach(Point point in points)  
        Response.Write(point.ToString() + "<br>");  
}  
  
class Point : IComparable  
{  
    public int X;  
    public int Y;  
  
    public Point(int x, int y)  
    {  
        X = x;  
        Y = y;  
    }  
}
```

```

public int CompareTo(object obj)
{
    Point p2 = (Point) obj;
    int p1val = (this.X * this.Y);
    int p2val = (p2.X * p2.Y);

    if(p1val < p2val)
        return(-1);
    else if(p1val > p2val)
        return(1);
    else
        return(0);
}

public override string ToString()
{
    return(string.Format("x: {0}, y: {1}", this.X, this.Y));
}
}
</script>

```

Die Tabelle zeigt die möglichen Rückgabewerte der Methode `CompareTo`. Der Wert entscheidet, wie der Vergleich der aktuellen Instanz mit der übergebenen ausfällt.

Tabelle 3.1 Die Rückgabewerte der Methode `Comparable.CompareTo`

Rückgabewert	Bedeutung
Kleiner 0	Aktuelle Instanz ist kleiner als die übergebene
Gleich 0	Aktuelle Instanz ist identisch mit der übergebenen
Größer 0	Aktuelle Instanz ist größer als die übergebene

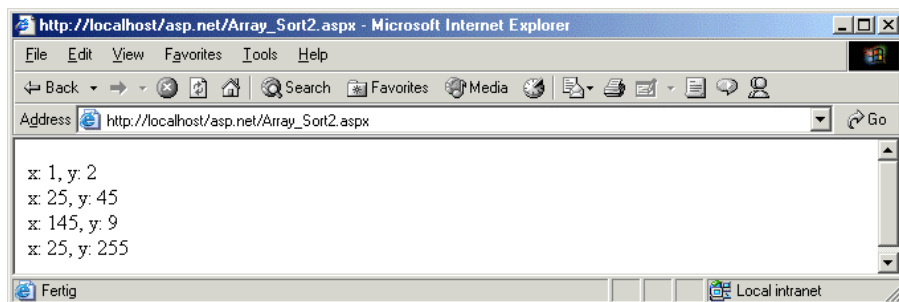


Abbildung 3.3 Die Punkte wurden korrekt sortiert.

Sofern Sie beispielsweise Objekte sortieren wollen, die die Schnittstelle `IComparable` nicht unterstützen, so können Sie einen externen Comparer erstellen. Die Schnittstelle `IComparer` implementiert eine einzige Methode `Compare`, der zwei zu vergleichende Objekte übergeben werden. Der Aufbau entspricht dem vorangegangenen Beispiel. Der individuelle Comparer wird einer Überladung der Methode `Sort` der Klasse `Array` übergeben.

Listing 3.9 `Array_Sort3.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Point[] points = {new Point(25,255),
                      new Point(25,45),
                      new Point(145,9),
                      new Point(1,2)};

    Array.Sort(points, new PointComparer());

    foreach(Point point in points)
        Response.Write(point.ToString() + "<br>");
}

class PointComparer : IComparer
{
    public int Compare(object obj1, object obj2)
    {
        Point p1 = (Point) obj1;
        Point p2 = (Point) obj2;
        int p1val = (p1.X * p1.Y);
        int p2val = (p2.X * p2.Y);

        if(p1val < p2val)
            return(-1);
        else if(p1val > p2val)
            return(1);
        else
            return(0);
    }
}

class Point
{
    public int X;
    public int Y;

    public Point(int x, int y)
```



```
{
    X = x;
    Y = y;
}

public override string ToString()
{
    return(string.Format("x: {0}, y: {1}", this.X, this.Y));
}
}

</script>
```



Eine Collection vom Typ `ArrayList` lässt sich auf eine ähnliche Weise sortieren. Auch hier wird eine Methode `Sort` angeboten, die entsprechende Überladungen anbietet. Es handelt sich jedoch um eine Instanzmethode.

3.5 ... ein Array inline erstellen?

Viele Methoden erwarten als Parameter ein wie auch immer geartetes Array. Gerade wenn die einzelnen Daten statisch oder als Variablen bereits zur Verfügung stehen und eigentlich nur übergeben werden sollen, ist es unkomfortabel, zunächst ein neues Array zu deklarieren. Das Listing zeigt dies anhand der Methode `WriteNumbers`, die ein übergebenes `int`-Array im Browser ausgeben soll.

Listing 3.10 `InlineArray1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    int[] numbers = {1, 2, 3, 4, 5};
    WriteNumbers("Zahlen", numbers);
}

void WriteNumbers(string text, int[] values)
{
    Response.Write(text + ": ");
    foreach(int i in values)
        Response.Write(i.ToString() + " ");

    Response.Write("<br>");
}

</script>
```

Die Abbildung zeigt, dass die explizite Anlage und Übergabe des Array – selbstverständlich – funktioniert. Dennoch ist dieses Vorgehen überflüssig lang. Schöner wäre es, wenn das Array direkt inline beim Aufruf der Methode angelegt werden könnte. Das ist an sich auch kein Problem, man muss nur die richtige Syntax kennen. Hier kommt sie:

Listing 3.11 InlineArray2.aspx

```
...  
void Page_Load(object sender, EventArgs e)  
{  
    WriteNumbers("Zahlen", new int[] {1, 2, 3, 4, 5});  
}  
...
```

Direkt hinter `new` und der Array-Definition werden die einzelnen Elemente in geschweiften Klammern notiert. Analog funktioniert dies für beliebige Datentypen.

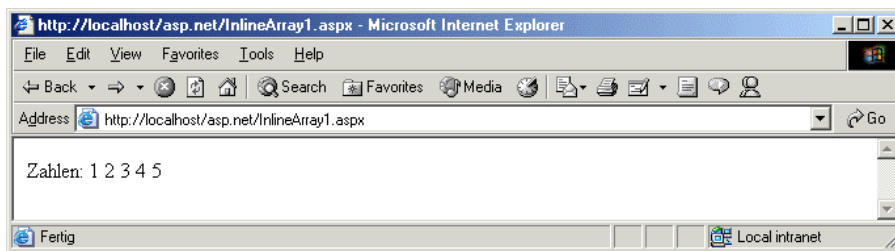


Abbildung 3.4 Die Werte des `int`-Arrays werden im Browser ausgegeben.



Sofern Sie wie im gezeigten Beispiel selbst die Kontrolle über die Zielmethode haben, kann es unter Umständen einfacher sein, das `params`-Schlüsselwort zu verwenden. Worum es sich dabei handelt, erfahren Sie im Rezept „... einer Methode eine unbekannte Anzahl von Parametern übergeben?“ in diesem Kapitel.

3.6 ... eine `if`-Abfrage inline durchführen?

`if`-Abfragen sind ein integraler Bestandteil der Entwicklung. Mitunter wäre es jedoch wünschenswert, die Notation ein wenig abzukürzen. Dies ist insbesondere der Fall, wenn einer Variablen bedingt entweder der eine oder der andere Wert zugewiesen werden soll. In `C#` benötigen Sie hierzu mindestens drei Befehlszeilen, je nach Schönschreibung des Quellcodes sogar noch mehr.

Das Listing zeigt die reguläre Verwendung von `if`. Abhängig von einer Benutzereingabe soll der Variablen `result` entweder der eine oder der andere Wert zugewiesen werden.

Listing 3.12 Inlinelf1.aspx

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    string result;
    if(tb.Text == "geheim") result = "OK";
    else result = "Nicht OK";

    Response.Write("Ihre Eingabe ist " + result);
}

</script>

<form runat="server">

    <p> Geben Sie "geheim" ein:
        <ASP:TextBox id="tb" runat="server"/>
    </p>

    <p>
        <ASP:Button runat="server" text="OK" onclick="bt_click"/>
    </p>

</form>
```

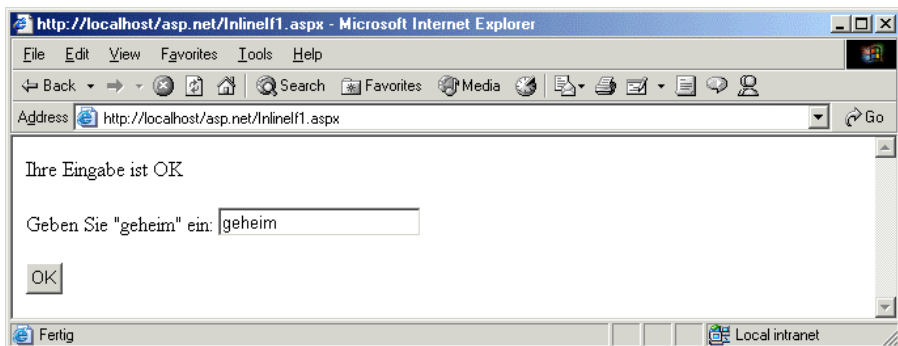


Abbildung 3.5 Die Bedingung hat eine korrekte Benutzereingabe festgestellt.

Mit Hilfe der von C# unterstützten inline if-Abfrage können Sie den Schreibumfang deutlich reduzieren, ohne an Übersichtlichkeit zu verlieren. Die Bedingung wird dabei innerhalb einer Zeile notiert. In Klammern wird zunächst der boolesche Ausdruck, ein Fragezeichen, der zurückzugebende Wert im Erfolgsfall, ein Doppelpunkt und der alternative Wert angegeben. Dies sieht schematisch so aus:

```
Variable = (Ausdruck ? Wahr : Falsch);
```

Natürlich lässt sich dieses Schema sehr einfach auf das gezeigte Beispiel anwenden und spart so deutlich an Tipparbeit. Oft wird die Inline-Notation übrigens auch Bedingungsoperator genannt.

Listing 3.13 Inlinelf2.aspx

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    string result = (tb.Text == "geheim" ? "OK" : "Nicht OK");
    Response.Write("Ihre Eingabe ist " + result);
}

</script>

...
```

Visual Basic .NET kennt diese Notation nicht. Dafür existiert jedoch eine alternative Funktion mit dem Namen `IIf`. Das dritte Listing zeigt die Umsetzung des Beispiels auf die zweite große .NET-Sprache.

Listing 3.14 Inlinelf3.aspx

```
<script runat="server">

Sub bt_click(Sender as Object, E as EventArgs)
    Dim result As String = IIf(tb.Text = "geheim", "OK", "Nicht OK")
    Response.Write("Ihre Eingabe ist " & result)
End Sub

</script>
```

3.7 ... einer Methode eine unbekannte Anzahl von Parametern übergeben?

In der Regel steht die Anzahl der Parameter einer Methode bereits zur Deklaration fest. Um eine unterschiedlich lange Parametersequenz zu unterstützen, können Sie verschiedene Überladungen einer Methode anbieten.

Visual C# .NET

Es besteht die Möglichkeit, einer Methode eine beliebige Anzahl von Parametern zu übergeben. Hierzu wird das Schlüsselwort `params` in der Deklaration der Methode verwendet. Dahinter wird ein eindimensionales Array notiert, das die Werte aufnimmt. Das Listing zeigt den Einsatz.

Listing 3.15 params1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    WriteNumbers("Zahlen", 1, 2, 3, 4, 5);
}

void WriteNumbers(string text, params int[] values)
{
    Response.Write(text + ": ");

    foreach(int i in values)
        Response.Write(i.ToString() + " ");

    Response.Write("<br>");
}

</script>
```

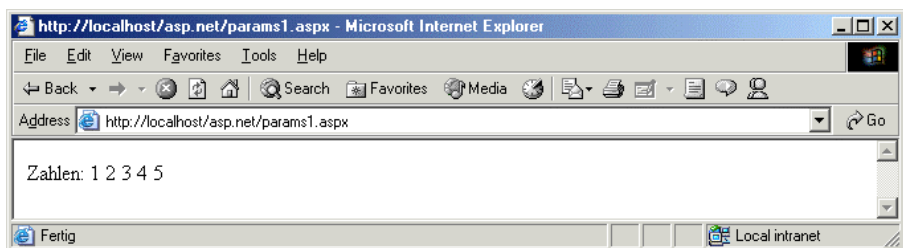


Abbildung 3.6 Die Zahlen wurden als Parameter an die Methode `WriteNumbers` übergeben.

Im vorliegenden Beispiel kann eine beliebige Anzahl von int-Werten übergeben werden. Selbstverständlich können Sie auch ein object-Array angeben, so dass beliebige Objekte aufgenommen werden können.

Listing 3.16 params2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    WriteObjects("Objekte", Request, Response, Session, Cache);
}

void WriteObjects(string text, params object[] values)
{
    Response.Write(text + "<br>");

    foreach(object obj in values)
        Response.Write("&bull; " + obj.ToString() + "<br>");
}

</script>
```

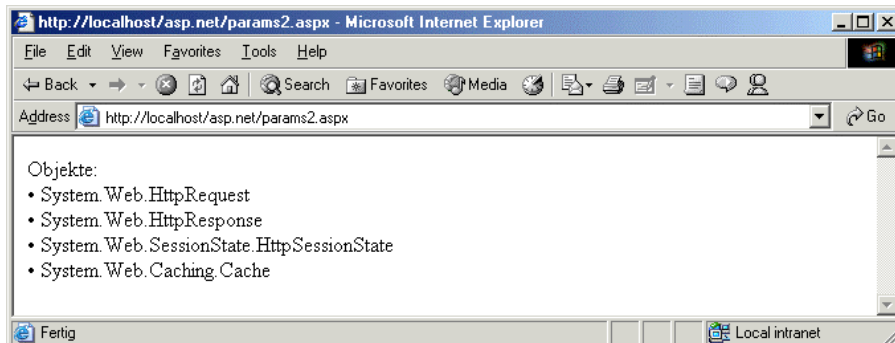


Abbildung 3.7 Auch beliebige Objekte lassen sich übergeben.



Beachten Sie bitte, dass die Angabe des Schlüsselwortes `params` nur einmal pro Methodendeklaration verwendet und anschließend keine weiteren Parameter notiert werden können.

Visual Basic .NET

Auch die zweite wichtige Sprache des .NET Frameworks verfügt über eine entsprechende Möglichkeit. Hier wird das Schlüsselwort `ParamArray` verwendet. Das Listing zeigt das umgewandelte erste Beispiel.

Listing 3.17 params1_vb.aspx

```
<script runat="server">

Sub Page_Load(Sender as Object, E As EventArgs)
    WriteNumbers("Zahlen", 1, 2, 3, 4, 5)
End Sub

Sub WriteNumbers(Text As String, ParamArray Values() As Integer)
    Response.Write(text & ": ")

    Dim i As Integer

    For Each i In Values
        Response.Write(i.ToString() & " ")
    Next

    Response.Write("<br>")
End Sub

</script>
```

3.8 ... eine Entsprechung zum VB-Schlüsselwort With in C# verwenden?

Die Sprache Visual Basic .NET kennt seit Version 4.0 ein äußerst hilfreiches Schlüsselwort mit dem Namen `with`. Über das Schlüsselwort wird ein Gültigkeitsbereich geöffnet. Ein nach dem Wort notiertes Objekt kann anschließend innerhalb des Bereiches verwendet werden, ohne dass der Variablenname erneut angegeben werden muss. Die Notation des Punkts reicht aus. Dies sieht beispielsweise wie im folgenden Listing gezeigt aus.

Listing 3.18 With1.aspx

```
<% @Page Language="VB" Debug="true" %>

<script runat="server">

Sub Page_Load(Sender As Object, E As EventArgs)

    With Response
        .Write("Hallo ")
        .Write("Welt!")
    End With

End Sub

</script>
```

Die Notation spart gerade bei vielen Zugriffen auf ein Objekt deutlich Zeit beim Schreiben des Quellcodes. Durch die Angabe des führenden Punktes bleibt zudem die Übersichtlichkeit gewahrt. Doch dies ist leider nur bei Verwendung von Visual Basic .NET als Entwicklungssprache der Fall. In C# steht das Schlüsselwort nicht zur Verfügung, auch eine analoge Entsprechung ist derzeit leider nicht vorgesehen.

In C# können Sie alternativ eine temporäre Variable mit einem (sehr) kurzen Namen verwenden. Dieser weisen Sie das gewünschte Objekt zu, benutzen es und terminieren die Variable anschließend wieder. Das folgende Listing zeigt eine entsprechende Implementierung:

Listing 3.19 With2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    HttpResponse r = Response;
    r.Write("Hallo ");
    r.Write("Welt!");
    r = null;
}

</script>
```

In einigen Newsgroups wurde zudem das Schlüsselwort `using` als Möglichkeit genannt, `with` nachzubilden. Gemeint ist die Anweisung, nicht die gleichnamige Direktive zum Einbinden von Namespaces. Der Ansatz ist jedoch nicht ganz richtig, denn die Anweisung hat eine andere Bedeutung. Schauen Sie sich bitte zunächst das folgende Listing an.

Listing 3.20 With3.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    using(HttpResponse r = Response)
    {
        r.Write("Hallo ");
        r.Write("Welt!");
    }
}

</script>

```

Prinzipiell sieht der Quellcode korrekt aus, doch das ist er nicht, vielmehr kommt es zu einem Kompilierungsfehler. Der Grund hierfür liegt in der Aufgabe von `using` begründet. Die Anweisung dient nicht etwa dem Einsparen von Tipparbeit, sondern wird zum Anlegen eines Gültigkeitsbereiches verwendet. Das angegebene Objekt wird nach dem Verlassen des Bereiches explizit terminiert. Hierzu muss dieses die Schnittstelle `IDisposable` und somit die Methode `Dispose` unterstützen. Dies gilt nicht für die Klasse `HttpResponse` (und wäre – selbst wenn – sehr gefährlich), und somit kann die Seite nicht kompiliert werden.

Das vierte Listing zeigt eine korrekte Verwendung von `using`. Hinter der Anweisung wird ein neues Objekt instanziiert. Es handelt sich um die Klasse `Bitmap`, mit Hilfe derer ein Bild in den Arbeitsspeicher geladen wird. Innerhalb des Gültigkeitsbereiches der Anweisung wird das Bild an den Client übertragen. Wird der Bereich verlassen, ist der Zugriff auf das Bild nicht mehr möglich. Im Listing ist dies durch einen Kommentar gekennzeichnet. Alle Ressourcen des Objekts wurden explizit mit Hilfe der Methode `Dispose` aus dem Speicher entfernt. Das Objekt selbst kann nun wie gewohnt durch die Gargabe `Collection` entsorgt werden.

Listing 3.21 With4.aspx

```

<% @Import Namespace="System.Drawing" %>
<% @Import Namespace="System.Drawing.Imaging" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.ContentType = "image/gif";
    string filename = Request.PhysicalApplicationPath +
        "hollywood-sign.gif";
}

```

```
using(Bitmap b = new Bitmap(filename))
{
    b.Save(Response.OutputStream, ImageFormat.Gif);
}
// Hier ist der Zugriff auf das Bild
// nicht mehr möglich!
}

</script>
```

Alternativ können Sie die Anweisung auch auf ein bestehendes Objekt anwenden. Auch hier ist der Zugriff auf das Objekt nach dem Verlassen des Gültigkeitsbereiches nicht weiter möglich.

Listing 3.22 With5.aspx

```
...
Bitmap b = new Bitmap(filename);
using(b)
{
    ...
}
```

Wie Sie sehen, existiert in C# derzeit leider kein direktes Abbild des Visual Basic-Schlüsselwortes `with`. Mit einer temporären Variablen lässt sich das Verhalten jedoch zumindest nachahmen. Die Anweisung `using` ist hingegen *keine* Alternative, da sie das Objekt terminiert und nur bei Klassen möglich ist, die die Schnittstelle `IDisposable` unterstützen.

3.9 ... einzelne Werte einer Enumeration abfragen?

Enumerationen sind ein hilfreiches Element zur Gestaltung von übersichtlichem Quellcode und einfach lesbaren Strukturen. Sie können einer Variablen einen von mehreren Werten einer Enumeration zuweisen und diesen an anderer Stelle wieder abfragen. Sie verwenden dabei in beiden Fällen statt numerischer Werte quasi Konstanten beziehungsweise Elemente der Enumeration.

Listing 3.23 Enum1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Test test = Test.Value1;
    if(test == Test.Value1)
```

```
{
    // ...
}
}

public enum Test
{
    Value1,
    Value2,
    Value3
}

</script>
```

Im einfachen Fall kann der Variablen nur jeweils ein Wert der Enumeration zugewiesen werden. Versehen Sie diese jedoch mit dem `Flags`-Attribut, so kann nun auch eine Kombination mehrerer Werte verknüpft zugewiesen werden. Diese werden verodert. Möchten Sie überprüfen, ob ein einzelner Wert der Enumeration gesetzt ist, ist ein einfacher Vergleich nicht mehr anwendbar. Sie müssen nun eine bitweise Verknüpfung durchführen. Das Listing zeigt dies.

Listing 3.24 Enum2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Test test = Test.Value1 | Test.Value3;

    if((test & Test.Value1) == Test.Value1)
        Response.Write("Value1<br>");

    if((test & Test.Value2) == Test.Value2)
        Response.Write("Value2<br>");

    if((test & Test.Value3) == Test.Value3)
        Response.Write("Value3<br>");
}

[Flags]
public enum Test
{
    Value1,
    Value2,
    Value3
}

</script>
```

Nacheinander werden die verschiedenen möglichen Werte abgefragt. Im Falle von „Value1“ und „Value3“ trifft die Bedingung zu, und der entsprechende Text wird im Browserfenster ausgegeben. Die Abbildung zeigt dies.

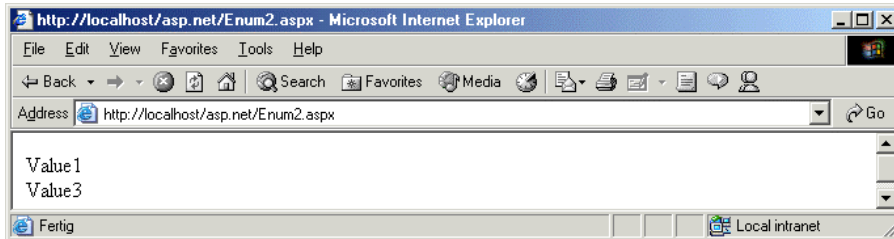


Abbildung 3.8 Die beiden Werte der Enumeration wurden gesetzt.

3.10 ... einen Enumerationswert aus einer Zeichenkette ermitteln?

Wird dem Benutzer eine Auswahl in Form von Multiple-Choice angeboten, steht im Hintergrund oft eine Enumeration mit korrespondierenden Werten. Um die Auswahl auf die Enumeration abzubilden, werden oftmals umständlich switch-Konstrukte verwendet. Das ist nicht notwendig, denn mit der statischen Methode `Parse` der Basisklasse `Enum` steht eine komfortablere Variante zur Verfügung.

Der Methode wird neben dem mittels `typeof` ermittelten Typ der Enumeration der gewünschte Wert als Zeichenkette übergeben. Sofern vorhanden wird das angeforderte Element der Enumeration zurückgeliefert – ganz ohne switch.

Das Beispiellisting zeigt die Verwendung der Methode. Über ein `DropDownList`-Control kann die Textausrichtung einer `CheckBox` gewählt werden. Je nach Auswahl verändert sich das Aussehen des zweiten Controls. Im Hintergrund wird der aktive Wert der Liste auf die Enumeration `TextAlign` abgebildet und der gleichnamigen Eigenschaft der `CheckBox` zugewiesen.

Listing 3.25 Enum3.aspx

```
<script runat="server">

void ddl_changed(object sender, EventArgs e)
{
    string value = ddl.SelectedItem.Value;
    TextAlign align = (TextAlign) Enum.Parse(typeof(TextAlign),
        value, true);
    cb.TextAlign = align;
}
```

```

</script>

<form runat="server">

    <p> Ausrichtung:
    <ASP:DropDownList id="ddl" runat="server"
        autopostback="true"
        OnSelectedIndexChanged="ddl_changed">
        <ASP:ListItem value="right" text="Rechts"/>
        <ASP:ListItem value="left" text="Links"/>
    </ASP:DropDownList>
    </p>

    <p> <ASP:CheckBox id="cb" runat="server" text="nur ein test"/> </p>

</form>

```

Sofern zu der übergebenen Zeichenkette kein korrespondierendes Element in der Enumeration existiert, wird eine Ausnahme von Typ `ArgumentException` geworfen. Um dies zu verhindern, können Sie zuvor mittels der statischen Methode `Enum.IsDefined` die Existenz des Wertes überprüfen.

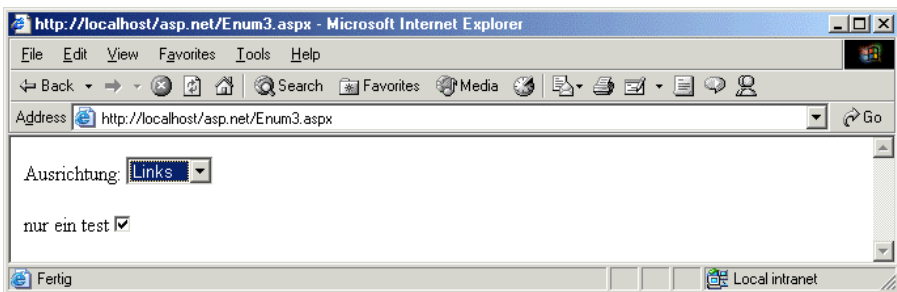


Abbildung 3.9 Der Text wird je nach Auswahl links oder rechts angezeigt.

3.11 ... alle Werte einer Enumeration auflisten?

Möchten Sie alle vorhandenen Werte einer Enumeration abfragen, verwenden Sie die statische Methode `GetNames` der Basisklasse `Enum`. Sie übergeben den Typ der Enumeration, den Sie zuvor mittels `typeof` beziehungsweise `GetType` ermittelt haben. Zurückgeliefert wird ein `string`-Array mit den Namen der einzelnen Werte.

Das folgende Beispiel basiert auf dem vorherigen aus dem Rezept „... einen Enumerationswert aus einer Zeichenkette ermitteln?“. Einem DropDownList-Control werden mittels DataBinding alle Werte der Enumeration `TextAlign` zugewiesen. Der Benutzer kann so die Textausrichtung des darunter platzierten CheckBox-Controls bestimmen.

Listing 3.26 Enum4.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    ddl.DataSource = Enum.GetNames(typeof(TextAlign));
    if(!IsPostBack) ddl.DataBind();
}

...

```

3.12 ... alle Eigenschaften eines Objekts abfragen?

Das .NET Framework bietet unter dem Stichwort Reflection Laufzeitzugriff auf die Metadaten beliebiger Objekte. Die Basis hierfür bildet die Klasse `Type`, die alle Informationen eines Datentyps repräsentiert. Eine Instanz dieser Klasse erhalten Sie über das Schlüsselwort `typeof` oder über die Methode `GetType`. Diese wird von der Mutter alle Objekte `object` implementiert und steht daher bei jedem Datentyp zur Verfügung.

Bevor Sie die Möglichkeiten von Reflection nutzen können, müssen Sie zunächst den entsprechenden Namespace einbinden:

```
<% @Import Namespace="System.Reflection" %>
```

Nun können Sie beispielsweise auf die Eigenschaften eines beliebigen Objekts zugreifen. Hierzu muss zunächst wie beschrieben eine Klasse `Type` instanziiert werden. Deren Eigenschaft `GetProperties` liefert ein Array mit allen Eigenschaften in Form eines `PropertyInfo`-Arrays. Jede Klasse im Array repräsentiert eine Eigenschaft des Objekts. Sie können beispielsweise den Namen ermitteln, und den Inhalt abfragen oder auch setzen.

Das Listing zeigt die Verwendung der Klasse `Type` zur Abfrage aller Eigenschaften eines beliebigen Objekts. Hierzu wurde die Methode `WriteProperties` implementiert, der das gewünschte Objekt übergeben wird. Alle Eigenschaften werden samt Namen (Eigenschaft `Name`) und Inhalt (Methode `GetValue`) im Browserfenster ausgegeben.

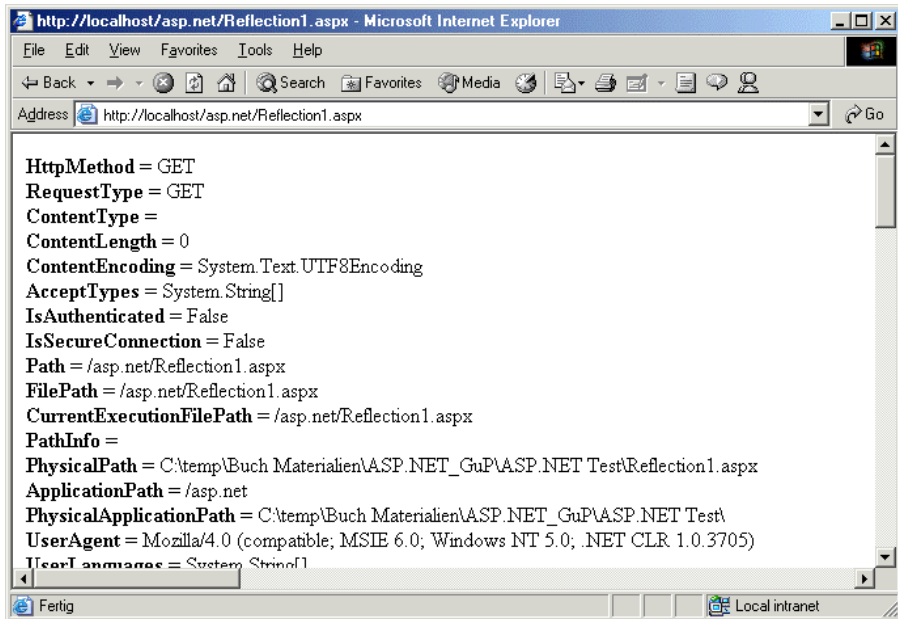


Abbildung 3.10 Alle Eigenschaften werden samt Inhalt ausgegeben.

Listing 3.27 Reflection1.aspx

```
<% @Import Namespace="System.Reflection" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    WriteProperties(this.Request);
}

void WriteProperties(object obj)
{
    Type t = obj.GetType();
    PropertyInfo[] properties = t.GetProperties();

    foreach(PropertyInfo property in properties)
    {
        try
        {
            Response.Write(string.Format("<b>{0}</b> = ", property.Name));
            Response.Write(string.Format("<{0}<br>",
                property.GetValue(obj, null)));
        }
        catch
    }
}
```

```
{
    Response.Write(" >>> Fehler <br>");
}
}
}

</script>
```

Die Abbildung zeigt deutlich die Ausgabe aller Eigenschaften der Klasse `HttpRequest` für die aktuelle Anforderung. Hierbei kann auch die undokumentierte Eigenschaft `Item` entdeckt werden, die in C# als Indexer zur Verfügung steht. Hierbei lässt sich ähnlich wie bei früheren ASP-Versionen der Inhalt der `Params`-Collection abfragen. Die drei folgenden Zeilen liefern allesamt die gleiche Ausgabe im Browserfenster:

```
Response.Write(Request["HTTP_USER_AGENT"]);
Response.Write(Request.Params["HTTP_USER_AGENT"]);
Response.Write(Request.ServerVariables["HTTP_USER_AGENT"]);
```

3.13 ... eine Eigenschaft per Reflection abfragen oder setzen?

Reflection ist eine wichtige Technologie, die den Laufzeitzugriff auf ein Objekt ermöglicht. Die Basis dafür bietet die Klasse `Type`, die einen Datentyp und somit auch eine Klasse zusammenfasst. Hierüber lassen sich alle (Meta-)Informationen des Objekts abfragen. Zudem können Sie objektorientiert auf die Mitglieder der Klasse zugreifen. Die Abfrage und das Setzen von Eigenschaften wird in diesem und dem nächsten Rezept beschrieben. Das sich daran anschließende Rezept beschreibt den Aufruf von Methoden.

Bevor Sie Reflections nutzen können, müssen Sie zunächst den entsprechenden Namespace einbinden:

```
<% @Import Namespace="System.Reflection" %>
```

Wie beschrieben, bildet die Klasse `Type` die Basis für Reflections. Um eine Instanz dieser zu erhalten, benutzen Sie die Methode `GetType`, die von `object` geerbt von jedem Objekt implementiert wird. Alternativ können Sie das Schlüsselwort `typeof` verwenden, dem der Name des entsprechenden Datentyps übergeben wird. Dies sieht beispielsweise so aus:


```
string str = "Hallo Welt";
Type t = str.GetType();
```

... oder ...

```
Type t = typeof(string);
```

Das umfangreiche Modell der Klasse unterstützt zwei Methoden zur Abfrage von Eigenschaften. `GetProperties` liefert alle verfügbaren Eigenschaften in Form eines `PropertyInfo`-Arrays. Eine einzelne Instanz derselben Klasse wird hingegen von der Methode `GetProperty` zurückgegeben, die zur Abfrage einer als Zeichenkette übergebenen Eigenschaft dient.

```
PropertyInfo p = t.GetProperty("Name");
```

Die Klasse `PropertyInfo` repräsentiert eine einzelne Eigenschaft eines Typs. Die Klasse ist dabei nicht an eine bestimmte Instanz des Typs, sondern nur an diesen selbst gebunden. Wenn Sie den Wert der Eigenschaft daher mittels der Methode `SetValue` festlegen wollen, müssen Sie immer auch die entsprechende Instanz übergeben.

Das folgende Listing zeigt den Einsatz von Reflections anhand eines Eingabeformulars. Der Inhalt eines `TextBox`-Controls soll einem Label zugewiesen werden. Dieses steht allerdings nur über eine als `object` deklarierte Variable zur Verfügung. Cobra (Reflection), übernehmen Sie ...

Listing 3.28 Reflection1.aspx

```
<% @Import Namespace="System.Reflection" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    object label = lb;
    Type t = label.GetType();
    PropertyInfo p = t.GetProperty("Text");
    p.SetValue(label, tb.Text, null);
}

</script>

<form runat="server">

    <p>
        <ASP:TextBox id="tb" runat="server" text="Hallo Welt!"/>
    </p>
```

```
<p> Ihre Eingabe:  
<ASP:Label id="lb" runat="server"/>  
</p>  
  
<ASP:Button runat="server" text="OK" onclick="bt_click"/>  
  
</form>
```

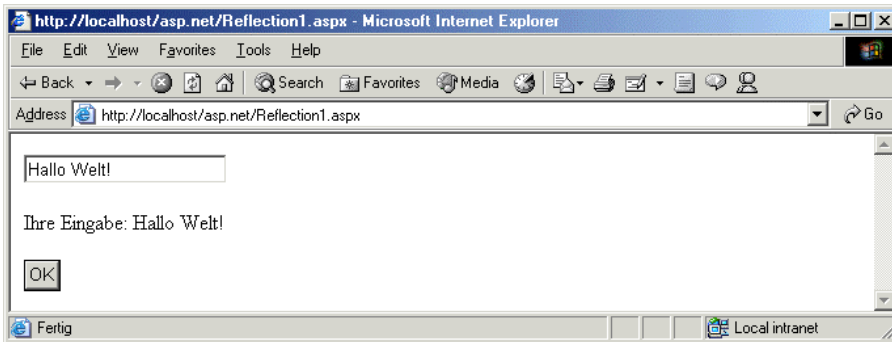


Abbildung 3.11 Der Text wurde dem Label erfolgreich zugewiesen.

Selbstverständlich können Sie den Inhalt einer Eigenschaft auch wieder abfragen. Hierzu sehen Sie im zweiten Listing die Methode `GetValue` im Einsatz, die ganz analog verwendet wird. Der Wert wird nun zunächst per Reflection abgefragt und anschließend wie im ersten Beispiel wieder zugewiesen.

Listing 3.29 Reflection2.aspx

```
<script runat="server">  
  
void bt_click(object sender, EventArgs e)  
{  
    object textbox = tb;  
    object label = lb;  
  
    Type t_tb = textbox.GetType();  
    PropertyInfo p_tb = t_tb.GetProperty("Text");  
    object value = p_tb.GetValue(textbox, null);  
  
    Type t_lb = label.GetType();  
    PropertyInfo p_lb = t_lb.GetProperty("Text");  
    p_lb.SetValue(label, value, null);  
}  
  
</script>
```

Wie Sie alle Eigenschaften und deren Wert abfragen können, lesen Sie im Rezept „... alle Eigenschaften eines Objekts abfragen?“.

3.14 ... eine indizierte Eigenschaft per Reflection abfragen oder setzen?

Im vorherigen Rezept „... eine Eigenschaft per Reflection abfragen oder setzen?“ haben Sie gelesen, wie Sie auf eine „normale“ Eigenschaft zur Laufzeit zugreifen können. Doch wie sieht es mit indizierten Eigenschaften aus? Hier kommt der jeweils letzte Parameter der Methode `GetValue` und `SetValue` zum Einsatz. Hier muss nun statt `null` der oder die Indexwerte in Form eines object-Arrays übergeben werden. Das Beispiel zeigt dies anhand einer Hashtable.

Listing 3.30 Reflection3.aspx

```
<% @Import Namespace="System.Reflection" %>

<script runat="server">

protected Hashtable values;

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        values = new Hashtable();
        ViewState["values"] = values;
    }
    else
        values = (Hashtable) ViewState["values"];
}

void bt_click(object sender, EventArgs e)
{
    Type t = values.GetType();
    PropertyInfo p = t.GetProperty("Item");
    p.SetValue(values, tb_value.Text, new object[] {tb_key.Text});

    lb.Text = string.Empty;
    foreach(object key in values.Keys)
    {
        string value = (string) p.GetValue(values, new object[] {key});
        lb.Text += (key.ToString() + " --> " + value + "<br>");
    }
}
```

```
</script>

<form runat="server">

  <p>
    Schlüssel:
    <ASP:TextBox id="tb_key" runat="server" text="Test"/> <br>
    Wert:
    <ASP:TextBox id="tb_value" runat="server" text="Hallo Welt!"/> <br>
  </p>

  <p> Ihre Eingaben:<br>
    <ASP:Label id="lb" runat="server"/>
  </p>

  <ASP:Button runat="server" text="OK" onclick="bt_click"/>

</form>
```

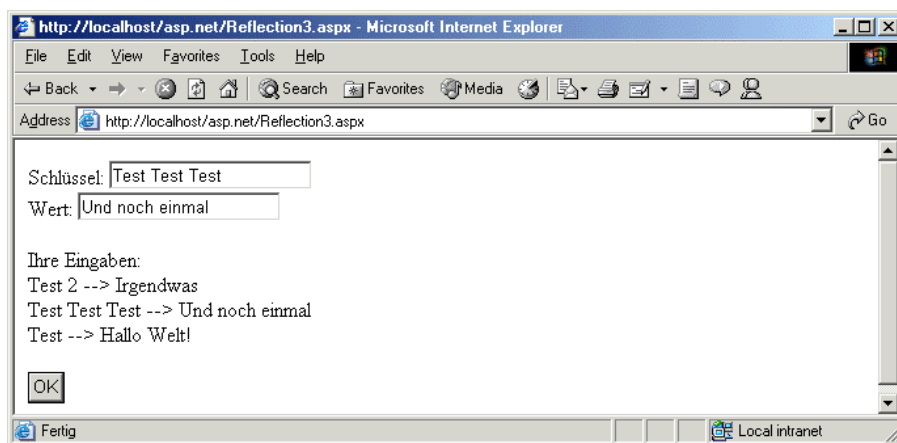


Abbildung 3.12 Auch auf indizierte Eigenschaften kann zugegriffen werden.



Vielleicht werden Sie sich fragen, welchen Eigenschaftsnamen Sie der Methode `GetProperty` übergeben sollen, denn schließlich greifen Sie ja ansonsten direkt auf den Indexer der Klasse zu. Intern läuft der Indexer unter dem Standardnamen „Item“, der auch im obigen Beispiel verwendet wurde.

3.15 ... eine Methode per Reflection aufrufen?

Die beiden vorangegangenen Rezepte haben den Zugriff auf Eigenschaften eines Objekts mittels Reflection gezeigt. Aufbauend darauf möchte ich Ihnen nun zeigen, wie Sie Methoden aufrufen können. Analog erhalten Sie hier über die Methoden `GetMethod` und `GetMethods` eine beziehungsweise mehrere Instanzen der Klasse `MethodInfo`, die eine einzelne Methode repräsentiert. Über `Invoke` kann die Methode letztlich ausgeführt werden. Die Parameter werden als `object-Array` übergeben.

Das Listing zeigt den Zugriff auf eine `Hashtable`. Mittels Reflections wird die Methode `Add` ausgeführt, und die Benutzereingaben werden dem `Dictionary` hinzugefügt.

Listing 3.31 Reflection4.aspx

```
<script runat="server">

protected Hashtable values;

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        values = new Hashtable();
        ViewState["values"] = values;
    }
    else
        values = (Hashtable) ViewState["values"];
}

void bt_click(object sender, EventArgs e)
{
    Type t = values.GetType();
    MethodInfo m = t.GetMethod("Add");
    m.Invoke(values, new object[] {tb_key.Text, tb_value.Text});

    lb.Text = string.Empty;
    foreach(DictionaryEntry de in values)
    {
        lb.Text += (de.Key.ToString() + " --> " +
            de.Value.ToString() + "<br>");
    }
}

</script>

<form runat="server">
```

```
<p>
  Schlüssel:
  <ASP:TextBox id="tb_key" runat="server" text="Test"/> <br>
  Wert:
  <ASP:TextBox id="tb_value" runat="server" text="Hallo Welt!"/> <br>
</p>

<p> Ihre Eingaben:<br>
  <ASP:Label id="lb" runat="server"/>
</p>

<ASP:Button runat="server" text="OK" onclick="bt_click"/>

</form>
```

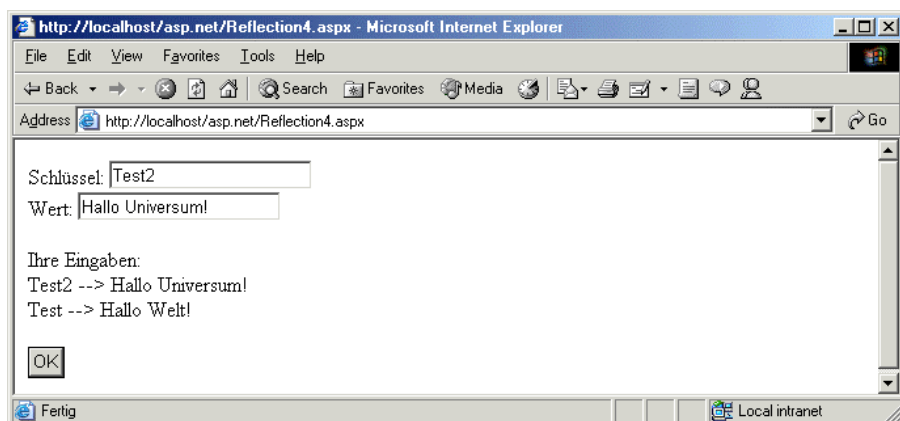


Abbildung 3.13 Die Elemente wurden über die Methode Add hinzugefügt.

Möglicherweise wollen Sie auch den Rückgabewert einer Methode erfragen. Mit Hilfe von Reflections kein Problem, denn `Invoke` liefert seinerseits den Wert der intern aufgerufenen Methode. Da als Rückgabewert `object` angegeben ist, muss vor der Verwendung des Ergebnisses noch eine explizite Typenkonvertierung erfolgen.

Listing 3.32 Reflection5.aspx

```
...
MethodInfo m_contains = t.GetMethod("Contains");
if((bool) m_contains.Invoke(values, new object[] {tb_key.Text}))
  Response.Write("\"Test\" ist enthalten.");
...
```

3.16 ... Detailinformationen über eine Komponente ermitteln?

Dank der neuartigen Technologie Reflection ist es im Grunde recht einfach möglich, programmatisch recht detaillierte Informationen über eine Assembly und die enthaltenen Klassen und Datentypen zu ermitteln. Mehrere Programme haben sich zur zentralen Aufgabe gemacht, diese Informationen möglichst allumfassend und möglichst übersichtlich zu präsentieren. Zwei dieser Programme möchte ich Ihnen an dieser Stelle vorstellen.

.NET Reflector

Das Programm .NET Reflector stammt vom deutschen Lutz Roeder. Es bietet einen übersichtlichen Einstieg in die verschiedenen Assemblies, Namespaces und Klassen des Systems. Zu jedem Typ können Eigenschaften, Methoden und auch über- beziehungsweise untergeordnete Typen abgefragt werden. Soweit vorhanden zeigt das Programm auch die entsprechende XML-Dokumentation sowie den IL-Bytecode an.

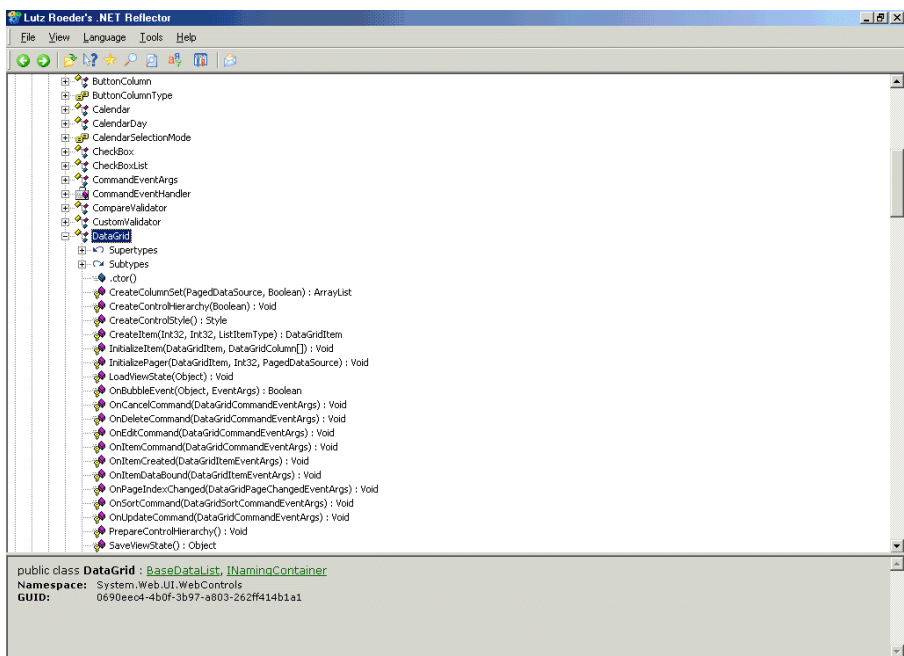


Abbildung 3.14 .NET Reflector im Einsatz mit dem DataGrid-Control

Die jeweils aktuelle Version des kostenlosen Programms finden Sie unter folgender Adresse:

www.aisto.com/roeder/dotnet/

Web Matrix Class Browser

Der ASP.NET-Editor Web Matrix wurde von Microsoft-Mitarbeitern entwickelt und wird von diesen kostenlos zur Verfügung gestellt. Quasi als Nebenprodukt enthält Web Matrix einen externen Class Browser, der ganz ähnlich wie .NET Reflector arbeitet. Auch hier werden zunächst Assemblies, Namespaces und Klassen zur Auswahl angeboten. Ein Klick öffnet ein MDI-Fenster mit den einzelnen Klassenmitgliedern und zeigt Detailinformationen an. Stammt ein Datentyp aus dem offiziellen Framework, kann per Mausklick die lokale oder Web-Dokumentation angezeigt werden.

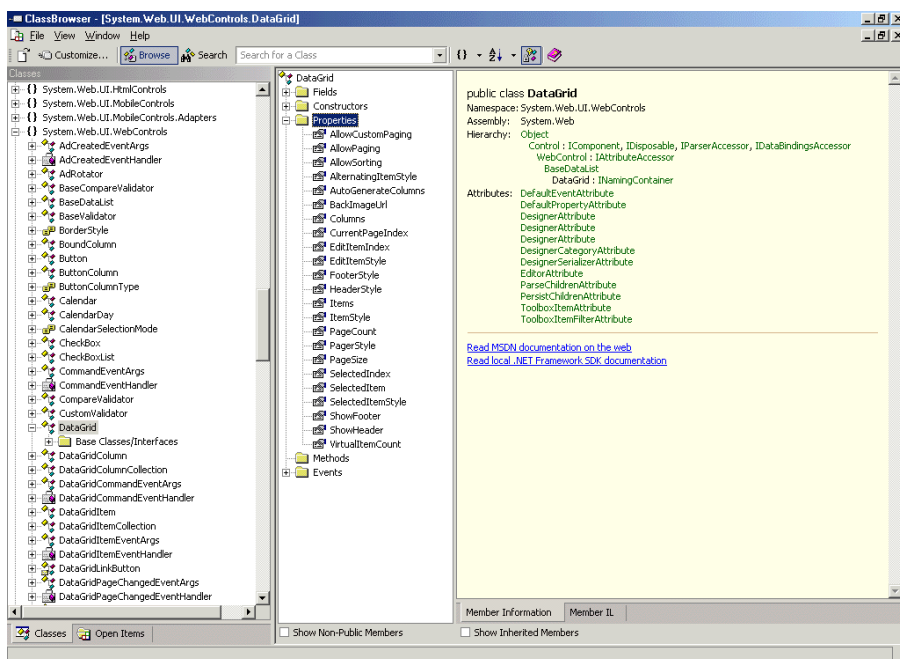


Abbildung 3.15 Auch der Class Browser kann Informationen zum DataGrid anzeigen.

Der Class Browser ist Bestandteil von Web Matrix und kann kostenlos auf der offiziellen ASP.NET-Website von Microsoft heruntergeladen werden:

<http://www.asp.net>

3.17 ... eine eigene Collection erstellen?

Collections sind ein wichtiges Hilfsmittel zur Organisation wiederkehrender Daten. In Programmstrukturen lassen sich diese oftmals auch sinnvoll nach außen geben. So handelt es sich beispielsweise bei der `DataGridColumnCollection` um eine selbige. Gegenüber einer untypisierten Klasse wie der `ArrayList` hat eine typisierte Collection den Vorteil, dass nur Elemente von einem explizit zugelassenen Datentyp angefügt werden können. Man bezeichnet derartige Collections auch als *strongly typed*.

Wenn Sie selbst eine solche Collection implementieren möchten, können Sie natürlich die entsprechenden Schnittstellen manuell unterstützen. Es handelt sich um `IEnumerable`, `ICollection` und unter Umständen `IList`. Insgesamt verfügen die drei Schnittstellen über 15 Mitglieder – viel Spaß!

Doch die Entwickler bei Microsoft haben vorgedacht und stellen daher eine abstrakte Basisklasse `CollectionBase` zur Verfügung, die intern eine `ArrayList` verwendet und die notwendigen Mitglieder bereits in Form expliziter Schnittstellenunterstützung enthält. In einer abgeleiteten Klasse können Sie diese Mitglieder durch eigene, öffentliche ersetzen. In der Wahl der Parameter sind Sie frei, so dass Sie hier die gewünschten Typen angeben können.

Im Beispiel sehen Sie eine individuelle Implementierung einer Collection auf Basis der abstrakten Klasse `CollectionBase`. Von den möglichen, zu implementierenden Mitgliedern `CopyTo`, `Add`, `Contains`, `IndexOf`, `Insert` und `Remove` wurden die drei wichtigsten implementiert. Die Aufrufe werden an die intern verwaltete `ArrayList`-Instanz weitergereicht, die über die als `protected` markierte Eigenschaft `InnerList` zur Verfügung steht. Als Grundelement für die Collection habe ich die Struktur `DateTime` gewählt, die individuelle Wahl obliegt jedoch Ihren Anforderungen.

Listing 3.33 `CollectionBase.cs`

```
using System;
using System.Collections;

namespace PAL.Projects.AspNetKochbuch
{
    public class DateTimeCollection : CollectionBase
```

```
{
    public void Add(DateTime dateTime)
    {
        InnerList.Add(dateTime);
    }

    public bool Contains(DateTime dateTime)
    {
        return(InnerList.Contains(dateTime));
    }

    public void Remove(DateTime dateTime)
    {
        InnerList.Remove(dateTime);
    }
}
```

Ich habe die Implementierung innerhalb einer C#-Quellcode-Datei vorgenommen. Diese muss vor der Benutzung mit `csc.exe` kompiliert und im `bin`-Verzeichnis der Web-Applikation abgelegt werden. Anschließend kann die Klasse mit dem Namen `DateTimeCollection` eingesetzt werden. Das zweite Listing zeigt dies anhand einiger errechneter Datumswerte. Diese werden der Collection angefügt und anschließend über eine `foreach...in`-Schleife wieder ausgegeben.

Listing 3.34 `DateTimeCollection1.aspx`

```
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DateTimeCollection dtcol = new DateTimeCollection();

    for(int i=0; i<10; i++)
    {
        dtcol.Add(DateTime.Now.AddDays(i));
    }

    foreach(DateTime dt in dtcol)
    {
        Response.Write(dt.ToString() + "<br>");
    }
}

</script>
```

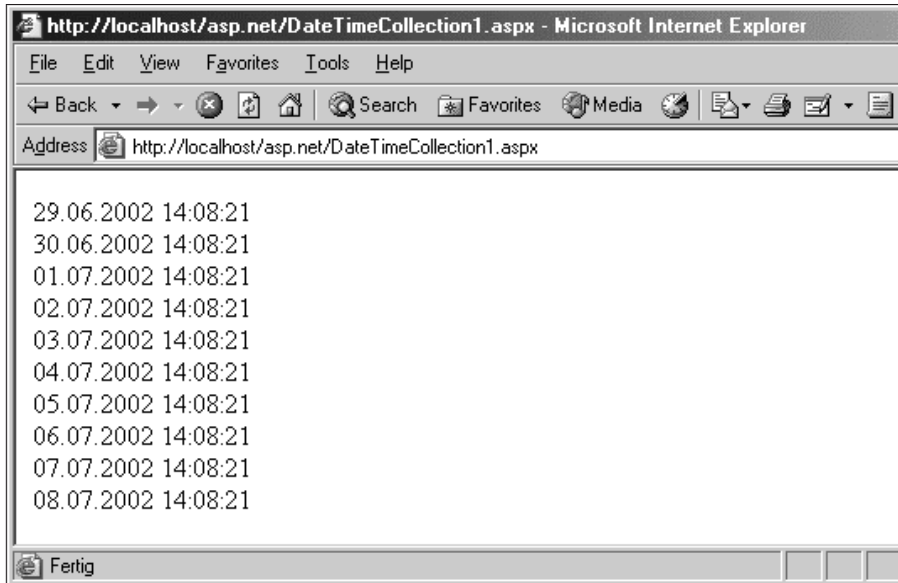


Abbildung 3.16 Die Collection kann nur Instanzen der Struktur DateTime aufnehmen.

Das gezeigte Verhalten ließe sich selbstverständlich auch mit einer regulären `ArrayList` erreichen. Hier würde allerdings auch das Hinzufügen beispielsweise einer Zeichenkette funktionieren, was bei der `DateTimeCollection` zum Glück nicht der Fall ist. Der folgende Versuch scheitert daher.

Listing 3.35 `DateTimeCollection2.aspx`

```
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DateTimeCollection dtcol = new DateTimeCollection();
    dtcol.Add("Hallo Welt");
}

</script>
```

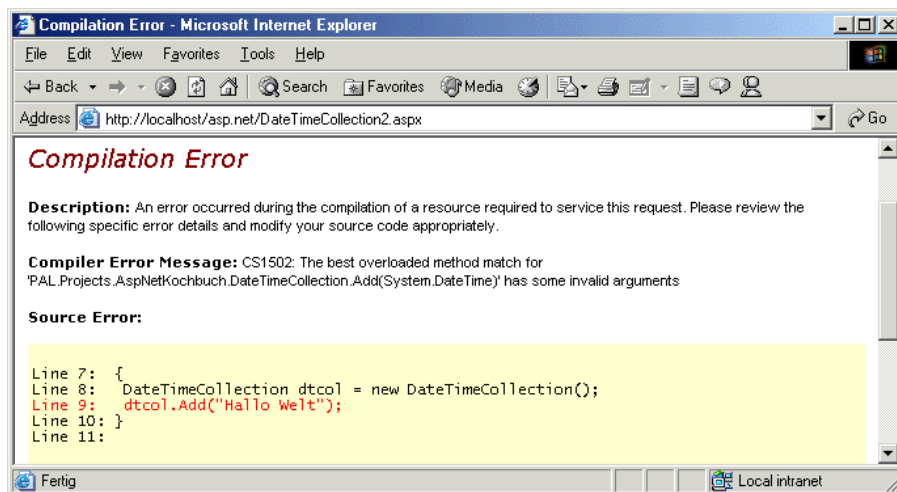


Abbildung 3.17 Zeichenketten oder andere Datentypen lassen sich nicht einfügen.

3.18 ... ein eigenes Dictionary erstellen?

Im vorherigen Rezept haben Sie gesehen, wie Sie mit einfachen Handgriffen eine individuelle Collection erstellen können. Analog ist dies auch für Dictionaries möglich. Die abstrakte Basisklasse `DictionaryBase` unterstützt dazu die Schnittstellen `IEnumerable`, `ICollection` und `IDictionary`. Zur individuellen Implementierung werden hier die Methoden `Add`, `Contains` und `Remove` angeboten. Im Listing ist ein neues `DateTimeDictionary` zu sehen, bei dem jeder Zeitwert mit einer eindeutigen Zeichenkette als Schlüssel versehen wird¹.

Listing 3.36 `DictionaryBase.cs`

```
using System;
using System.Collections;

namespace PAL.Projects.AspNetKochbuch
{
    public class DateTimeDictionary : DictionaryBase
    {
        public void Add(string key, DateTime dateTime)
```

¹ Man kann sich darüber streiten, ob ein `DateTimeDictionary` in der Praxis tatsächlich Anwendung findet. Dennoch zeigt die Implementierung deutlich den Unterschied zu der zuvor beschriebenen `DateTimeCollection`, weswegen ich an dieser Stelle keinen anderen Datentyp gewählt habe.

```
{
    InnerHashtable.Add(key, dateTime);
}

public bool Contains(string key)
{
    return(InnerHashtable.Contains(key));
}

public void Remove(string key)
{
    InnerHashtable.Remove(key);
}

public DateTime this[string key]
{
    get { return((DateTime) InnerHashtable[key]); }
    set { InnerHashtable[key] = value; }
}
}
```

Neben den genannten Methoden habe ich zusätzlich noch einen Indexer implementiert, der auf den Indexer der intern über die Eigenschaft `InnerHashtable` verwalteten `Hashtable` verweist. Im Beispiel sehen Sie den Einsatz der neuen Klasse.

Listing 3.37 DateTimeDictionary1.aspx

```
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DateTimeDictionary dtdic = new DateTimeDictionary();

    dtdic.Add("Vorgestern", DateTime.Now.AddDays(-2));
    dtdic.Add("Gestern", DateTime.Now.AddDays(-1));
    dtdic.Add("Heute", DateTime.Now);
    dtdic.Add("Morgen", DateTime.Now.AddDays(1));
    dtdic.Add("Übermorgen", DateTime.Now.AddDays(2));

    foreach(DictionaryEntry de in dtdic) {
        Response.Write(de.Key + ": " +
            ((DateTime) de.Value).ToString("d") + "<br>");
    }
}

</script>
```

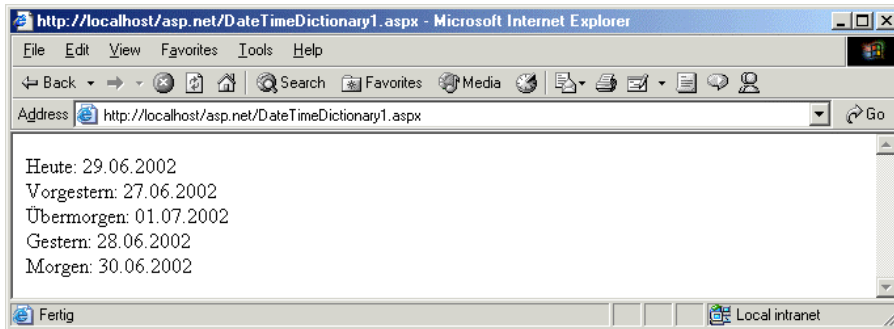


Abbildung 3.18 Im Dictionary werden Datumsangaben mit einem Schlüssel abgelegt.

3.19 ... Hilfsfunktionen global hinterlegen?

Sie kennen das sicherlich auch: eine oft benötigte Hilfsfunktion soll global hinterlegt werden, damit diese ad hoc in jeder Seite zur Verfügung steht. Mangels einfacher Möglichkeiten bleibt es oft dann doch bei der alten Variante, das Rad in jeder Seite neu zu erfinden. Doch das muss nicht sein, denn es gibt eine einfache Möglichkeit für derartige Funktionen.

Zunächst erstellen Sie die gewünschte Funktionalität. Hierzu legen Sie eine neue Quellcode-Datei an, referenzieren die benötigten Namespaces und legen eine neue Klasse an. Notieren Sie die Klasse bitte ohne zusätzliche Verschachtelung in einen eigenen Namespace direkt im obersten Gültigkeitsbereich. Innerhalb der Klasse legen Sie nun die Funktion als *statische* Methoden.

Die im Beispiellisting gezeigte Methode `GetGreeting` liefert eine tageszeitabhängige Begrüßung in Form einer Zeichenkette; ziemlich simpel, aber für eine Vorstellung der Technik genügt's.

Listing 3.38 `HelperMethods1.cs`

```
using System;
using System.Web;

public class Helper
{
    public static string GetGreeting()
    {
        int hour = DateTime.Now.Hour;

        if(hour <= 10) return("Guten Morgen");
        if(hour <= 17) return("Guten Tag");
    }
}
```

```
    return("Guten Abend");  
}  
  
}
```

Nachdem Sie die Quellcode-Datei erstellt haben, kompilieren Sie diese mittels `csc.exe` und verschieben die so erzeugte DLL in das bin-Verzeichnis Ihrer Web-Applikation:

```
csc /t:library HelperMethods1.cs
```

Das war's auch schon; ab sofort steht die entwickelte Methode global in jeder Seite der Web-Applikation zur Verfügung. Das glauben Sie nicht? Ich beweise es Ihnen mit der folgenden Beispielseite.

Listing 3.39 HelperTest1.aspx

```
<script runat="server">  
  
void Page_Load(object sender, EventArgs e)  
{  
    Response.Write(Helper.GetGreeting());  
}  
  
</script>
```

Die Abbildung zeigt, dass es wirklich funktioniert, und das ohne jegliche explizite Einbindung oder Referenzierung eines Namespaces. O.K., aus Sicht der objektorientierten Programmierung ist das System vielleicht nicht einwandfrei, sinnvoll eingesetzt bietet es dafür aber ein enormes Potenzial zum Zeitsparen.

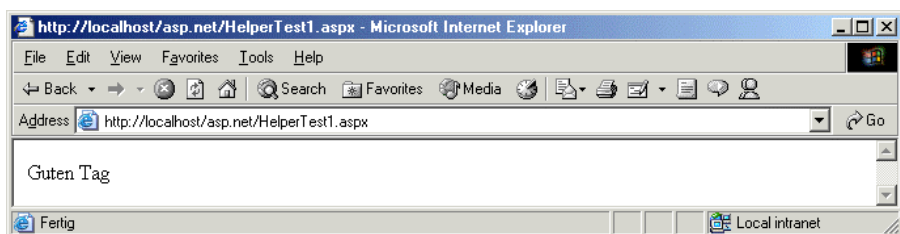


Abbildung 3.19 Die Begrüßung wurde von der Hilfsmethode erzeugt.

Selbstverständlich können Sie die statische Methode auch mit Parametern oder komplexeren Rückgabewerten wie Instanzen eigener Klassen ausstatten. Den Möglichkeiten sind keine Grenzen gesetzt. So verwundert es auch nicht, dass Sie die Hilfsfunktion über die reguläre DataBinding-Syntax nutzen können:

Listing 3.40 HelperTest2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DataBind();
}

</script>

<%% Helper.GetGreeting() %>
```

Sie können diese schöne Möglichkeit natürlich auch mit anderen Programmiersprachen wie beispielsweise Visual Basic .NET nutzen. Sie müssen lediglich sicherstellen, dass sowohl die Klasse als auch die Methode als öffentlich markiert sind. Letztere muss zudem statisch implementiert werden, damit die Klasse nicht erst instanziiert werden muss. Das nächste Listing zeigt eine analoge Variante für Visual Basic .NET.

Listing 3.41 HelperMethods1.vb

```
Imports System

Public Class Helper

    Public Shared Function GetGreeting() As String

        Dim hour As Integer = DateTime.Now.Hour

        Dim welcome As String

        If hour <= 10 Then
            welcome = "Guten Morgen"
        ElseIf hour <= 17 Then
            welcome = "Guten Tag"
        Else
            welcome = "Guten Abend"
        End If

        GetGreeting = welcome

    End Function

End Class
```


Statt mit `csc.exe` muss diese Quellcode-Datei mit dem spezifischen Compiler `vbc.exe` umgesetzt werden:

```
vbc /t:library HelperMethods1.vb
```

Auch die so erzeugte DLL muss natürlich im bin-Verzeichnis der Web-Applikation abgelegt werden. Anschließend können Sie die gezeigten Beispiele ohne Änderung weiterverwenden. Diesen ist es schließlich „egal“, ob die aufgerufene Methode in C# oder VB geschrieben wurde.



Eines sollten Sie unbedingt vermeiden. Verwenden Sie für die Benennung der Klasse keinen bereits vorhandenen Namen, sondern einen zutiefst eigenen. Und diesen sollten Sie auch nur genau einmal verwenden. Sofern Sie weitere Hilfsklassen einführen, verwenden Sie einen anderen Namen oder erweitern einfach die ursprüngliche Klasse.

3.20 ... innerhalb einer beliebigen Klasse auf die ASP.NET-Objekte zugreifen?

Innerhalb der vorgegebenen Klasse wie `Page` und `UserControl` ist der Zugriff auf das ASP.NET-Objektmodell problemlos möglich. Die verschiedenen Objekte stehen hier als Eigenschaften zur Verfügung und können direkt verwendet werden. Doch wie ist der Zugriff auf diese so wichtigen Elemente möglich, wenn Sie sich innerhalb einer individuellen Klasse befinden? Sie könnten hier einen Verweis auf die übergeordnete Seite (Klasse `Page`) übergeben oder, noch viel schlimmer, Verweise auf alle benötigten Objekte wie `Request` und `Response`. Doch wirklich schön ist das nicht und notwendig schon gar nicht.

Die gesamte Bearbeitung einer Client-Anfrage mitsamt allen benötigten Objekten wird als Kontext zusammengefasst. Es handelt sich dabei um eine Instanz der Klasse `HttpContext`. Über entsprechende Eigenschaften bietet diese analog zur Klasse `Page` Zugriff auf das gesamte benötigte Objektmodell. Verständlicherweise reicht es nicht, die Klasse manuell zu instanziiieren, denn Sie wollen ja vermutlich auf die aktuelle Client-Anfrage zugreifen. Zugriff auf diesen ausgewählten Kontext erhalten Sie über die statische Eigenschaft `Current`. Die Verwendung sieht daher beispielsweise so aus:

```
HttpContext Context = HttpContext.Current;
HttpResponse Response = Context.Response;
HttpRequest Request = Context.Request;
...
```

Das Listing zeigt die Verwendung dieser Zugriffsart. Gezeigt wird eine Klasse `Calc`, die als Quasi-Taschenrechner fungiert. Im Konstruktor lässt sich ein initiieller Wert übergeben, dem mittels `Add` Werte hinzuaddiert werden können. Das über die Eigenschaft `value` angebotene Ergebnis lässt sich mit Hilfe der Methode `WriteToClient` direkt an den Client senden. Auf das Objektmodell wird über die gezeigte Klasse `HttpContext` zugegriffen.

Listing 3.42 Context1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Calc calc = new Calc(20);
    calc.Add(25.4545);
    calc.WriteToClient();
}

class Calc
{
    double _Value;

    public Calc(double value)
    {
        _Value = value;
    }

    public double Value
    {
        get { return(_Value); }
    }

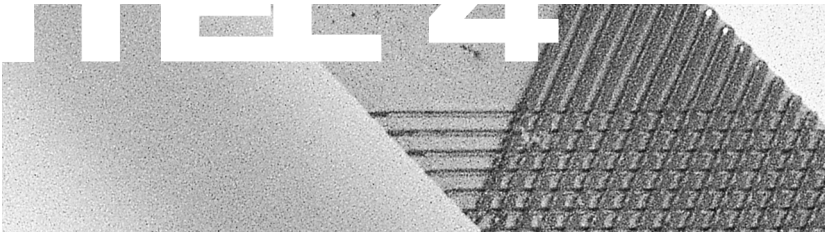
    public void Add(double value)
    {
        _Value += value;
    }

    public void WriteToClient()
    {
        HttpContext Context = HttpContext.Current;
        HttpResponse Response = Context.Response;
        Response.Write(this.Value);
    }
}

</script>
```


4

Zeichenketten



Wie kann ich ...

4 Zeichenketten

Strings oder auch Zeichenketten sind elementarer Bestandteil der Programmierung. Insbesondere wenn ein Benutzer mit einem Programm interagiert, ist die Verwendung und Überprüfung von Zeichenketten obligatorisch. In diesem Kapitel finden Sie zahlreiche Tipps und kleine Lösungen zum alltäglichen Umgang mit Zeichen und Zeichenketten.

4.1 ... überprüfen, ob eine Zeichenkette einen numerischen Wert enthält?

Visual Basic .NET bietet mit der Funktion `IsNumeric` eine einfache Möglichkeit, zu überprüfen, ob eine Zeichenkette einen numerischen Wert enthält und somit mittels der entsprechenden Konvertierungsmethoden umgewandelt werden kann. Dies ist insbesondere in Verbindung mit Benutzereingaben eine sehr wichtige Funktionalität.

Listing 4.1 IsNumeric1.aspx

```
<script runat="server">

Sub Page_Load(sender as Object, e as EventArgs)

    Response.Write(string.Format("Ist ""{0}"" numerisch? {1}<br>",
        "25", IIf(IsNumeric("25"), "Ja", "Nein")))
    Response.Write(string.Format("Ist ""{0}"" numerisch? {1}<br>",
        "aaa25", IIf(IsNumeric("aaa25"), "Ja", "Nein")))

End Sub

</script>
```

In C# existiert keine analoge Möglichkeit, so dass Sie diese manuell implementieren müssen. Nachfolgend ein Entwurf einer entsprechenden Methode. Es wird hierbei versucht, die übergebene Zeichenkette in einen int-Wert umzuwandeln. Misslingt dies, wird eine Ausnahme geworfen. Ist dies der Fall, wird die Ausnahme direkt innerhalb der Methode behandelt und ein negativer Rückgabewert geliefert.

Listing 4.2 IsNumeric2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write(string.Format("Ist \"{0}\" numerisch? {1}<br>",
        "25", IsNumeric("25")?"Ja":"Nein"));
    Response.Write(string.Format("Ist \"{0}\" numerisch? {1}<br>",
        "aaa25", IsNumeric("aaa25")?"Ja":"Nein"));
}

bool IsNumeric(string value)
{
    try
    {
        int.Parse(value);
        return(true);
    }
    catch
    {
        return(false);
    }
}

</script>
```

In beiden Fällen wird korrekt erkannt, dass der Wert „25“ konvertiert werden kann, der Wert „aaa25“ jedoch nicht.

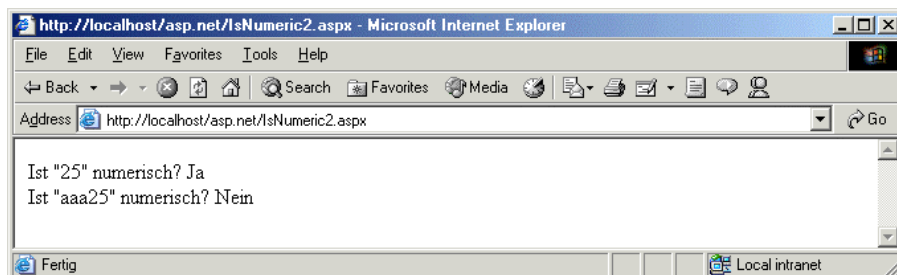


Abbildung 4.1 IsNumeric erkennt, ob es sich um einen numerischen Wert handelt.

Eine weitere Möglichkeit ist die Iteration aller Zeichen und deren Typen mit Hilfe der statischen Methoden des Datentyps `char`. Informationen hierzu finden Sie im Rezept „... ermitteln, um was für eine Art von Zeichen es sich handelt?“.

4.2 ... überprüfen, ob eine Zeichenkette einen Datumswert enthält?

Analog zu `IsNumeric` bietet Visual Basic .NET auch eine Funktion `IsDate`, mit Hilfe derer ein Datumswert in einer Zeichenkette zuverlässig erkannt werden kann. Auch diese Funktionalität ist in C# nicht vorhanden, lässt sich aber einfach nachrüsten.

Listing 4.3 `IsDate1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write(string.Format("Ist \"{0}\" ein Datum? {1}<br>",
        "07.04.2002", IsDate("07.04.2002")?"Ja":"Nein"));
    Response.Write(string.Format("Ist \"{0}\" ein Datum? {1}<br>",
        "heute", IsDate("heute")?"Ja":"Nein"));
}

bool IsDate(string value)
{
    try
    {
        DateTime.Parse(value);
        return(true);
    }
    catch
    {
        return(false);
    }
}

</script>
```

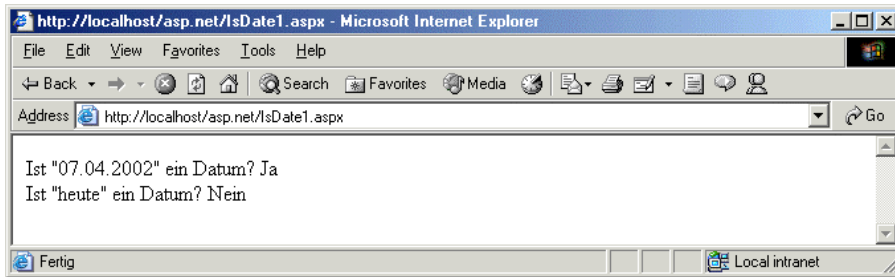


Abbildung 4.2 Die Zeichenkette „heute“ ist keine gültige Datumsangabe.

4.3 ... zwei Teilzeichenketten vertauschen?

Mitunter sollen in einer gegebenen Zeichenkette zwei Wörter oder Teilbereiche vertauscht werden. Die Klasse bietet hierzu leider keine entsprechende Methode `Swap` oder Ähnliches an. Wie so oft ist daher Eigeninitiative gefragt.

Das Vertauschen von Teilzeichenketten funktioniert über die `Replace`-Methode. Das Kernproblem ist hierbei, nicht alle Elemente mit demselben Inhalt zu ersetzen. Damit dies nicht geschieht, sind drei Ersetzungen erforderlich. Zunächst muss die erste Zeichenkette gegen einen temporären Wert ersetzt werden. Hierzu kann ein Hashcode benutzt werden. Anschließend wird der zweite Wert durch den ersten ersetzt und der temporäre Wert durch den zweiten. Das Listing zeigt diesen Trick.

Listing 4.4 `Swap1.aspx`

```
void Page_Load(object sender, EventArgs e)
{
    string text = "Unsere Erde ist eine kleine Welt.";
    text = Swap(text, "Erde", "Welt");
    Response.Write(text);
}

string Swap(string text, string swap1, string swap2)
{
    string tmp = swap1.GetHashCode().ToString();

    text = text.Replace(swap1, tmp);
    text = text.Replace(swap2, swap1);
    text = text.Replace(tmp, swap2);

    return(text);
}
```

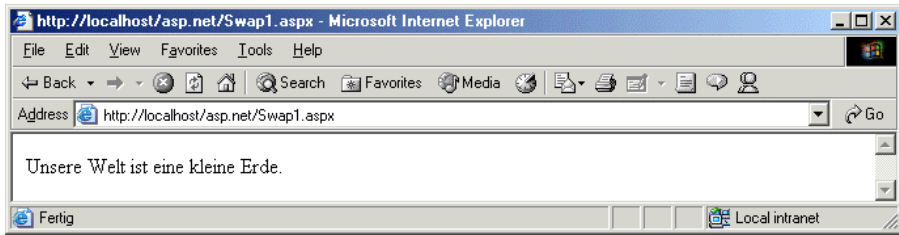



Abbildung 4.3 Die Wörter Erde und Welt wurden vertauscht.

4.4 ... die ersten x Wörter einer Zeichenkette abfragen?

Internetangebote bestehen zu einem großen Prozentsatz aus Textseiten. Oftmals kommen die benötigten Inhalte aus einer Datenbank. Aus Gründen der Übersichtlichkeit kommt es vor, dass statt des gesamten Textes nur ein Teil dargestellt werden soll. Über einen Button oder Link kann der Benutzer auf Wunsch die vollständige Version anfordern.

Die Implementierung eines solchen Systems ist sehr einfach, doch wie soll der Kurztext erzeugt werden? Ein Abschneiden nach vielleicht 100 Zeichen ist zwar machbar, allerdings nicht besonders schön. Es bietet sich an, nach einer festgelegten Anzahl von Wörtern abzuschneiden. In Ermangelung einer „schönen“ Methode der Klasse `string` bedarf es hierzu manueller Zeichenkettenoperationen.

Das Listing zeigt die Implementierung einer Methode `GetWords` sowie deren Verwendung. Übergeben wird der Originaltext sowie die Anzahl der zurückzuliefernden Wörter. Eine einfache `for`-Schleife in Verbindung mit der Methode `string.IndexOf` liefert die Teilzeichenkette.

Listing 4.5 `GetWords1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string text = "Hallo Welt, dies ist ein ganz toller Text.";
    string words = GetWords(text, 5);
    Response.Write(words);
}

string GetWords(string source, int wordcount)
{
    int pos = 0;
    for(int i=0; i<wordcount; i++)
    {
```

```
pos = source.IndexOf(' ', pos+1);  
if(pos == -1) return(source);  
}  
return(source.Substring(0, pos));  
}  
  
</script>
```

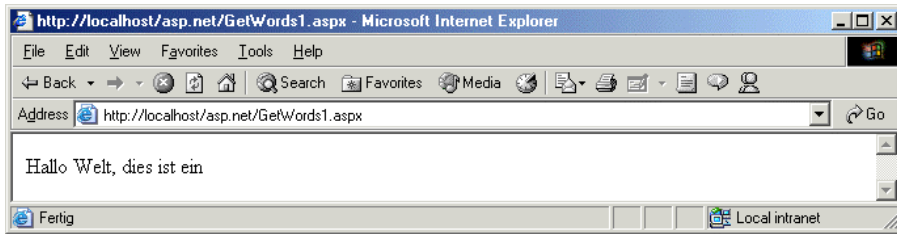


Abbildung 4.4 Nach fünf Wörtern ist Schluss.

Wenngleich in der Praxis vermutlich selten gebraucht, können Sie die Methode die Wörter auch vom Ende der Zeichenkette nehmen lassen.

Listing 4.6 GetWords2.aspx

```
<script runat="server">  
  
void Page_Load(object sender, EventArgs e)  
{  
    string text = "Hallo Welt, dies ist ein ganz toller Text.";  
    string words = GetWordsRight(text, 5);  
    Response.Write(words);  
}  
  
string GetWordsRight(string source, int wordcount)  
{  
    int pos = source.Length;  
    for(int i=0; i<wordcount; i++)  
    {  
        pos = source.LastIndexOf(' ', pos-1);  
        if(pos == -1) return(source);  
    }  
    return(source.Substring(pos));  
}  
  
</script>
```

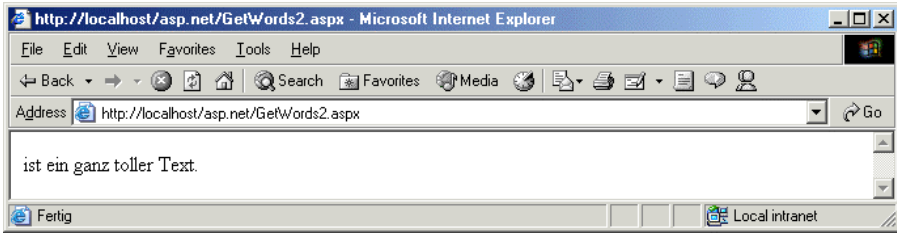


Abbildung 4.5 Hier wurden die fünf letzten Wörter ermittelt.

4.5 ... die Zeichen einer Zeichenkette iterieren?

Die Klasse `String`, die hinter jeder Zeichenkette steht, bietet verschiedene Ansätze, durch die einzelnen Zeichen zu iterieren. Jedes Zeichen wird dabei über den Datentyp `Char` repräsentiert.

Enumerator

Variante Nummer eins verwendet den Enumerator der Klasse. Dieser wird über die Schnittstelle `IEnumerable` implementiert und kann wie gewöhnlich über eine `foreach...in`-Schleife angesprochen werden.

Listing 4.7 chars1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt";

    foreach(char c in str)
        Response.Write(c + "<br>");
}

</script>
```

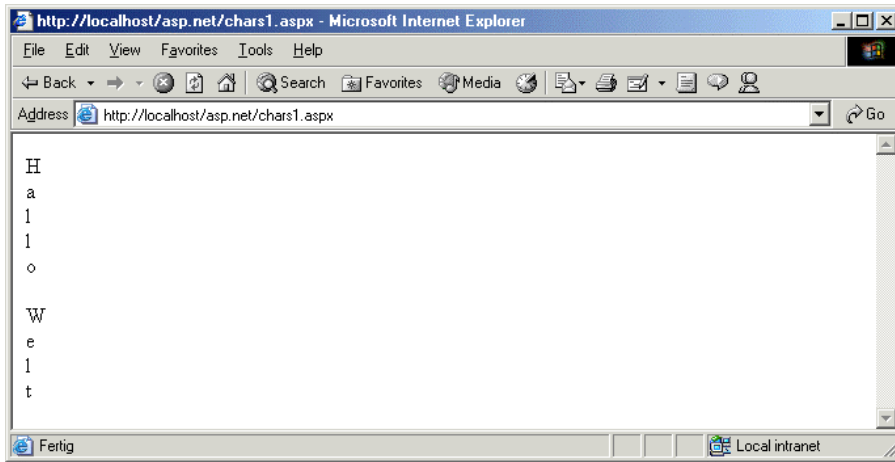


Abbildung 4.6 Alle Zeichen der Zeichenkette werden einzeln ausgegeben.

Indexer

Variante Nummer zwei verwendet den numerischen Indexer, den die Klasse `String` ebenfalls unterstützt. Mit Hilfe einer regulären `for`-Schleife lassen sich so die einzelnen Zeichen abfragen.

Listing 4.8 chars2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt";

    for(int i=0; i<str.Length; i++)
        Response.Write(str[i] + "<br>");
}

</script>
```

Da Klassen in Visual Basic .NET nicht über einen Indexer verfügen, muss hier explizit die Eigenschaft `Chars` zum Zugriff auf die Zeichen angegeben werden. Die VB-Umsetzung sieht folglich so aus:

Listing 4.9 chars2_vb.aspx

```
<script runat="server">

Sub Page_Load(Sender As Object, E As EventArgs)

    Dim str As String = "Hallo Welt"

    Dim i As Integer
    For i = 0 To str.Length-1
        Response.Write(str.Chars(i) & "<br>")
    Next

End Sub

</script>
```

Char-Array

Die dritte und letzte Variante, die ich Ihnen vorstellen möchte, verwendet die Methode `ToCharArray`, um die Zeichenkette in ein `char`-Array umzuwandeln. Dieses kann mit den üblichen Möglichkeiten eines Arrays eingesetzt und durchlaufen werden, zum Beispiel auch über den `Array-Enumerator`.

Listing 4.10 chars3.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt";

    char[] chars = str.ToCharArray();

    foreach(char c in chars)
        Response.Write(c + "<br>");
}

</script>
```

4.6 ... ermitteln, um was für eine Art von Zeichen es sich handelt?

Mitunter ist es notwendig, die Art von eingegebenen Zeichen zu überprüfen. Ein einzelnes Zeichen wird über den Datentyp `char` repräsentiert. Dieser bietet diverse statische Methoden an, mit Hilfe derer sich beispielsweise abfragen lässt, ob es sich um eine Zahl, einen Klein- oder Großbuchstaben handelt.

Das Listing zeigt die Überprüfung einer Kreditkartennummer. Gültige Zeichen sind ausschließlich Zahlen sowie der Bindestrich. Zur Überprüfung wird die statische Methode `char.IsNumber` verwendet.

Listing 4.11 IsNumber1.aspx

```
<script runat="server">

void tb_validate(object sender, ServerValidateEventArgs e)
{
    e.IsValid = false;
    foreach(char c in e.Value)
    {
        if((!char.IsNumber(c)) && (c != '-'))
        {
            return;
            Response.Write(c);
        }
    }
    e.IsValid = true;
}

</script>

<form runat="server">

<p> Kreditkartennummer (Zahlen und Bindestriche):<br>
    <ASP:TextBox id="tb" runat="server"/>
    <ASP:Button runat="server" text="OK"/>
</p>

<ASP:CustomValidator runat="server"
    ControlToValidate="tb"
    OnServerValidate="tb_validate"
    text="Bitte geben Sie nur korrekte Zeichen an!"/>

</form>
```

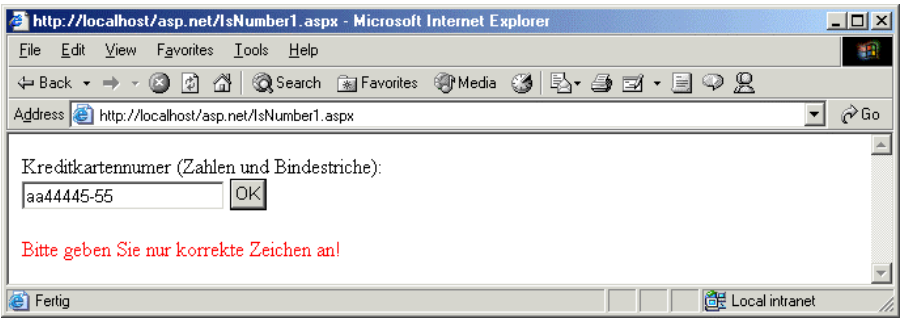


Abbildung 4.7 Es wurden ungültige Zeichen eingegeben.

Sicherlich ist dies nicht die eleganteste und effektivste Art und Weise, eine Kreditkartennummer zu validieren (vergleiche Kapitel „E-Commerce“), das Beispiel zeigt aber dennoch den Einsatz der genannten Methode. Die Struktur bietet derer weitere an; die Tabelle zeigt eine Übersicht.

Tabelle 4.1 Die statischen Methoden des Datentyps `char`

Methode	Überprüft auf ...
<code>IsControl</code>	Control-Zeichen (Umbruch, Tabulator etc.)
<code>IsDigit</code>	Ziffern
<code>IsLetter</code>	Buchstaben
<code>IsLetterOrDigit</code>	Buchstaben oder Ziffern
<code>IsLower</code>	Kleinbuchstaben
<code>IsNumber</code>	Zahlen (Ziffern oder Hex-Werte)
<code>IsPunctuation</code>	Satzzeichen
<code>IsSeparator</code>	Leerzeichen
<code>IsSurrogate</code>	Unicoce-Platzhalter
<code>IsSymbol</code>	Symbole (+, – etc.)
<code>IsUpper</code>	Großbuchstaben
<code>IsWhiteSpace</code>	WhiteSpaces (Leerzeichen, Umbruch etc.)

4.7 ... das Vorkommen eines Zeichens in einer Zeichenkette zählen?

Es ist mitunter notwendig, die Vorkommen eines Zeichens in einer Zeichenkette zu zählen. Mit Hilfe des Indexers der Zeichenkette ist eine Methode `CountChar` mit geringem Aufwand geschrieben.

Listing 4.12 CountChar1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt, dies ist ein toller Satz.";
    Response.Write(CountChar(str, 'i').ToString());
}

int CountChar(string str, char c)
{
    int count = 0;
    for(int i=0; i<str.Length; i++)
        if(str[i] == c) count++;
    return(count);
}

</script>
```

Das Ergebnis dieses Beispiels ist 3, denn das Zeichen „i“ ist drei Mal in der Zeichenkette enthalten. Doch wie sieht es mit Teilzeichenketten oder Wörtern aus. Wie lassen sich diese zählen? Hier kann die Methode `IndexOf` verwendet werden. Das zweite Beispiel liefert 2, denn die Teilzeichenkette „ll“ ist zwei Mal enthalten.

Listing 4.13 CountString1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt, dies ist ein toller Satz.";
    Response.Write(CountString(str, "ll").ToString());
}

int CountString(string str, string part)
{
    int count = 0;
    int pos = -1;

    do {
        pos = str.IndexOf(part, pos+1);
        if(pos != -1) count++;
    } while(pos != -1);

    return(count);
}

</script>
```


4.8 ... die Anzahl Wörter in einer Zeichenkette zählen?

Die vermutlich einfachste Variante, die Anzahl Wörter in einer Zeichenkette zu ermitteln, ist die des Zählens der Leerzeichen. Deren Anzahl plus eins ergibt die Anzahl der Wörter. Auf Basis des Listings aus dem Rezept „... das Vorkommen eines Zeichens in einer Zeichenkette zählen?“ ist die Abfrage schnell erledigt.

Listing 4.14 CountWords1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt, dies ist ein toller Satz.";
    int words = CountChar(str, ' ') + 1;
    Response.Write(words.ToString() + " Wörter");
}

int CountChar(string str, char c)
{
    int count = 0;
    for(int i=0; i<str.Length; i++)
        if(str[i] == c) count++;
    return(count);
}

</script>
```

Das Beispiel gibt die korrekte Anzahl von sieben Wörtern aus, hinterlässt aber dennoch einen unangenehmen Beigeschmack. Wirklich *schön* ist diese Abfrage nun wirklich nicht. Wesentlich eleganter und ein wenig kürzer ist da die Verwendung eines regulären Ausdrucks.

Listing 4.15 CountWords2.aspx

```
<% @Import Namespace="System.Text.RegularExpressions" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str = "Hallo Welt, dies ist ein toller Satz.";
    int words = CountWords(str);
    Response.Write(words.ToString() + " Wörter");
}

}
```

```
int CountWords(string str)
{
    Regex regex = new Regex(@"[\w]+");
    return(regex.Matches(str).Count);
}

</script>
```

4.9 ... eine Zeichenkette aus wiederholenden Zeichen zusammensetzen?

Oftmals soll eine Zeichenkette auf Basis der Wiederholung eines einzelnen Zeichens aufgebaut werden. Der Konstruktor der Klasse `string` bietet eine Überladung, bei der das gewünschte Zeichen sowie die Wiederholungsanzahl übergeben werden können. Für eine ganze Zeichenkette existiert eine analoge Möglichkeit nicht. Hier muss eine einfache eigene Methode herhalten. Das Listing zeigt beides.

Listing 4.16 RepeatString1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string str1 = new string('a', 10);
    Response.Write("<p>" + str1 + "</p>");

    string str2 = RepeatString("abc ", 10);
    Response.Write("<p>" + str2 + "</p>");
}

string RepeatString(string str, int count)
{
    string result = string.Empty;

    for(int i=0; i<count; i++)
        result += str;

    return(result);
}

</script>
```

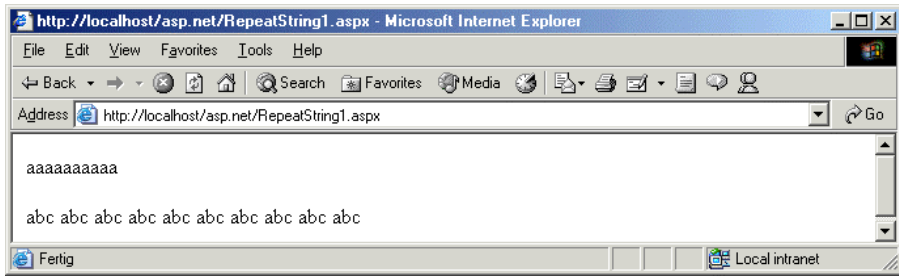


Abbildung 4.8 Das Zeichen beziehungsweise die Zeichenkette wird 10 Mal wiederholt.

4.10 ... eine Zeichenkette aufsplitten?

Oftmals enthält eine Zeichenkette eine Liste von wiederkehrenden Elementen. Diese können beispielsweise mit einem Komma oder einem Semikolon getrennt hinterlegt sein. Soll nun auf die einzelnen Elemente zugegriffen werden, könnte man die regulären Zeichenkettenoperationen wie `IndexOf` und `Substring` verwenden. Eleganter ist jedoch die Verwendung der Methode `Split`, die die Zeichenkette entsprechend einem übergebenen Separator aufteilt und in Form eines string-Arrays zurückliefert.

Listing 4.17 split1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string text = "Hallo Welt, dies ist ein ganz toller Satz!";
    string[] words = text.Split(' ');

    foreach(string word in words)
        Response.Write(word + "<br>");
}

</script>
```

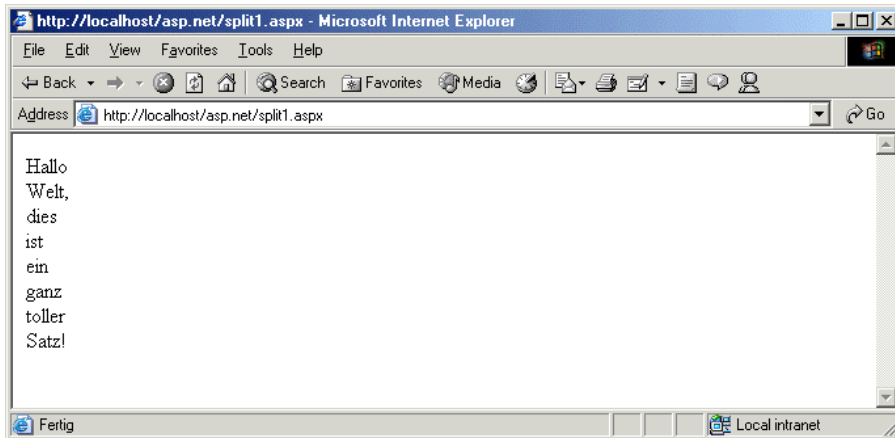


Abbildung 4.9 Die Zeichenkette wurde in Teilzeichenketten aufgesplittet.

Der Methode `Split` wird der zu suchende Separator als `char` übergeben. Sie können daher nur ein einzelnes Zeichen angeben und müssen dies in einfachen Anführungsstrichen ('x') notieren. Alternativ können Sie auch mehrere mögliche Trennzeichen als `char`-Array übergeben. Die Trennung erfolgt bei jedem dieser Zeichen.

Listing 4.18 `split2.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string text = "Hallo Welt, dies ist ein ganz toller Satz!";
    string[] words = text.Split(new char[] { ' ', ',', '.', '!' });

    foreach(string word in words)
        Response.Write(word + "<br>");
}

</script>
```

4.11 ... eine Zeichenkette zusammenführen?

Das vorherige Rezept hat gezeigt, wie Sie eine Zeichenkette in ein `string`-Array zerlegen lassen können. Umgekehrt können Sie die Methode `Join` verwenden, um Teile zu einer Gesamtzeichenkette zusammenzuführen. Neben dem `string`-Array wird der statischen Methode auch das zu verwendende Trennzeichen übergeben.

Listing 4.19 join1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string text1 = "Hallo Welt, dies ist ein ganz toller Satz!";
    string[] words = text1.Split(' ');

    string text2 = string.Join(" -STOP- ", words);

    Response.Write(text2);
}

</script>
```

Gibt es in Zeiten von Emails eigentlich noch Telegramme? Mein letztes habe ich zum zwölften Geburtstag bekommen, das ist schon eine Zeit lang her. Es sah ähnlich aus wie in der Abbildung.

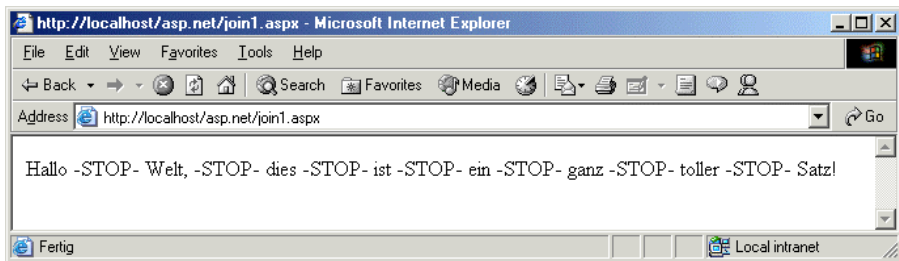


Abbildung 4.10 Die Zeichenkette wurde mittels Join zusammengefügt.

Selbstverständlich können Sie die Zeichenketten auch ohne Trennzeichen zusammenfügen. In diesem Fall übergeben Sie einfach `string.Empty` oder besser noch `null`.

Listing 4.20 join2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string text1 = "Hallo Welt, dies ist ein ganz toller Satz!";
    string[] words = text1.Split(' ');
    string text2 = string.Join(null, words);
    Response.Write(text2);
}

</script>
```

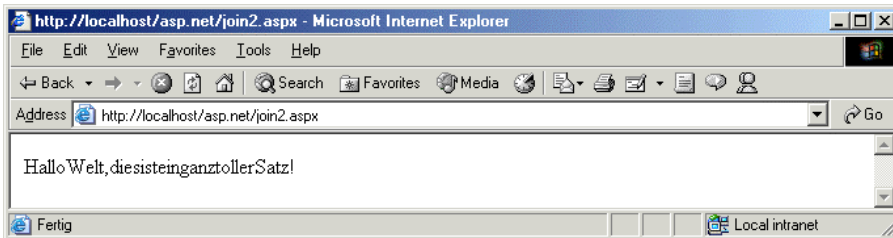


Abbildung 4.11 Auch eine Form von Kompression ...

4.12 ... eine Zahl in ein Wort umwandeln?

So langsam, aber sicher kommen Schecks leider ein wenig aus der Mode. Dennoch stehen Entwickler immer noch häufig vor der Aufgabe, eine gegebene Zahl in das entsprechende Wortpendant umzuwandeln. Im Internet bin ich über eine derartige Routine gestoßen, die ich kurzerhand auf ASP.NET beziehungsweise C# umgewandelt habe.

Das Listing zeigt die Routine im Einsatz. Die Zahl wird in ihre Bestandteile zerlegt und in die entsprechenden Wörter wie „drei“, „hundert“ und „zwanzig“ umgewandelt. Das Beispiel enthält ein Eingabefeld, so dass Sie die Umwandlung einfach ausprobieren können.

Listing 4.21 NumWords1.aspx

```
<script runat="server">

string[] ss = { "ein", "zwei", "drei", "vier", "fünf", "sechs", "sieben",
    "acht", "neun" };
string[] ds = { "zehn", "elf", "zwölf", "dreizehn", "vierzehn", "fünfzehn",
    "sechzehn", "siebzehn", "achtzehn", "neunzehn" };
string[] ts = { "zwanzig", "dreißig", "vierzig", "fünfzig", "sechzig",
    "siebzig", "achtzig", "neunzig" };
string[] qs = { "", "tausend", "million", "milliarde" };

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack) tb_changed(tb, EventArgs.Empty);
}

void tb_changed(object sender, EventArgs e)
{
    lb.Text = num2words(int.Parse(tb.Text));
}
```

```

string num2words(int iNum)
{
    bool b = false;
    string s = string.Empty;
    int i = iNum;

    if(i < 0)
    {
        b = true;
        i = i*-1;
    }
    else if(i == 0)
    {
        s = "null";
    }
    else
    {
        for(int j = 0; j<=3; j++)
        {
            int iii = i % 1000;
            i /= 1000;
            if(iii > 0) s = nnn2words(iii) + " " + qs[j] + " " + s;
        }
    }

    if(b) s = "minus " + s;
    return(s.Trim());
}

string nnn2words(int iNum)
{
    string s = string.Empty;
    int i = (iNum % 10);
    if(i > 0) s = ss[i-1];
    int ii = (iNum % 100)/10;

    if(ii == 1) s = ds[i];
    else if((ii>1) && (ii<10)) s = ts[ii-2] + " und " + s;

    i = (iNum / 100) % 10;
    if(i > 0) s = ss[i-1] + " hundert " + s;
    return(s);
}

</script>

<form runat="server">

```

```
<ASP:TextBox id="tb" runat="server" autopostback="true"
ontextchanged="tb_changed" text="12345"/>

<p> <ASP:Label id="lb" runat="server"/> </p>

</form>
```

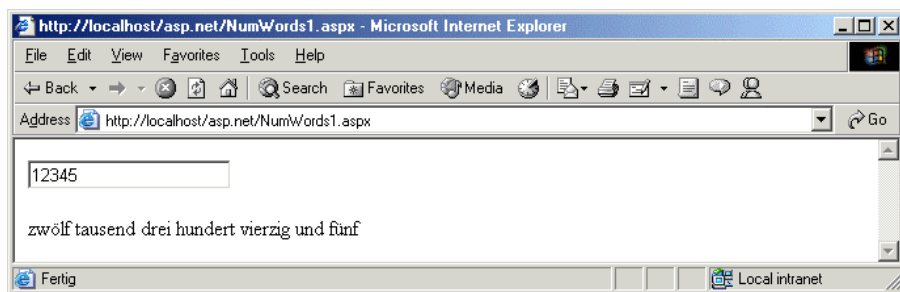


Abbildung 4.12 Die Zahl wurde in die entsprechenden Wörter umgewandelt.



Bitte beachten Sie, dass es sich um eine aus dem Englischen übersetzte Routine handelt. Das bedeutet, dass einige deutsche Eigenarten wie die gesprochene Reihenfolge bestimmter Wörter nicht optional berücksichtigt werden. Dennoch liefert das Listing brauchbare Ergebnisse und lässt sich somit gut in der Praxis verwenden.

4.13 ... reguläre Ausdrücke verwenden?

Hinweise zur Verwendung von regulären Ausdrücken zur Validierung von Benutzereingaben finden Sie im Rezept „... überprüfen, ob eine Eingabe einem individuellen Schema entspricht?“ im Kapitel „Eingabeformulare“.

4.14 ... Zeichen mit regulären Ausdrücken ersetzen?

Reguläre Ausdrücke lassen sich nicht nur zum Suchen von Elementen und zum Validieren von Benutzereingaben, sondern auch zum Ersetzen von Zeichenketten verwenden. Statt eines festen Suchworts kann dabei auf die bekannte Mustersuche zurückgegriffen werden. Zum Einsatz kommt die mehrfach überladene Methode `Replace` der Klasse `Regex` aus dem Namespace `System.Text.RegularExpressions`.

Das Listing zeigt die Verwendung der Methode. Der dem Konstruktor der Klasse übergebene reguläre Ausdruck entspricht dem Aufbau einer Email-Adresse. Dieses Muster wird genutzt, um mit Hilfe der Methode `Replace` alle vorkommenden Adressen durch eine neue zu ersetzen.

Listing 4.22 Replace1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Regex regex = new Regex(@"[\w\.-]+\.[\w\.-]+\. *
    [\w\.-]{2,63}\.[a-zA-Z]{2,4}");

    string text1 = "Meine Email-Adresse lautet hallo@welt.de.<br>
    Sie können mich aber auch unter weiss@nicht.com erreichen.";
    string text2 = regex.Replace(text1, "feedback@asp-buch.de");

    Response.Write("<p>Vorher: " + text1 + "</p>");
    Response.Write("<p>Nachher: " + text2 + "</p>");
}

</script>
```



Abbildung 4.13 Die Email-Adressen wurden durch eine neue ersetzt.

Sie können die übergebenen Muster nicht nur durch statische Elemente ersetzen, sondern hier auch auf den Inhalt des gefundenen Eintrags zurückgreifen. Die Möglichkeiten entsprechen der bekannten Gruppierungssyntax. Ich habe das Beispiel so abgeändert, dass von der gefundenen Email-Adresse nur die Domain geändert wird. Der davor notierte Benutzername bleibt bestehen.

Listing 4.23 Replace2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Regex regex = new Regex(@"([\w\.-]+)@([\w\.-]+\.)*
    [\w\.-]{2,63}\.[a-zA-Z]{2,4}");

    string text1 = "Meine Email-Adresse lautet hallo@welt.de.<br>
    Sie können mich aber auch unter weiss@nicht.com erreichen.";
    string text2 = regex.Replace(text1, "$1asp-buch.de");

    Response.Write("<p>Vorher: " + text1 + "</p>");
    Response.Write("<p>Nachher: " + text2 + "</p>");
}

</script>
```

**Abbildung 4.14** Die Email-Adressen wurden nur teilweise geändert.

Das folgende Beispiel demonstriert das Vorgehen noch einmal mit einem etwas einfacheren Muster. Gesucht wird nach einer Zahl mit Nachkommastellen. Als Trennzeichen ist der *Punkt* angegeben. Durch die Klammerung kann auf die Zahlen vor und nach dem Punkt getrennt zugegriffen werden. Genau dies macht sich der Ersetzungsausdruck zunutze. Hier werden die beiden Teile der Fundstelle mit einem *Komma* getrennt.

Listing 4.24 Replace3.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string text1 = "Das Buch ist den Betrag von 49.90 wirklich wert.";
```

```

string text2 = Regex.Replace(text1, @"(\d+)\.(\d+)", "$1,$2");

Response.Write("Vorher: " + text1 + "<br>");
Response.Write("Nachher: " + text2 + "<br>");
}

</script>

```

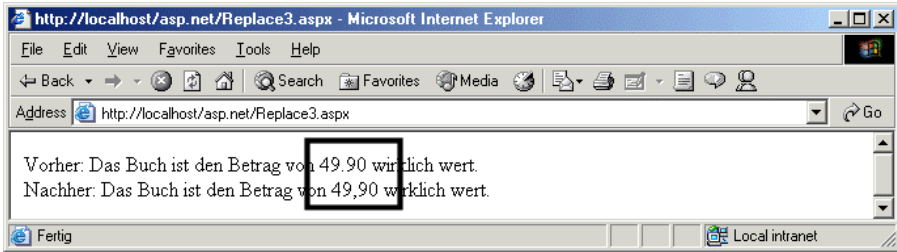


Abbildung 4.15 Die englische Schreibweise wurde in die deutsche umgewandelt.

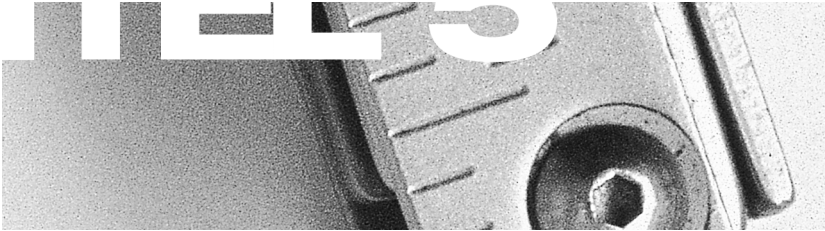


Die allermeisten regulären Ausdrücke enthalten Sonderzeichen, die von C# als Escape-Zeichen interpretiert werden. Sie sollten diesem vorbeugen, indem Sie vor alle derartigen Zeichenketten einen Klammeraffen setzen. C# interessiert sich dann nicht mehr für den Inhalt und interpretiert die Escape-Zeichen auch nicht mehr:

```
... @"(\d+)\.(\d+)", "$1,$2");
```

5

Mathematik und Berechnungen



Wie kann ich ...

5 Mathematik und Berechnungen

Nach Adam Riese zählt dieses Kapitel 20 Seiten. Auf diesen finden Sie zahlreiche Berechnungen. Diese entstammen der Mathematik, beschreiben aber auch den Umgang mit Dati. Auch hier gilt es zu rechnen.

5.1 ... die aktuelle Uhrzeit ermitteln?

Die Verwaltung einer Zeitinformation obliegt der Struktur `DateTime`. Deren statische Eigenschaft `Now` liefert eine solche Struktur mit dem Zeitpunkt bestehend aus Datum und Uhrzeit. Wollen Sie die Zeit direkt im Browserfenster ausgeben, verwenden Sie die Methode `ToString` und übergeben dieser optional Formatierungsanweisungen.

Listing 5.1 `Now1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DataBind();
}

</script>

<p> Die aktuelle Uhrzeit... </p>

<p>
    &bull; <%# DateTime.Now.ToString() %> <br>
    &bull; <%# DateTime.Now.ToString("d") %> <br>
    &bull; <%# DateTime.Now.ToString("D") %> <br>
    &bull; <%# DateTime.Now.ToString("f") %> <br>
    &bull; <%# DateTime.Now.ToString("F") %> <br>
    &bull; <%# DateTime.Now.ToString("g") %> <br>
    &bull; <%# DateTime.Now.ToString("G") %>
</p>
```

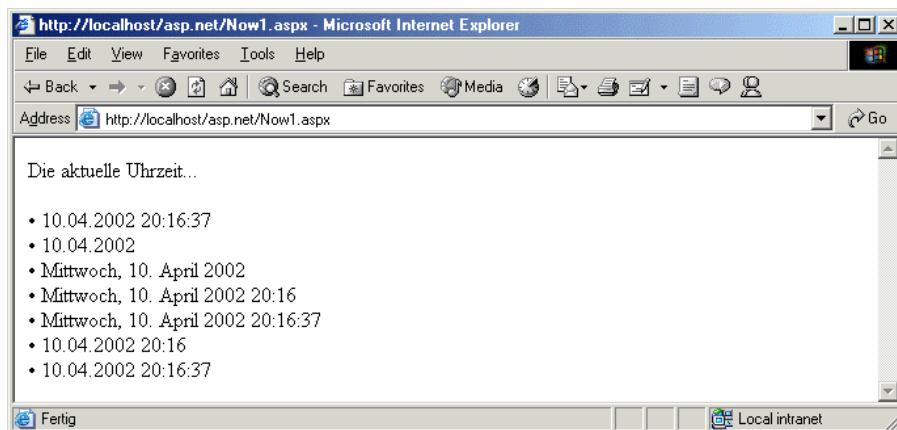


Abbildung 5.1 Entschuldigung, wie spät ist es bitte?

Die Formatierung der Ausgaben hängt von den aktuell gewählten Ländereinstellungen ab. Diese werden von der Klasse `CultureInfo` aus dem Namespace `System.Globalization` repräsentiert. Sollen die Ausgaben in einem landesspezifischen Format ausgegeben werden, können Sie über die `Locale-ID` eine spezifische Instanz der `Culture`-Klasse erstellen und diese als `Format-Provider` an die Methode `ToString` übergeben.

Das folgende Listing zeigt erneut die Ausgabe der aktuellen Uhrzeit. Über ein `DropDownList`-Control kann die gewünschte Landeseinstellung gewählt werden.

Listing 5.2 `Now2.aspx`

```
<script runat="server">

CultureInfo culture;

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        culture = CultureInfo.CurrentCulture;
        DataBind();
    }
}

void SelectedIndexChanged(object sender, EventArgs e)
{
    culture = CultureInfo.CreateSpecificCulture(list.SelectedItem.Value);
    DataBind();
}

</script>
```

```

<form runat="server">

    <p> Die aktuelle Uhrzeit ... </p>

    <p> Ländereinstellungen:
    <ASP:DropDownList id="list" runat="server"
        OnSelectedIndexChanged="SelectedIndexChanged"
        AutoPostBack="true">
        <ASP:ListItem Value="de" Text="Deutsch"/>
        <ASP:ListItem Value="en" Text="Englisch"/>
        <ASP:ListItem Value="fi" Text="Finnisch"/>
        <ASP:ListItem Value="fr" Text="Französisch"/>
        <ASP:ListItem Value="it" Text="Italienisch"/>
    </ASP:DropDownList>
    </p>

    ...

```

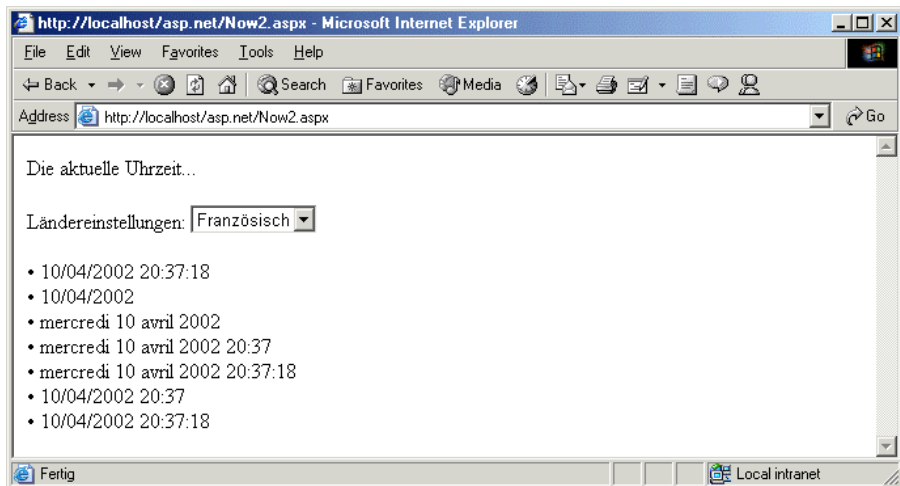


Abbildung 5.2 Vorbereitung für das Essen beim Franzosen – Froschschenkel am Mittwoch

Möchten Sie die Ländereinstellung für die gesamte Seite setzen, so können Sie die erstellte Instanz der Klasse `CultureInfo` der Eigenschaft `CurrentCulture` des aktiven Threads zuweisen. Alle nachfolgenden Formatierungen basieren anschließend auf der gewählten Einstellung:

```

Thread.CurrentThread.CurrentCulture =
    CultureInfo.CreateSpecificCulture(list.SelectedItem.Value);

```

5.2 ... eine tageszeitabhängige Begrüßung anzeigen?

Ein nettes Gimmick ist die tageszeitabhängige Begrüßung der Website-Besucher. Mit einem simplen if...else-Block lässt sich ein derartiger Gruß realisieren. Ausgewertet wird die Eigenschaft `Hour` der Struktur `DateTime`.

Listing 5.3 Welcome1.aspx

```
<script runat="server">

string welcome;

void Page_Load(object sender, EventArgs e)
{
    int hour = DateTime.Now.Hour;

    if(hour <= 10) welcome = "Guten Morgen";
    else if(hour <=17) welcome = "Guten Tag";
    else welcome = "Guten Abend";

    DataBind();
}

</script>

<h1> <%# welcome %> um <%# DateTime.Now.ToString("t") %>! </h1>
```

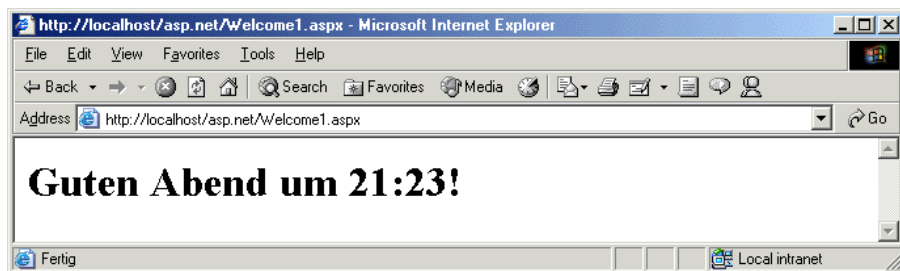


Abbildung 5.3 Zeit, den Rechner auszuschalten ...

5.3 ... den aktuellen Wochentag ermitteln?

Sie haben prinzipiell zwei Möglichkeiten, den aktuellen Wochentag auszugeben. Wollen Sie den Tagesnamen lediglich im Browser ausgeben, verwenden Sie die im

vorherigen Rezept gezeigte Methode `ToString` der Struktur `DateTime`. Auch hier können Sie selbstverständlich unterschiedliche landessprachliche Ausgaben erzeugen. Das Listing zeigt dies.

Listing 5.4 `DayOfWeek1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DataBind();
}

</script>

<p> Heutiger Wochentag: <br>
    <%# DateTime.Now.ToString("dddd") %> <br>

    <%# DateTime.Now.ToString("dddd",
        CultureInfo.CreateSpecificCulture("en")) %> <br>
    <%# DateTime.Now.ToString("dddd",
        CultureInfo.CreateSpecificCulture("fi")) %> <br>
    <%# DateTime.Now.ToString("dddd",
        CultureInfo.CreateSpecificCulture("fr")) %> <br>
    <%# DateTime.Now.ToString("dddd",
        CultureInfo.CreateSpecificCulture("it")) %>

</p>
```

Möchten Sie den Wochentag hingegen programmatisch weiterverarbeiten, liefert Ihnen die Eigenschaft `DayOfWeek` der Struktur `DateTime` einen Wert der gleichnamigen Enumeration `DayOfWeek`.

Listing 5.5 `DayOfWeek2.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    DayOfWeek dow = DateTime.Now.DayOfWeek;

    if(dow == DayOfWeek.Saturday || dow == DayOfWeek.Sunday)
        Response.Write("Hurraaaaa, Wochenende ...!");
    else
        Response.Write("Aufgestanden und ab zur Arbeit ...!");
}

</script>
```

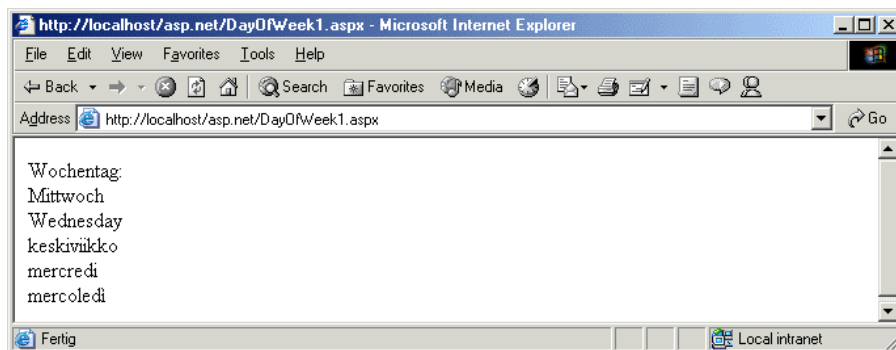


Abbildung 5.4 Der aktuelle Wochentag in verschiedenen Sprachen

5.4 ... die Kalenderwoche eines Tages ermitteln?

Die Kalenderwoche ist insbesondere im Geschäftsumfeld eine wichtige Einheit. Die Klasse `Calendar` aus dem Namespace `System.Globalization` ist nicht mit dem gleichnamigen Web Control zu vergleichen. Sie liefert eine abstrakte Basis für unterschiedliche Kalender, wie zum Beispiel den bei uns üblichen gregorianischen, und gibt uns eine ganze Reihe von kalenderspezifischen Methoden an die Hand.

Eine passende Instanz der Klasse `Calendar` kann über `CultureInfo.CurrentCulture.Calendar` abgefragt werden. Zurückgeliefert werden dürfte in aller Regel eine Instanz der abgeleiteten Klasse `GregorianCalendar`.

Über die Methode `GetWeekOfYear` lässt sich die Kalenderwoche eines übergebenen Datums abfragen. Dabei muss die zu verwendende Regel zur Berechnung der ersten Woche sowie der erste Wochentag übergeben werden. Letzteres ist bei uns der Montag. Die Woche Nummer eins ist hingegen die erste Woche im Januar, die mindestens vier Tage umfasst. Ergo handelt es sich um die Woche, die den vierten Januar enthält. Ist der 01.01. beispielsweise ein Sonntag, so wird als Kalenderwoche die letzte Woche des Vorjahres angegeben.

Listing 5.6 WeekOfYear1.aspx

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

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        tb.Text = DateTime.Now.ToString("d");
    }
}
```

```

    tb_TextChanged(tb, EventArgs.Empty);
}
}

void tb_TextChanged(object sender, EventArgs e)
{
    DateTime dt = DateTime.Parse(tb.Text);
    System.Globalization.Calendar calendar = Culture-
Info.CurrentCulture.Calendar;
    int weekofyear = calendar.GetWeekOfYear(dt, CalendarWeek-
Rule.FirstFourDayWeek, DayOfWeek.Monday);
    lb.Text = weekofyear.ToString();
}

</script>

<form runat="server">

    <p> Datum:
        <ASP:TextBox id="tb" runat="server"
            AutoPostBack="true"
            OnTextChanged="tb_TextChanged"/>
    </p>

    <p> Kalenderwoche: <ASP:Label id="lb" runat="server"/> </p>

</form>

```

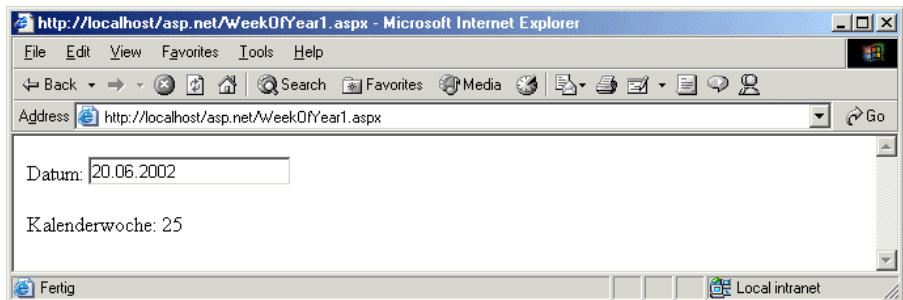


Abbildung 5.5 Diese ist die 25. Woche des Jahres.

Bedauerlicherweise arbeitet die Methode nicht in allen Fällen korrekt. So wird beispielsweise für den 31.12.2002 die Kalenderwoche 53 zurückgeliefert. Korrekt müsste es sich jedoch um die erste Woche 2003 handeln, da Silvester auf einen Dienstag fällt und somit der 04.01.2003 in derselben Woche liegt. Ganz offensichtlich haben die Amerikaner im Betatest nur das bei ihnen vorherrschende System geprüft.

Das Thema wurde in den deutschen .NET-Newsgroups ausführlich diskutiert. Herausgekommen ist eine individuelle Berechnung, die durch die nachfolgend gezeigte Struktur abgebildet wird.

```
public struct WeekOfYear
{
    public int Year;
    public int Week;

    public WeekOfYear(int year, int week)
    {
        Year = year;
        Week = week;
    }

    public static WeekOfYear FromDateTime(DateTime date)
    {
        DateTime dt_this = GetFirstDayOfFirstWeek(date.Year);
        DateTime dt_next = GetFirstDayOfFirstWeek(date.Year + 1);

        if(date > dt_next) return(new WeekOfYear(date.Year + 1, 1));
        else return(new WeekOfYear(date.Year, (date - dt_this).Days /
            7 + 1));
    }

    private static DateTime GetFirstDayOfFirstWeek(int year)
    {
        DateTime dt_jan04 = new DateTime(year, 1, 4);
        int weekday = (((int) dt_jan04.DayOfWeek) + 6) % 7 + 1;
        DateTime dt_fdfw = dt_jan04.AddDays(1 - weekday);
        return(dt_fdfw);
    }

    public override string ToString()
    {
        return(string.Format("{0}/{1}", this.Week, this.Year));
    }
}
```

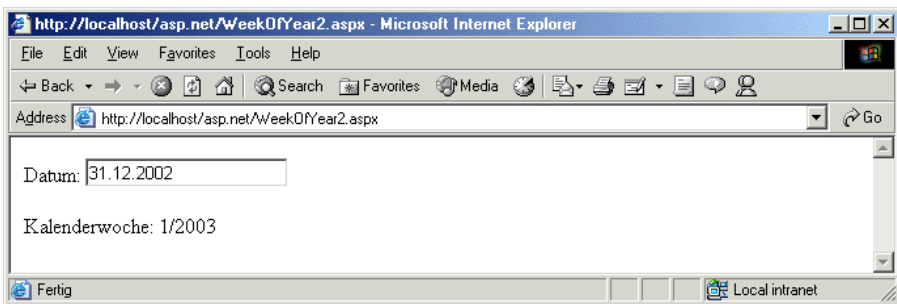
Die Berechnung erfolgt hier ausgehend vom vierten Januar des Jahres. Zurückgeliefert wird statt eines numerischen Werts immer eine Struktur. Diese enthält zwei Mitglieder für Kalenderwoche und das Jahr. Angewandt auf das zuvor vorgestellte Beispiel zeigt die Abbildung nun auch für den letzten Tag des Jahres die korrekte Kalenderwoche an. Diese liegt bereits im kommenden Jahr.

Listing 5.7 WeekOfYear2.aspx

```
...

void tb_TextChanged(object sender, EventArgs e)
{
    DateTime dt = DateTime.Parse(tb.Text);
    WeekOfYear wof = WeekOfYear.FromDateTime(dt);
    lb.Text = wof.ToString();
}

...
```

**Abbildung 5.6** Der letzte Tag des Jahres liegt in der Kalenderwoche 1 des nächsten Jahres.

5.5 ... die Anzahl der Tage eines Monats abfragen?

Auch die Anzahl der Tage eines gegebenen Monats sind mitunter interessant. Hier hilft die statische Methode `DateTime.DaysInMonth` weiter, der das gewünschte Jahr sowie Monat übergeben werden.

Listing 5.8 DaysInMonth1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        tb.Text = DateTime.Now.ToString("d");
        tb_TextChanged(tb, EventArgs.Empty);
    }
}

void tb_TextChanged(object sender, EventArgs e)
```

```
{
    DateTime dt = DateTime.Parse(tb.Text);
    int dim = DateTime.DaysInMonth(dt.Year, dt.Month);
    lb.Text = dim.ToString();
}

</script>

<form runat="server">

    <p> Datum:
    <ASP:TextBox id="tb" runat="server"
        AutoPostBack="true"
        OnTextChanged="tb_TextChanged" />
    </p>

    <p> Tage im Monat: <ASP:Label id="lb" runat="server" /> </p>

</form>
```

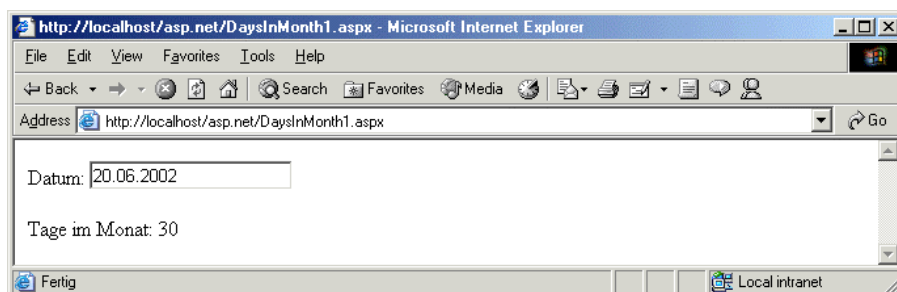


Abbildung 5.7 Der Monat umfaßt dreißig Tage.

5.6 ... ermitteln, ob es sich um ein Schaltjahr handelt?

Was Sie schon immer wissen wollten, aber nie zu fragen gewagt ... Folge #541. In dieser Ausgabe: „Wann ist das nächste Schaltjahr?“ Na ja, es ist wohl nicht die Frage aller Fragen, aber manchmal ist es durchaus nützlich zu wissen, ob es sich um ein Schaltjahr handelt. Neben anderen Klassen stellt auch der Datentyp `DateTime` eine statische Methode `IsLeapYear` zur Verfügung. Der boolesche Rückgabewert informiert, ob das als `int`-Wert übergebene Jahr ein Schaltjahr ist.

Listing 5.9 IsLeapYear1.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        for(int i=0; i<10; i++)
        {
            int year = DateTime.Now.Year + i;
            ddl.Items.Add(year.ToString());
        }
        ddl.SelectedIndex = 0;
        ddl_SelectedIndexChanged(ddl, EventArgs.Empty);
    }
}

void ddl_SelectedIndexChanged(object sender, EventArgs e)
{
    int year = int.Parse(ddl.SelectedItem.Value);
    lb.Text = (DateTime.IsLeapYear(year) ? "ja" : "nein");
}

</script>

<form runat="server">

    <p> Jahr:
        <ASP:DropDownList id="ddl" runat="server"
            AutoPostBack="true"
            OnSelectedIndexChanged="ddl_SelectedIndexChanged"/>
    </p>

    <p> Schaltjahr? <ASP:Label id="lb" runat="server"/> </p>

</form>

```

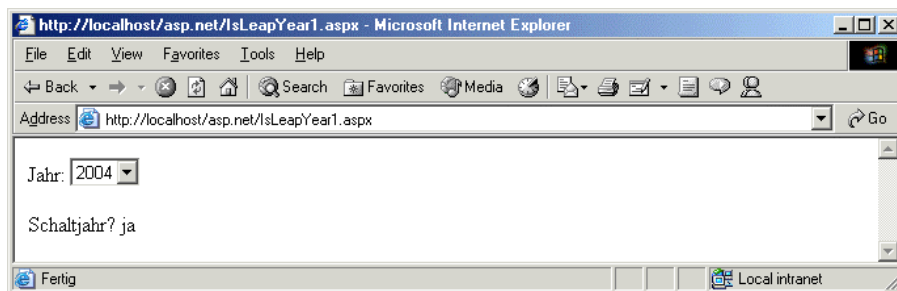


Abbildung 5.8 Die Antwort ...

5.7 ... das Datum der letzten Änderung ausgeben?

Gerade bei häufig aktualisierten und frequentierten Internet-Angeboten ist es für den Besucher mitunter interessant, den Zeitpunkt der letzten Aktualisierung der angezeigten Seite zu erfahren. Viele Websites listen dieses Datum im Fußbereich der Seite auf. Über die Möglichkeiten der Struktur `DateTime` und der `File`-Klasse ein Kinderspiel.

Listing 5.10 LastModified1.aspx

```
<script runat="server">

DateTime GetLastModified()
{
    string file = Server.MapPath(Request.Path);
    return(File.GetLastWriteTime(file));
}

</script>

<h1> Irgendeine Seite </h1>

<p> Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext </p>

<hr/>

<p style="font-size:8pt;"> Seite zuletzt geändert am
<%= GetLastModified() %> </p>
```

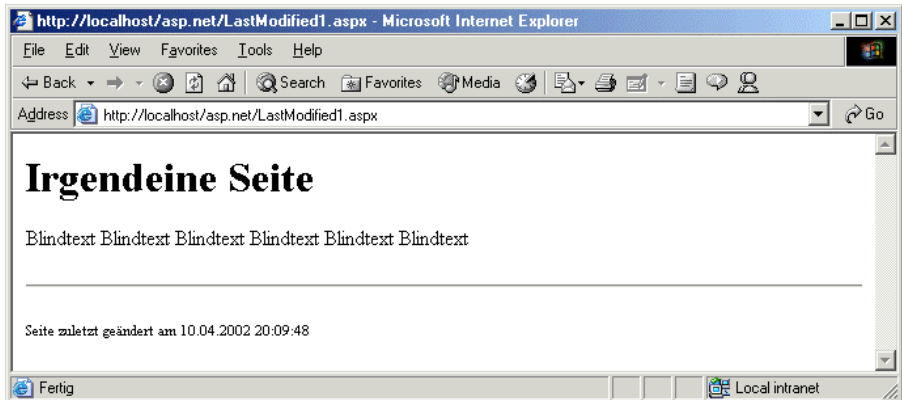



Abbildung 5.9 Die Seite ist auf dem aktuellen Stand.



Die gezeigte Ausgabe hat einen entscheidenden Haken. Gerade bei ASP.NET wird der Inhalt vieler Seiten ausschließlich durch eine unterliegende Datenbank bestimmt. Das angegebene Datum bezieht sich ausschließlich auf die Quellcode-Datei und kann Änderungen an der Datenquelle nicht berücksichtigen.

5.8 ... Fahrenheit in Celsius umrechnen und umgekehrt?

Sie meinen, dieses Rezept ist hier falsch? Ich finde nicht, denn eine derartige Umrechnung kann durchaus öfter benötigt werden, als Sie denken. Beispielsweise bei der Implementierung eines Wetter-Web Services wie im Rezept „... einen Web Service im UDDI-Verzeichnis finden?“ (Kapitel „XML Web Services“) beschrieben.

Die Umrechnung von Fahrenheit nach Celsius erfolgt nach einem einfachen Rechenschema:

$$T_c = (5/9) * (T_f - 32)$$

Umgekehrt geht es genauso einfach:

$$T_f = ((9/5) * T_c) + 32$$

Zwei einfache Methoden übernehmen die Umrechnung in der folgenden ASP.NET-Seite. Diese erlaubt die Eingabe und Umrechnung von beliebigen Temperaturwerten. Per Mausklick lassen sich nun beliebige Temperaturwerte konvertieren.

Listing 5.11 CalcTemperature1.aspx

```
<script runat="server">

void c2f_click(object sender, EventArgs e)
{
    double Tc = double.Parse(celsius.Text);
    double Tf = CelsiusToFahrenheit(Tc);
    fahrenheit.Text = Tf.ToString("0.00");
}

void f2c_click(object sender, EventArgs e)
{
    double Tf = double.Parse(fahrenheit.Text);
    double Tc = FahrenheitToCelsius(Tf);
    celsius.Text = Tc.ToString("0.00");
}

double CelsiusToFahrenheit(double tc)
{
    return(((9/5F)*tc)+32);
}

double FahrenheitToCelsius(double tf)
{
    return((5/9F)*(tf-32));
}

</script>

<form runat="server">

<p> Celsius:
    <ASP:TextBox id="celsius" runat="server"/>
    <ASP:Button runat="server" onclick="c2f_click" text="Umrechnen"/>
</p>

<p> Fahrenheit:
    <ASP:TextBox id="fahrenheit" runat="server"/>
    <ASP:Button runat="server" onclick="f2c_click" text="Umrechnen"/>
</p>

</form>
```



Beachten Sie beim Rechnen mit Nachkommastellen die Angabe von „F“ hinter der Zahl. Die im Beispiel gezeigten Zahlen 5 beziehungsweise 9 werden als int-Wert dargestellt. Dieser verfügt nicht über Nachkomma-

stellen, und daher ist das Ergebnis von $5/9$ immer 0. Durch Angabe von „F“ bei einer der Zahlen wird diese als float-Wert interpretiert, und eine korrekte Berechnung der Nachkommastellen ist möglich.

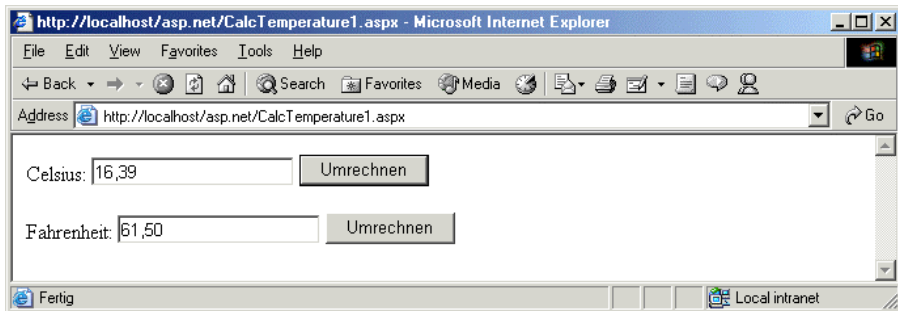


Abbildung 5.10 Die Umrechnung ist in beiden Richtungen möglich.

Nur der Ordnung halber möchte ich noch zwei weitere Temperaturskalierungen erwähnen. Kelvin basiert auf der Celsius-Angabe. 0° entspricht bei dieser Skalierung jedoch der absoluten 0-Temperatur, also der Temperatur, bei der jegliche molekulare Bewegung stoppt. Dieser Wert entspricht $-273,16^\circ \text{C}$. Um aus Celsius Kelvin zu errechnen, subtrahieren Sie daher einfach den Wert 273,16.

Die zweite Temperaturskalierung heißt Rankine und ist mit Kelvin vergleichbar. Auch hier sind keine Negativwerte vorgesehen, und 0 entspricht der absoluten 0-Temperatur. Die Berechnung erfolgt hier meist auf Basis von Fahrenheit, wobei der Wert 460 addiert werden muss.

5.9 ... eine Zahl runden?

Nichts einfacher als das! Verwenden Sie die statische Methode `Round` der Klasse `Math`.

Listing 5.12 Round1.aspx

```
void Page_Load(object sender, EventArgs e)
{
    Response.Write(string.Format("{0}<br>", Math.Round(4.4)));
    Response.Write(string.Format("{0}<br>", Math.Round(4.5)));
    Response.Write(string.Format("{0}<br>", Math.Round(4.6)));
}
```

Das Listing gibt die ganzzahligen, gerundeten Werte aus. In diesem Fall sind dies 4, 4 und 5. Mittels Überladungen der Methode können Sie jedoch auch die Anzahl der zu rundenden Nachkommastellen angeben.

Listing 5.13 Round2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write(string.Format("{0}<br>", Math.Round(4.44, 1)));
    Response.Write(string.Format("{0}<br>", Math.Round(4.55, 1)));
    Response.Write(string.Format("{0}<br>", Math.Round(4.66, 1)));
}

</script>
```

Im zweiten Listing werden alle Zahlen auf eine Nachkommastelle gerundet. Die Ausgabe im Browser ist in der Abbildung zu sehen.

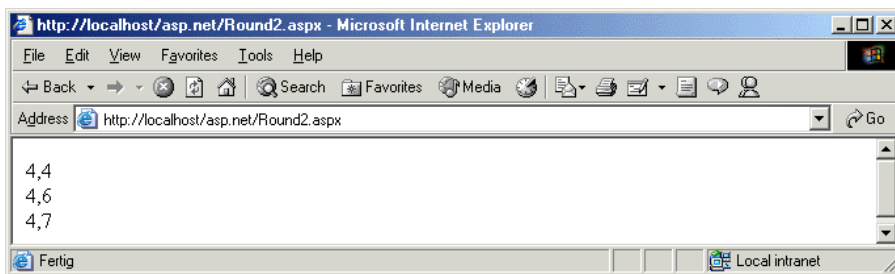


Abbildung 5.11 Die Zahlen wurden auf eine Nachkommastelle gerundet.

5.10 ... die nächste Ganzzahl ermitteln?

Die Klasse `Math` aus dem Mutter-Namespace `System` liefert statische Methoden für viele nützliche mathematische Operationen. Auch die Ermittlung der nächsten Ganzzahl ist hiermit möglich. Für die nächsthöhere Ganzzahl verwenden Sie die Methode `Ceiling`, für die nächst niedrigere `Floor`.

Listing 5.14 CeilingFloor1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
```

```
double value = 4.56565;
Response.Write("Wert: " + value.ToString() + "<br>");
Response.Write("Ceiling: " + Math.Ceiling(value).ToString() +
    "<br>");
Response.Write("Floor: " + Math.Floor(value).ToString() + "<br>");
}

</script>
```

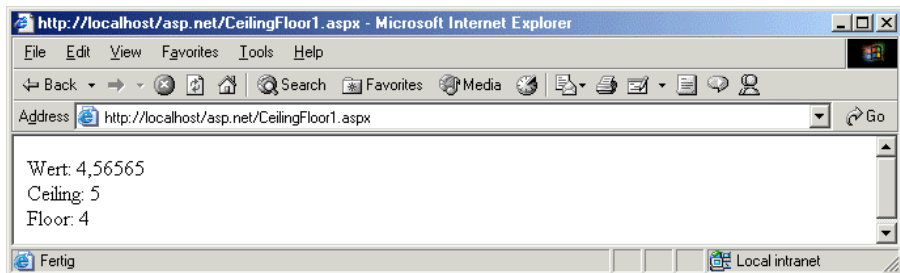


Abbildung 5.12 Die Methoden liefern die nächste Ganzzahl.

5.11 ... die kleinere/größere zweier Zahlen ermitteln?

Mitunter benötigen Sie die kleinere oder größere von zwei Zahlen. Statt hierfür jeweils eine if-Bedingung zu implementieren, können Sie einfach die statischen Methoden `Min` beziehungsweise `Max` der Klasse `Math` aus dem Namespace `System` verwenden. Beide Methoden kennen zahlreiche Überladungen für nahezu alle numerischen Datentypen einschließlich `byte`. Der Rückgabewert entspricht jeweils dem übergebenen Typ.

Listing 5.15 MinMax1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    int value1 = 5;
    int value2 = 10;
    Response.Write("Min: " + Math.Min(5, 10).ToString() + "<br>");
    Response.Write("Max: " + Math.Max(5, 10).ToString() + "<br>");
}

</script>
```

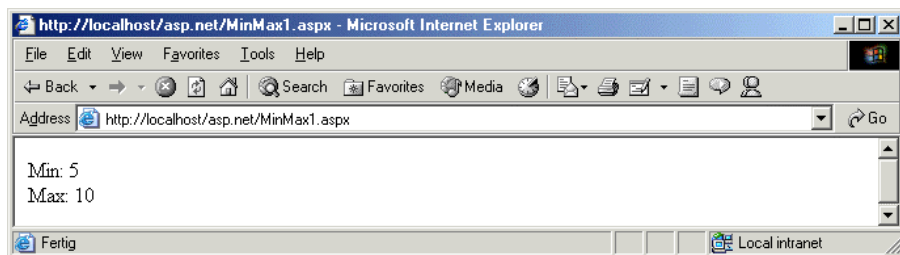


Abbildung 5.13 Min und Max liefern die kleinere beziehungsweise größere zweier Zahlen.

5.12 ... gerade/ungerade Zahlen ermitteln?

Man könnte annehmen, dass die Klasse `Math` aus dem Haupt-Namespace `System` zwei statische Methoden `Even` und `Odd` zur Ermittlung von geraden und ungeraden Zahlen anbieten würde. Doch hier haben die Entwickler wohl ein klein wenig geschlafen. Das macht nichts, denn mit einer kleinen Modulo-Division durch zwei lässt sich nachhelfen.

Listing 5.16 EvenOdd1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("Ist 5 gerade? " + Even(5).ToString() + "<br>");
    Response.Write("Ist 10 gerade? " + Even(10).ToString() + "<br>");
}

bool Even(int value)
{
    return(value % 2 == 0);
}

bool Odd(int value)
{
    return(value % 2 != 0);
}

</script>
```

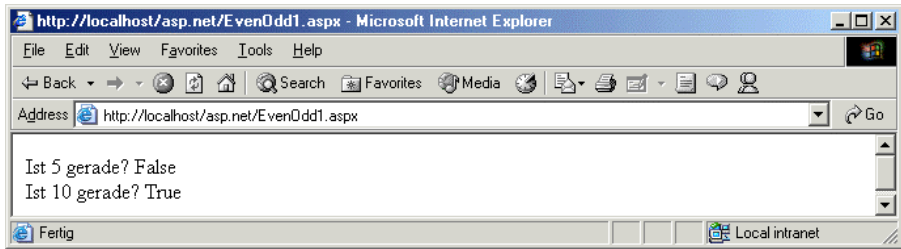
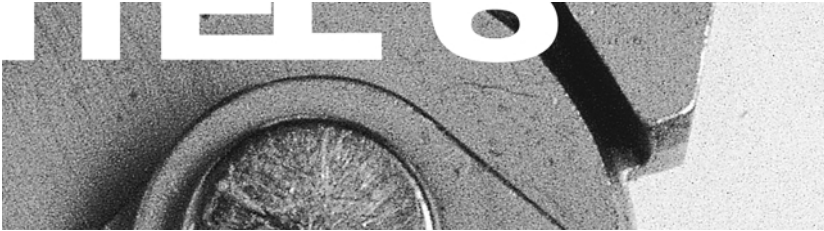


Abbildung 5.14 Die Methoden Even und Odd lassen sich ganz einfach nachbilden.

6

Eingabeformulare und Web Controls



Wie kann ich ...

6 Eingabeformulare und Web Controls

Dieses Kapitel übertrifft alle anderen dieses Buches zumindest im Hinblick auf den Umfang um Längen. Das ist kein Wunder, denn schließlich sind Eingabeformulare das A und O der Entwicklung von Web-Applikationen. Wie Sie die zahlreichen Web Controls sinnvoll in der Praxis einsetzen und das Äußerste aus ihnen heraus-holen, erfahren Sie hier.

6.1 ... die Reihenfolge von Eingabefeldern festlegen?

Alle Web Controls besitzen eine Eigenschaft `TabIndex`, der Sie einen fortlaufenden Wert zuweisen können. Mit Internet Explorer Version 4.0 und höher werden die Controls in der so angegebenen Reihenfolge angesprungen, wenn der Benutzer die Tabulatortaste betätigt.

Eine derartige Möglichkeit ist insbesondere immer dann nützlich, wenn die logische Reihenfolge nicht der optischen entspricht. Das nachfolgende Eingabeformular besitzt eine Tabelle mit mehreren Eingabefeldern. Im Regelfall würden diese von links nach rechts und oben nach unten angesprochen werden. Mit Hilfe der `TabIndex`-Eigenschaft erfolgt die Eingabe aber nun von oben nach unten und erst dann von links nach rechts.

Listing 6.1 `TabIndex1.aspx`

```
<form runat="server">

  <table>

    <tr>
      <td> Name: <ASP:TextBox runat="server" TabIndex="1"/> </td>
      <td> Telefon: <ASP:TextBox runat="server" TabIndex="5"/> </td>
    </tr>

    <tr>
      <td> Vorname: <ASP:TextBox runat="server" TabIndex="2"/> </td>
      <td> Telefax: <ASP:TextBox runat="server" TabIndex="6"/> </td>
```

```
</tr>

<tr>
  <td> Strasse: <ASP:TextBox runat="server" TabIndex="3"/> </td>
  <td> Email: <ASP:TextBox runat="server" TabIndex="7"/> </td>
</tr>

<tr>
  <td> PLZ & Ort: <ASP:TextBox runat="server" TabIndex="4"/> </td>
  <td> Internet: <ASP:TextBox runat="server" TabIndex="8"/> </td>
</tr>

</table>

</form>
```

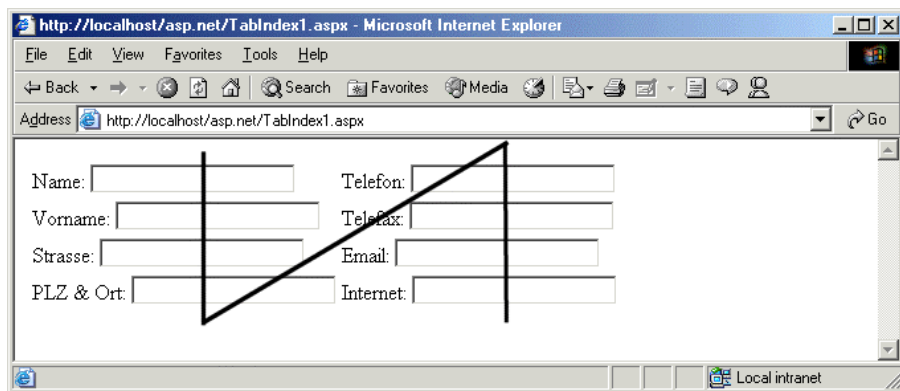


Abbildung 6.1 Die Linie kennzeichnet die Eingabereihenfolge.



Beachten Sie bitte, dass das erste Eingabefeld den TabIndex 1 und *nicht* 0 besitzen muss. 0 ist der initiale Standardwert der Eigenschaft und bedeutet, dass das zugehörige HTML-Attribut gar nicht erst ausgegeben wird.

6.2 ... eine Benutzereingabe erzwingen?

Das Validation Control `RequiredFieldValidator` bietet eine einfache Möglichkeit, eine Benutzereingabe zu erzwingen. Weisen Sie der Eigenschaft `ControlToValidate` lediglich das zu überprüfende Control zu.

Listing 6.2 RequiredFieldValidator1.aspx

```

<form runat="server">

    <p> Name:
        <ASP:TextBox id="name" runat="server"/>
        <ASP:Button runat="server" Text="OK"/>
        <ASP:Button runat="server" Text="Abbrechen"
            CausesValidation="false"/>
    </p>

    <ASP:RequiredFieldValidator runat="server"
        ControlToValidate="name">
        Bitte geben Sie einen Namen an!
    </ASP:RequiredFieldValidator>

</form>

```

Die verschiedenen Buttons lösen automatisch eine client- beziehungsweise serverseitige Validierung des Controls aus. Erst wenn diese erfolgreich war, wird das dahinter liegende Ereignis ausgelöst. Soll ein Button nicht in einer Validierung resultieren, so setzen Sie die Eigenschaft `CausesValidation` auf `false`, wie im Beispiel am Button „Abbrechen“ gezeigt.

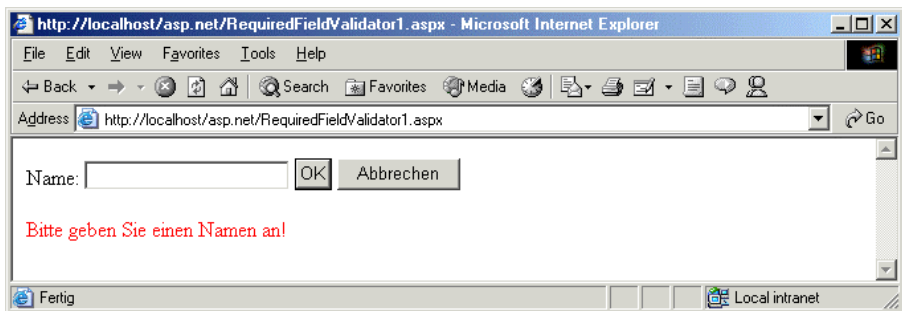


Abbildung 6.2 Der Benutzer muss seinen Namen eingeben.

6.3 ... überprüfen, ob ein numerischer Wert eingegeben wurde?

Verwenden Sie hierzu das Validation Control `CompareValidator`. Geben Sie als Operator `DataTypeCheck` und als Datentyp `Integer` an. Nun überprüft das Control automatisch, ob es sich bei der Benutzereingabe um einen numerischen

Wert handelt. Beachten Sie, dass Sie vor der Verwendung des Wertes diesen noch einmal manuell umwandeln müssen. Dies erledigen Sie mit Hilfe der Methode `int.Parse`.

Listing 6.3 CompareValidator1.aspx

```
<script runat="server">

void Button_Click(object sender, EventArgs e)
{
    int value = int.Parse(numvalue.Text);
    Response.Write(string.Format("Sie haben {0} eingegeben.", value));
}

</script>

<form runat="server">

    <p> Numerischer Wert:
        <ASP:TextBox id="numvalue" runat="server"/>
        <ASP:Button runat="server" Text="OK" OnClick="Button_Click"/>
    </p>

    <ASP:CompareValidator runat="server"
        ControlToValidate="numvalue"
        Operator="DataTypeCheck"
        Type="Integer">
        Bitte geben Sie einen numerischen Wert an!
    </ASP:CompareValidator>

</form>
```

Um sicherzustellen, dass der Benutzer nicht einfach eine leere Eingabe bestätigt und somit die Überprüfung unterläuft, sollten Sie zusätzlich ein `RequiredFieldValidator`-Control platzieren.

Listing 6.4 CompareValidator2.aspx

```
...
<ASP:RequiredFieldValidator runat="server"
    ControlToValidate="numvalue">
    Bitte geben Sie einen numerischen Wert an!
</ASP:RequiredFieldValidator>
...
```

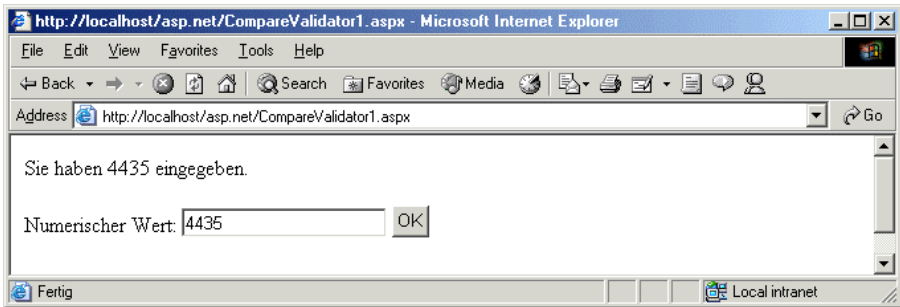


Abbildung 6.3 Das klappt nur, wenn ein numerischer Wert eingegeben wurde.



Möchten Sie eine gegebene Zeichenkette auf einen numerischen Wert hin überprüfen, können Sie unter Visual Basic .NET die Funktion `IsNumeric` verwenden. Im Rezept „... überprüfen, ob eine Zeichenkette einen numerischen Wert enthält?“ im Kapitel „Zeichenketten“ finden Sie eine analoge Möglichkeit für C#.

6.4 ... überprüfen, ob ein korrektes Datum eingegeben wurde?

Analog zu dem numerischen Wert aus dem vorherigen Rezept können Sie auch überprüfen, dass ein korrektes Datum angegeben wurde. Auch hier verwenden Sie das `CompareValidator`-Control und ändern lediglich den zu überprüfenden Datentyp auf `Date`.

Listing 6.5 `CompareValidator3.aspx`

```
<script runat="server">

void Button_Click(object sender, EventArgs e)
{
    DateTime value = DateTime.Parse(datevalue.Text);
    Response.Write(string.Format("Sie haben {0:D} eingegeben.", value));
}

</script>

<form runat="server">

<p> Datum:
    <ASP:TextBox id="datevalue" runat="server"/>
    <ASP:Button runat="server" Text="OK" OnClick="Button_Click"/>
</p>
```

```
<ASP:CompareValidator runat="server"
    ControlToValidate="datevalue"
    Operator="DataTypeCheck"
    Type="Date">
    Bitte geben Sie ein Datum an!
</ASP:CompareValidator>

</form>
```

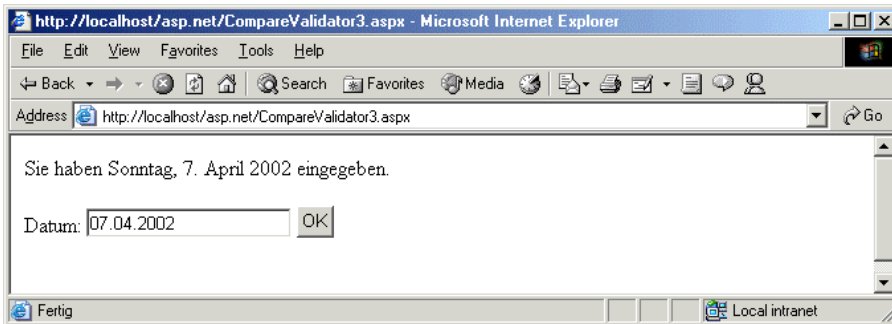


Abbildung 6.4 Nur korrekte Datumswerte sind erlaubt.

6.5 ... überprüfen, ob ein korrektes Datum in der Zukunft angegeben wurde?

Oftmals soll ein einzugebendes Datum in der Zukunft liegen, beispielsweise hat ein Liefertermin in der Vergangenheit wenig Sinn. Auch diese Überprüfung ist mit Hilfe des CompareValidator-Controls möglich.

Das zu vergleichende Datum muss hierbei dynamisch zum Beispiel im Page_Load-Ereignis zugewiesen werden. Da die Eigenschaft `ValueToCompare` des Validation Controls eine Zeichenkette erwartet, muss das Datum zunächst entsprechend konvertiert werden.

Listing 6.6 CompareValidator4.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    datevalue_compare.ValueToCompare = DateTime.Now.ToString("d");
}

void Button_Click(object sender, EventArgs e)
```

```

{
    DateTime value = DateTime.Parse(datevalue.Text);
    Response.Write(string.Format("Sie haben {0:D} eingegeben.", value));
}

</script>

<form runat="server">

    <p> Datum in der Zukunft:
        <ASP:TextBox id="datevalue" runat="server"/>
        <ASP:Button runat="server" Text="OK" OnClick="Button_Click"/>
    </p>

    <ASP:CompareValidator id="datevalue_compare" runat="server"
        ControlToValidate="datevalue"
        Operator="GreaterThan"
        Type="Date">
        Bitte geben Sie ein Datum in der Zukunft an!
    </ASP:CompareValidator>

</form>

```

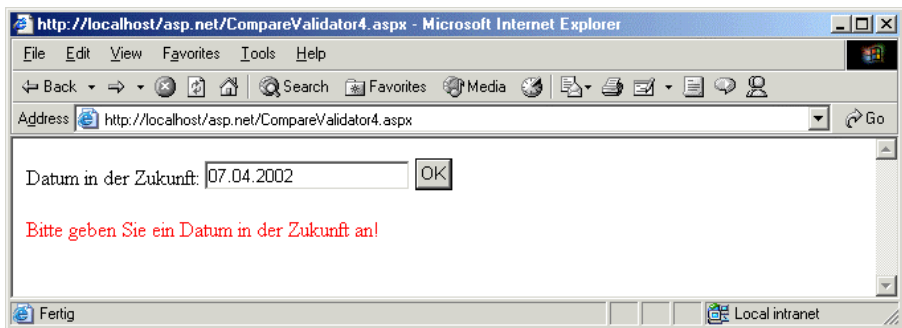


Abbildung 6.5 Das Datum liegt nicht in der Zukunft.

Wollen Sie den Datumswert zudem auch noch nach oben begrenzen, so verwenden Sie das Validation Control `RangeValidator`. Hier weisen Sie den Eigenschaften `MinimumValue` und `MaximumValue` die beiden Daten zu, zwischen denen sich die Eingabe bewegen muss.

Listing 6.7 RangeValidator5.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    datevalue_range.MinimumValue = DateTime.Now.AddDays(1).ToString("d");
    datevalue_range.MaximumValue =
        DateTime.Now.AddDays(14).ToString("d");
}

void Button_Click(object sender, EventArgs e)
{
    DateTime value = DateTime.Parse(datevalue.Text);
    Response.Write(string.Format("Sie haben {0:D} eingegeben.", value));
}

</script>

<form runat="server">

<p> Datum in der Zukunft:
    <ASP:TextBox id="datevalue" runat="server"/>
    <ASP:Button runat="server" Text="OK" OnClick="Button_Click"/>
</p>

<ASP:RangeValidator id="datevalue_range" runat="server"
    ControlToValidate="datevalue"
    Type="Date">
    Bitte geben Sie ein Datum in den nächsten 14 Tagen an!
</ASP:RangeValidator>

</form>
```

6.6 ... eine Email-Adresse syntaktisch überprüfen?

Es gibt unterschiedliche Ansätze, eine Email-Adresse syntaktisch zu prüfen. Die vermutlich effektivste stellt das RegularExpressionValidator-Control zur Verfügung. Diesem kann ein ausgefeilter Ausdruck übergeben werden, der den prinzipiellen Aufbau jeder Email-Adresse beschreibt.

Diese Formel sieht wie folgt aus:

```
^[\\w\\.\\-]+@[\\w\\.\\-]+\\.([\\w\\.\\-]{2,63})\\.([a-zA-Z]{2,4})$
```


Das Listing zeigt das vollständige Beispiel, das den Ausdruck verwendet, um eine eingegebene Email-Adresse zu validieren.

Listing 6.8 RegularExpressionValidator1.aspx

```
<form runat="server">

eMail-Adresse: <input type="text" id="txtEmail" runat="server"> <br>

<input type="submit" value="OK">

<ASP:RegularExpressionValidator id="valEmail" runat="server"
  ControlToValidate="txtEmail"
  ValidationExpression=
    "^[\w\.\-]+\@([\w\-]+\.)*\[\w\-\]{2,63}\. [a-zA-Z]{2,4}$">
  <p>Bitte geben Sie eine gültige Email-Adresse ein!</p>
</ASP:RegularExpressionValidator>

</form>
```

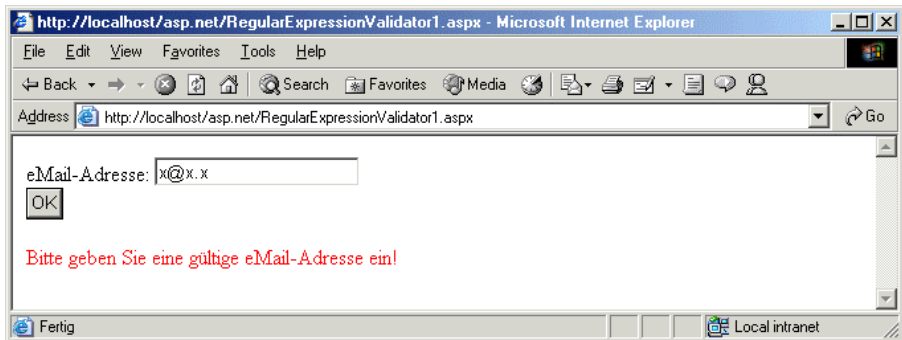


Abbildung 6.6 Schlecht gefälschte Email-Adressen werden direkt erkannt.

6.7 ... eine CheckBox validieren?

Ein CheckBox-Control zu validieren ist gar nicht so einfach, obwohl das durchaus ein recht häufiger Anwendungsfall ist. Denken Sie beispielsweise an die allgemeinen Geschäftsbedingungen, die vor dem Absenden einer Online-Bestellung explizit angehakt und so akzeptiert werden sollen. Leider sind die mitgelieferten Validation-Controls jedoch nicht in der Lage, dies zu überprüfen. Die Abbildung zeigt, was passiert, wenn man es doch versucht, hier in Verbindung mit einem CompareValidator-Control. Schade auch ...

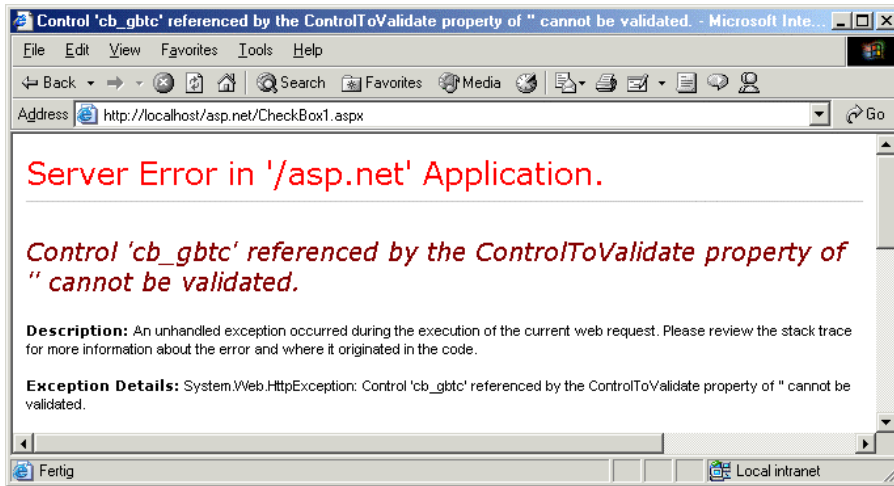


Abbildung 6.7 Das CompareValidator-Control kann die CheckBox nicht überprüfen.

Aber keine Sorge, der Heimwerker-König lebt! Dank der offenen Struktur lässt sich mit einigen Handgriffen ein eigenes Validation Control erstellen.

In einer separaten Quellcode-Datei wird dazu eine von `BaseValidator` abgeleitete Klasse mit dem Namen `CheckBoxValidator` angelegt. Damit das Control funktioniert, müssen zunächst die Methoden `ControlPropertiesValid` und `EvaluateIsValid` überschrieben werden. Die erste informiert, ob das zugewiesene Objekt validierbar ist, die zweite validiert es. Einer zusätzlichen Eigenschaft `ValueToCompare` kann zugewiesen werden, ob die CheckBox aktiviert oder deaktiviert werden muss, wobei der Standardwert auf der Hand liegt.

Listing 6.9 `CheckBoxValidator1.cs`

```
using System;
using System.Web.UI.WebControls;

namespace PAL.WebControls
{
    public class CheckBoxValidator : BaseValidator
    {
        protected const string vtc = "ValueToCompare";

        public bool ValueToCompare
        {
            get
            {
                if (ViewState[vtc] == null) return (true);
                return ((bool) ViewState[vtc]);
            }
        }
    }
}
```

```

    set { ViewState[vtc] = value; }
}

protected virtual CheckBox CheckBoxToValidate
{
    get
    {
        return(Parent.FindControl(this.ControlToValidate) as CheckBox);
    }
}

protected override bool ControlPropertiesValid()
{
    return(this.CheckBoxToValidate != null);
}

protected override bool EvaluateIsValid()
{
    CheckBox cb = this.CheckBoxToValidate;
    if(cb == null) return(false);
    return(cb.Checked == this.ValueToCompare);
}
}
}

```

Ist das Validation-Control erstellt, muss es mit dem Kommandozeilenprogramm `csc.exe` kompiliert und anschließend im bin-Verzeichnis der Web-Applikation abgelegt werden:

```
csc /t:library CheckBoxValidator1.cs
```

Jetzt geht es zur Sache, denn nun soll das Control beweisen, dass es auch wirklich funktioniert; das Beispiel der allgemeinen Geschäftsbedingung einer Online-Bestellung muss herhalten. Die Abbildung zeigt, ob's funktioniert.

Listing 6.10 CheckBox2.aspx

```

<% @Register TagPrefix="PAL" Namespace="PAL.WebControls"
    Assembly="CheckBoxValidator1" %>

<form runat="server">

    <ASP:CheckBox id="cb_gbtc" runat="server"
        text="Ich habe die AGB gelesen und erkenne diese an."/>

    <PAL:CheckBoxValidator runat="server"
        ControlToValidate="cb_gbtc"
        ErrorMessage="Sie müssen die AGB anerkennen, um fortzufahren"

```

```
text="***"/>

<ASP:ValidationSummary runat="server"/>

<p> <ASP:Button runat="server" text="Bestellen"/> </p>

</form>
```

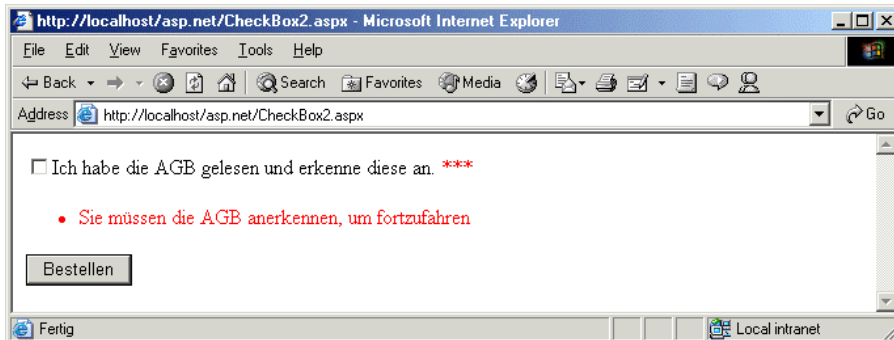


Abbildung 6.8 Das neue CheckBoxValidator-Control funktioniert ordnungsgemäß.

Soll stattdessen sichergestellt werden, dass die CheckBox nicht angehakt wird, setzen Sie einfach die Eigenschaft `ValueToCompare` auf `false`. Sinn hat das in diesem Zusammenhang allerdings nicht.

```
<PAL:CheckBoxValidator runat="server"
  ControlToValidate="cb_gbtc"
  ValueToCompare="false"
  ErrorMessage="Sie müssen die AGB anerkennen, um fortzufahren"
  text="***"/>
```

Wie Sie sehen, lässt sich mit recht einfachen Mitteln ein individuelles Validation-Control erstellen. Den Möglichkeiten sind dabei kaum Grenzen gesetzt, und so können Sie – wann immer noch nicht vorhanden und sinnvoll einsetzbar – weitere Überprüfungen flugs implementieren.

6.8 ... überprüfen, ob eine Eingabe einem individuellen Schema entspricht?

Mit Hilfe der Validation Controls lassen sich auch sehr individuelle Überprüfungen der Benutzereingaben durchführen. Zwei der insgesamt fünf Controls unterstützen Sie hierbei. Zum einen handelt es sich um den `RegularExpressionValidator` und zum anderen um den `CustomValidator`.

RegularExpressionValidator

Reguläre Ausdrücke sind eine sehr leistungsfähige Quasisprache zur Abbildung von Mustern. Im Unterschied zu einer regulären Sprache wird nicht der Weg, sondern das Ziel beschrieben. In dieser Hinsicht sind reguläre Ausdrücke sehr stark mit SQL verwandt. Bei der Validierung wird versucht, das vorgegebene Muster auf die Benutzereingabe abzubilden. Gelingt dies, ist die Eingabe korrekt, ansonsten nicht. Eine ausführliche Beschreibung der Möglichkeiten finden Sie im .NET Framework SDK oder auch im Buch „ASP.NET Grundlagen und Profiwissen“.

Die Definition des Validation Controls erfolgt in üblicher Weise. Der zu verwendende Ausdruck wird der Eigenschaft `ValidationExpression` zugewiesen. Das Listing zeigt die syntaktische Validierung einer Email-Adresse:

Listing 6.11 `RegularExpressionValidator1.aspx`

```
<form runat="server">

Email-Adresse: <input type="text" id="txtEmail" runat="server"> <br>

<input type="submit" value="OK">

<ASP:RegularExpressionValidator id="valEmail" runat="server"
  ControlToValidate="txtEmail"
  ValidationExpression=
    "^[\w\.\-]+\@([\w\-]+\.)*[\w\-]{2,63}\.[a-zA-Z]{2,4}$">
  <p>Bitte geben Sie eine gültige Email-Adresse ein!</p>
</ASP:RegularExpressionValidator>

</form>
```

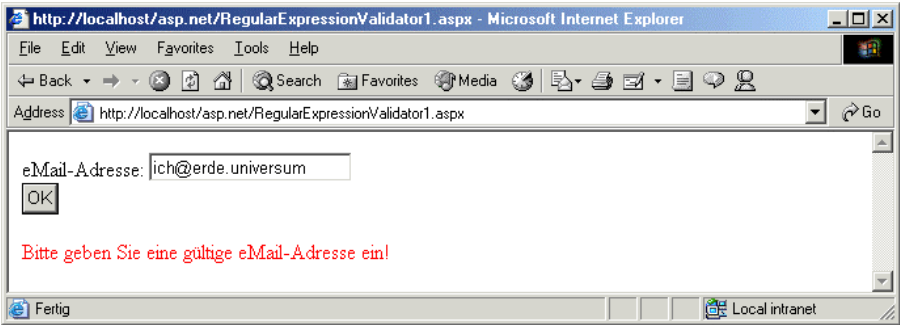


Abbildung 6.9 Die Email-Adresse wurde per regulärem Ausdruck überprüft.

Die Validierung erfolgt serverseitig mit Hilfe der Klasse `Regex` aus dem Namespace `System.Text.RegularExpressions`. Was dort intern passiert, lässt sich sehr einfach nachvollziehen. Das zweite Listing zeigt einen manuellen Nachbau der Validierung. Die Ausgabe dieses Beispiels ist dem vorherigen zum Verwechseln ähnlich.

Listing 6.12 `RegularExpressionValidator2.aspx`

```
<% @Import Namespace="System.Text.RegularExpressions" %>

<script runat="server">

void ok_click(object sender, EventArgs e)
{
    Regex regex = new Regex(@"^[\\w\\.\\-]+@[\\w\\-]+\\.\\s*"
        [\\w\\-]{2,63}\\.[a-zA-Z]{2,4}$");

    if(!regex.IsMatch(txtEmail.Text))
        msg.Text = "Bitte geben Sie eine gültige Email-Adresse ein!";
    else
        msg.Text = string.Empty;
}

</script>

<form runat="server">

eMail-Adresse: <ASP:TextBox id="txtEmail" runat="server"/> <br>

<ASP:Button runat="server" text="OK" onclick="ok_click"/>

<p> <ASP:Label id="msg" runat="server"/> </p>

</form>
```

Wenn Sie das erste Listing noch einmal ausprobieren, werden Sie feststellen, dass die Validierung bereits beim Verlassen des Eingabefeldes ausgeführt wird. Ein PostBack zum Server ist nicht notwendig. Dies erklärt sich durch die Tatsache, dass bereits eine clientseitige Validierung mit Hilfe von JavaScript durchgeführt wird. Hierbei kommt der gleiche reguläre Ausdruck zum Einsatz. Da es bei der Implementierung dieser jedoch Unterschiede zwischen .NET und JavaScript gibt, müssen Sie unbedingt darauf achten, dass der von Ihnen gewählte Ausdruck kompatibel ist. Ist dies nicht gegeben, sollten Sie die clientseitige Validierung über die Eigenschaft `EnableClientScript` deaktivieren.

Listing 6.13 `RegularExpressionValidator3.aspx`

```
<form runat="server">

Email-Adresse: <input type="text" id="txtEmail" runat="server"> <br>

<input type="submit" value="OK">

<ASP:RegularExpressionValidator id="valEmail" runat="server"
  EnableClientScript="false"
  ControlToValidate="txtEmail"
  ValidationExpression=
    "^[\w\.\-]+\@([\w\-]+\.)*[\w\-]{2,63}\.[a-zA-Z]{2,4}$">
  <p>Bitte geben Sie eine gültige Email-Adresse ein!</p>
</ASP:RegularExpressionValidator>

</form>
```

Damit Sie das Rad nicht bei jeder Validierung neu erfinden müssen, enthält die folgende Tabelle einige häufig verwendete Ausdrücke.

Tabelle 6.1 Wichtige reguläre Ausdrücke zur Eingabevalidierung

Beschreibung	Ausdruck
Email	<code>^[\w\.\-]+\@([\w\-]+\.)*[\w\-]{2,63}\.[a-zA-Z]{2,4}\$</code>
Internet-Adresse	<code>http://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?</code>
Telefonnummer	<code>((\((0\d\d\d)\) (\(0\d{3}\))?\d)?\d\d \d\d \d\d (\(0\d{4}\) \d \d\d\d-\d\d\d?)</code>
Deutsche PLZ	<code>(D-)?\d{5}</code>
Einfaches Passwort	<code>^(?=.*\d){4,8}\$</code>
Komplexes Passwort	<code>^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{4,8}\$</code>
Kreditkartennummer	<code>\d{4}-?\d{4}-?\d{4}-?\d{4}</code>

CustomValidator

Falls die Benutzereingaben einem komplizierten Schema unterliegen, dass mit Hilfe eines regulären Ausdrucks nicht überprüft werden kann, bietet sich das CustomValidator-Control an. Hier obliegt die Validierung ausschließlich Ihnen. Der Vorteil gegenüber einer expliziten Überprüfung beispielsweise innerhalb der Ereignisbehandlung eines Buttons liegt in der nahtlosen Integration in das Validierungskonzept von ASP.NET.

Das Control wirft zur Validierung ein Ereignis `ServerValidate`. Über die Ereignisargumente kann auf die Benutzereingabe zugegriffen und auch das Ergebnis der Validierung zurückgegeben werden. Im Beispiel sehen Sie die Überprüfung eines Passwortes. Dieses muss einem der hinterlegten Werte genügen.

Listing 6.14 CustomValidator1.aspx

```
<script runat="server">

void PW_Validate(object sender, ServerValidateEventArgs e)
{
    string[] pws = {"geheim", "secret", "pssst"};
    string pw = e.Value.ToLower();
    e.IsValid = (Array.IndexOf(pws, pw) > -1);
}

</script>

<form runat="server">

Passwort <input type="text" id="txtPW" runat="server"> <br>

<input type="submit" value="OK">

<ASP:CustomValidator runat="server"
    ControlToValidate="txtPW"
    OnServerValidate="PW_Validate">
    <p>Bitte geben Sie ein korrektes Passwort ein!</p>
</ASP:CustomValidator>

</form>
```

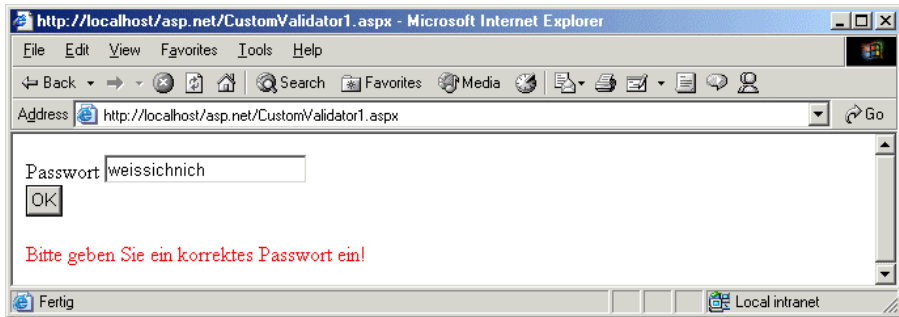



Abbildung 6.10 Das eingegebene Passwort ist nicht in der Liste enthalten.

Die gezeigte Validierung wird ausschließlich serverseitig durchgeführt. Sie können diese zusätzlich auch auf dem Client ausführen. Hierzu weisen Sie der Eigenschaft `ClientValidationFunction` den Namen der entsprechenden Funktion zu, die Sie der Seite als Script mitgeben müssen. Der Aufbau der Funktion entspricht dem serverseitigen Ereignis. Wenngleich theoretisch auch andere Scriptsprachen möglich wären, wird vornehmlich JavaScript eingesetzt, da dieses von der Mehrzahl der Browser unterstützt wird. Das zweite Beispiel zeigt eine derartige Validierung auf dem Client.

Listing 6.15 CustomValidator2.aspx

```
<script runat="server">

void PW_Validate(object sender, ServerValidateEventArgs e)
{
    string[] pws = {"geheim", "secret", "pssst"};
    string pw = e.Value.ToLower();
    e.IsValid = (Array.IndexOf(pws, pw) > -1);
}

</script>

<form runat="server">

Passwort <input type="text" id="txtPW" runat="server"> <br>

<input type="submit" value="OK">

<ASP:CustomValidator runat="server"
    ControlToValidate="txtPW"
    OnServerValidate="PW_Validate"
    ClientValidationFunction="PW_Validate_Client">
    <p>Bitte geben Sie ein korrektes Passwort ein!</p>
</ASP:CustomValidator>
```

```
</form>

<script language="jscript">
<!--

function PW_Validate_Client(sender, e)
{
    var pws = new Array("geheim", "secret", "pssst");
    var pw = e.Value.toLowerCase();
    e.IsValid = (ArrayIndexOf(pws, pw) > -1);
}

function ArrayIndexOf(array, value)
{
    for(i=0; i<array.length; i++)
    {
        if(array[i] == value) return(i);
    }
    return(-1);
}

// -->
</script>
```



Das gezeigte Beispiel resultiert in einer clientseitigen Validierung. Ist das Passwort falsch, so wird dies bereits vor dem PostBack erkannt. Dies bedeutet auch, dass die möglichen Passwörter an den Client übertragen werden müssen. Der Quellcode steht dort eins zu eins zur Verfügung, und die Daten können im Klartext ausgelesen werden. Sie sollten daher immer darauf achten, dass Sie mit einer clientseitigen Validierung nicht zu viele Informationen preisgeben.



Beachten Sie bitte auch, dass die clientseitige Validierung sehr einfach umgangen werden kann. Es reicht aus, wenn der verwendete Browser kein JavaScript unterstützt oder dieses deaktiviert wurde. Sie sollten daher *unbedingt* immer eine zusätzliche Überprüfung der Eingaben auf dem Server durchführen.

6.9 ... ein Eingabefeld schreibschützen?

Alle Web Controls bieten die Eigenschaft `Enabled` an. Setzen Sie diese auf `false`, so ist eine Eingabe nicht mehr möglich. Bei HTML Server Controls müssen Sie die Eigenschaft `Disabled` auf `true` setzen, um das analoge Ergebnis zu erzielen.

Listing 6.16 Enabled1.aspx

```

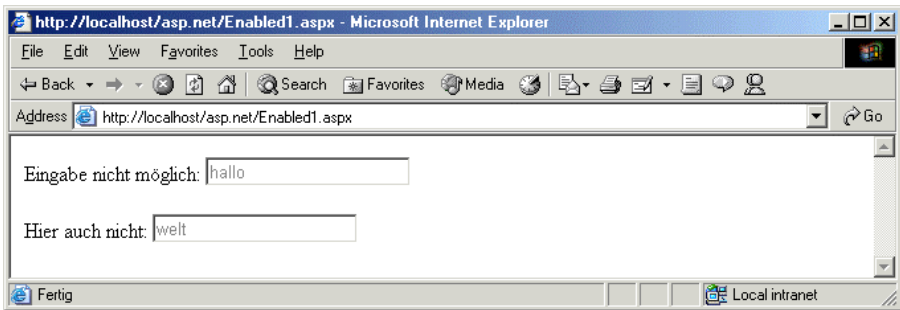
<form runat="server">

    <p> Eingabe nicht möglich:
      <ASP:TextBox runat="server" Enabled="false" text="hallo"/>
    </p>

    <p> Hier auch nicht:
      <input type="text" runat="server" Disabled="true" value="welt"/>
    </p>

</form>

```

**Abbildung 6.11** Der graue Text symbolisiert, dass eine Eingabe nicht möglich ist.**6.10 ... eine mehrstufige Auswahlliste realisieren?**

Mehrstufige oder hierarchische Auswahllisten sind im Internet sehr oft zu finden. Auf amerikanischen Websites werden sie verwendet, um nach der Landesauswahl zusätzlich den Bundesstaat anzugeben. Diese Auswahl soll aber natürlich nur verwendet werden, sofern als Land die USA selektiert wurden.

Hierzulande wird eine derartige Auswahl unter anderem im Marketingbereich verwendet. Ein Besucher wählt zunächst aus, über welche Art von Medium er von dem Angebot des Unternehmens erfahren hat. In einem zweiten Schritt kann er angeben, welches Medium ihn konkret zu einem Besuch veranlasst hat.

Das Listing zeigt die Erstellung einer solchen Auswahlliste. Grundlage ist dabei eine kleine Datenbank mit den zwei relational verknüpften Tabellen „media“ und „contacts“. Die erste Tabelle wird an das gleichnamige DropDownList-Control gebunden. Eine Benutzerauswahl resultiert dank AutoPostBack in einem serverseitigen Ereignis, in dessen Behandlung die zweite Liste gefüllt wird. Sind keine Einträge vorhanden, wird die zweite Auswahl komplett ausgeblendet.

Listing 6.17 ListBox_Hierarchie1.aspx

```
<script language="C#" runat=server>

string connstr = "Provider=Microsoft.Jet.OLEDB.4.0;" +
    @"Data Source=c:\inetpub\wwwroot\aspnet\contacts.mdb";

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new OleDbConnection(connstr);
    conn.Open();

    string SQL = "SELECT * FROM Media;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    media.DataSource =
        cmd.ExecuteReader(CommandBehavior.CloseConnection);

    if(!IsPostBack)
    {
        media.DataBind();
        media.Items.Insert(0, " - Bitte auswählen - ");
        contacts.Items.Add(" - Bitte zunächst Kontaktart auswählen - ");
    }
}

void media_Change(object sender, EventArgs e)
{
    contacts.Items.Clear();

    if(media.SelectedIndex == 0)
    {
        contacts.Items.Add(" - Bitte zunächst Kontaktart auswählen - ");
    }
    else
    {
        OleDbConnection conn = new OleDbConnection(connstr);
        conn.Open();

        string SQL = "SELECT * FROM Contacts WHERE MediaID=@MediaID;";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        cmd.Parameters.Add("@MediaID", media.SelectedIndex);
        contacts.DataSource =
            cmd.ExecuteReader(CommandBehavior.CloseConnection);
        contacts.DataBind();

        select2.Visible = (contacts.Items.Count != 0);
    }
}

</script>
```

```

<form runat="server">

<p> Bitte wählen Sie aus, wie Sie von uns erfahren haben.</p>

<p> Kontaktart:
  <ASP:DropDownList id="media" runat="server"
    DataTextField="Text"
    DataValueField="ID"
    AutoPostBack="true"
    OnSelectedIndexChanged="media_Change"
    Width="230"/>
</p>

<ASP:PlaceHolder id="select2" runat="server">
  <p> Kontakt durch:<br>
    <ASP:ListBox id="contacts" runat="server"
      DataTextField="Text"
      DataValueField="ID"
      Width="300"
      Height="80"/>
  </p>
</ASP:PlaceHolder>

</form>

```

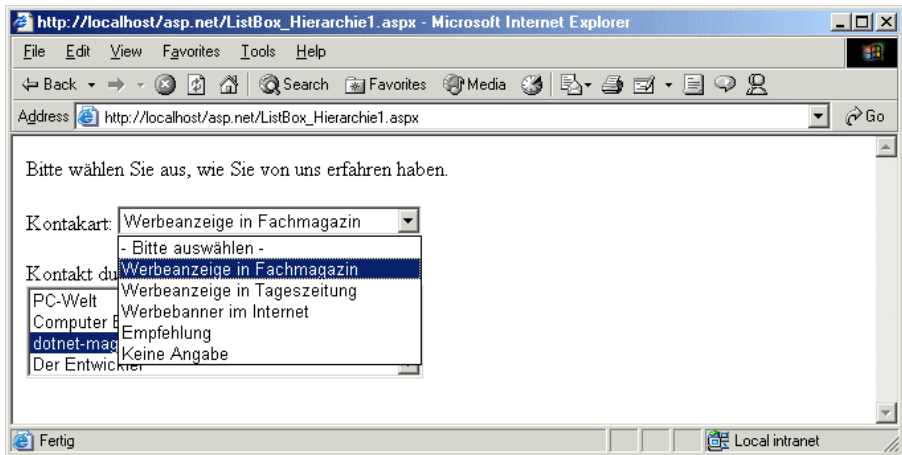


Abbildung 6.12 Mit wenigen Zeilen zu einer hierarchischen Auswahlliste

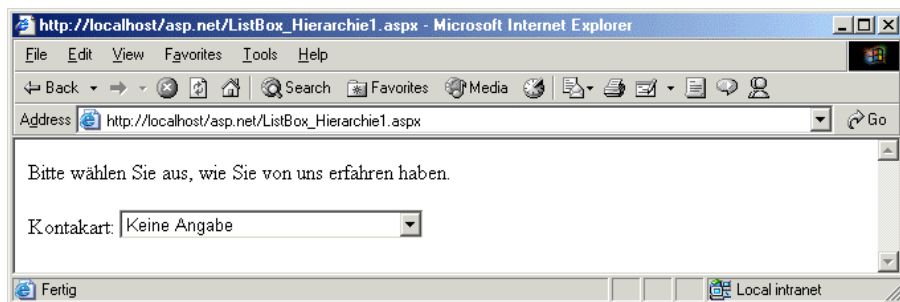


Abbildung 6.13 Ist keine zweite Auswahl notwendig, wird diese ausgeblendet.

Es bietet sich an, eine derartige Funktionalität in einem User Control zu kapseln. Auf diese Weise können Sie die Auswahlliste in unterschiedlichen Seiten immer wieder verwenden.

6.11 ... eine Liste von Ländern anzeigen?

Möchte man die Daten eines Besuchers erfragen, so stößt man zwangsläufig auf ein Feld zur Angabe des Landes. Es hat sich quasi als Standard etabliert, der manuellen Eingabe durch den Benutzer eine Auswahlliste vorzuziehen.

Die zur Anzeige notwendigen Daten liefert eine kleine Datenbank. Diese enthält neben der Landeskennung den deutschen sowie englischen Namen. Sie können die Datenbank somit auch auf lokalisierten Seiten wiederverwenden. Im Beispiel zu der Datenbank können Sie mittels zweier RadioButton-Controls zwischen deutschen und englischen Landesnamen wechseln.

Listing 6.18 Country_List1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new OleDbConnection(
        "Provider=Microsoft.Jet.OLEDB.4.0;" +
        @"Data Source=c:\inetpub\wwwroot\aspnet\countries.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Countries;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    countrylist.DataSource =
        cmd.ExecuteReader(CommandBehavior.CloseConnection);

    if(!IsPostBack) countrylist.DataBind();
}
```

```
void lang_CheckedChanged(object sender, EventArgs e)
{
    countrylist.DataTextField = (de.Checked ? "TextDE" : "TextEN");
    countrylist.DataBind();
}

</script>

<form runat="server">

<p> Sprache:
  <ASP:RadioButton id="de" GroupName="lang" runat="server"
    Text="Deutsch" OnCheckedChanged="lang_CheckedChanged"
    AutoPostBack="true"
    Checked="true"/>

  <ASP:RadioButton id="en" GroupName="lang" runat="server"
    Text="English" OnCheckedChanged="lang_CheckedChanged"
    AutoPostBack="true"/>
</p>

<p> Bitte wählen Sie Ihr Land aus:
  <ASP:DropDownList id="countrylist" runat="server"
    DataTextField="TextDE"
    DataValueField="ID"/>
</p>

</form>
```



Beachten Sie bitte, dass es unterschiedliche Landeskennungen gibt. Es gibt mehrere internationale Standards für zwei- und dreistellige Codes (zum Beispiel ISO 3166-1:1997) sowie davon abgeleitete deutsche und europäische Fassungen (zum Beispiel DIN EN ISO 3166-1:1998-04). Die in dieser Datenbank vorliegenden Kennungen entsprechen den von der Deutschen Post akzeptierten Daten und sind insofern in der Praxis verwendbar. Die anderen Kennzeichen erhalten Sie sehr einfach, wenn Sie in einer Suchmaschine die Nummern der Standards eingeben.

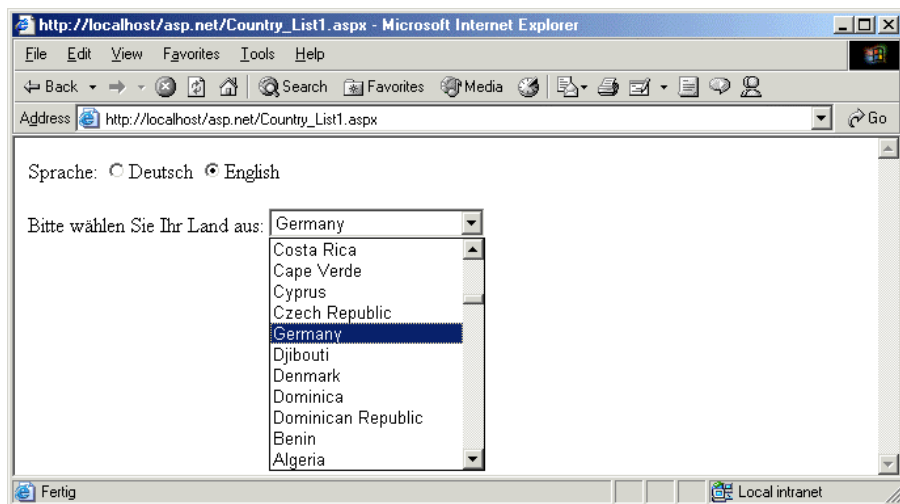


Abbildung 6.14 Die Liste erleichtert die Auswahl und bringt konsistente Daten.

6.12 ... ein einfaches Kontaktformular erstellen?

Eine Website dient sehr häufig als erstes Informationsmedium für Interessenten. Je mehr Informationen Sie übersichtlich bereitstellen, desto besser kann sich der potenzielle Kunde unverbindlich und anonym informieren. Doch bei sorgfältiger Zusammenstellung der Daten bleibt immer noch eine Frage offen, die der Interessent im Dialog mit dem Anbieter klären möchte.

Um die Anfrage dem Interessenten so einfach wie möglich zu machen und die Hemmschwelle der Kontaktaufnahme herunterzusetzen, sollte statt eines einfachen Links zur einer Email-Adresse ein übersichtliches Kontaktformular angeboten werden. Der Interessent gibt hier seine persönlichen Daten an und beschreibt kurz sein Anliegen. Intern wird daraus eine Email an den Anbieter erzeugt. Dieser kann dem Interessenten einfach per Email antworten.

Meiner Erfahrung nach sollte ein Kontaktformular sehr einfach aufgebaut sein. Es sollten möglichst wenige persönliche Daten abgefragt werden. Je einfacher und schneller das Ausfüllen des Formulars vonstatten geht, desto eher wird der Interessent diese Möglichkeit in Anspruch nehmen.

Das folgende Beispiel zeigt ein solches Kontaktformular. Es enthält vier Eingabefelder für Vorname, Name, Email-Adresse und die eigentliche Anfrage. Über mehrere Validation-Controls wird sichergestellt, dass alle obligatorischen Eingaben

gemacht werden. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben und ein kleines Symbol angezeigt. Sie können dies in der Abbildung erkennen. Die Implementierung entspricht den im Verlaufe des Kapitels gezeigten Rezepten.

Kontaktformular

Vielen Dank für Ihr Interesse! Sehr gerne beantworten wir Ihre Fragen. Bitte füllen Sie einfach die folgenden Felder aus. Mit * versehene Felder sind zur Bearbeitung Ihrer Anfrage obligatorisch.

Vorname:

Name*:

Email*:

Anfrage*:

- Bitte geben Sie Ihre Email-Adresse ein
- Bitte geben Sie Ihre Anfrage ein

Abbildung 6.15 Es wurden nicht alle benötigten Daten eingegeben.

Listing 6.19 contactform1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Assembly src="MailParser.cs" %>
<% @Import Namespace="System.Web.Mail" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    ph1.Visible = false;
    ph2.Visible = true;

    MailMessageExt mail = new
        MailMessageExt(Server.MapPath("contact.txt"), Request.Form);
    SmtMail.Send(mail);
}

</script>
```

```
<form runat="server">

<ASP:PlaceHolder id="ph1" runat="server">

<h1>Kontaktformular</h1>
<p> Vielen Dank für Ihr Interesse! Sehr gerne beantworten wir Ihre
  Fragen. Bitte füllen Sie einfach die folgenden Felder aus. Mit *
  versehene Felder sind zur Bearbeitung Ihrer Anfrage obligatorisch.
</p>

<table width="300">

  <tr>
    <td width="25%"> Vorname: </td>
    <td width="65%">
      <ASP:TextBox id="tb_firstname" runat="server" width="100%"/>
    </td>
    <td width="10%"> &nbsp; </td>
  </tr>

  <tr>
    <td width="25%"> Name*: </td>
    <td width="65%">
      <ASP:TextBox id="tb_lastname" runat="server" width="100%"/>
    </td>
    <td width="10%">
      <ASP:RequiredFieldValidator runat="server"
        ControlToValidate="tb_lastname"
        display="dynamic"
        ErrorMessage="Bitte geben Sie Ihren Namen ein">
        
      </ASP:RequiredFieldValidator>
    </td>
  </tr>

  <tr>
    <td width="25%"> Email*: </td>
    <td width="65%">
      <ASP:TextBox id="tb_email" runat="server" width="100%"/>
    </td>
    <td width="10%">
      <ASP:RequiredFieldValidator runat="server"
        ControlToValidate="tb_email"
        display="dynamic"
        ErrorMessage="Bitte geben Sie Ihre Email-Adresse ein">
        
      </ASP:RequiredFieldValidator>
    </td>
  </tr>

</table>

</form>
```

```

        <ASP:RegularExpressionValidator id="valEmail" runat="server"
        ControlToValidate="tb_email"
        ValidationExpression="^[\\w\\.\\-]+@([\\w\\.\\-]+\\.)*
        [\\w\\.\\-]{2,63}\\.[a-zA-Z]{2,4}$"
        display="dynamic"
        ErrorMessage="Bitte geben Sie eine korrekte Email-Adresse ein">
        
        </ASP:RegularExpressionValidator>
    </td>
</tr>

<tr>
    <td width="25%" valign="top"> Anfrage*: </td>
    <td width="65%"> <ASP:TextBox id="tb_text" runat="server"
        textmode="multiline" width="100%"/> </td>
    <td width="10%">
        <ASP:RequiredFieldValidator runat="server"
        ControlToValidate="tb_text"
        display="dynamic"
        ErrorMessage="Bitte geben Sie Ihre Anfrage ein">
        
        </ASP:RequiredFieldValidator>
    </td>
</tr>

</table>

<ASP:ValidationSummary runat="server"/>

<p> <ASP:Button runat="server" onclick="bt_click"
    text="Formular absenden"/> </p>

</ASP:Placeholder>

<ASP:Placeholder id="ph2" runat="server" visible="false">
    <h1> Vielen Dank! </h1>
    <p> Ihre Anfrage wurde weitergeleitet. Wir werden uns in Kürze mit
        Ihnen in Verbindung setzen. </p>
</ASP:Placeholder>

</form>

```

Das Web-Formular enthält zwei Placeholder-Controls. Diese sind dafür verantwortlich, dass nach dem Absenden des Formulars dieses ausgeblendet und stattdessen ein kurzes Dankeswort angezeigt wird. Sie sehen diese Umschaltung in der Abbildung. Selbstverständlich könnten Sie die Ausgabe à la „Vielen Dank, Herr ...“ personalisieren.

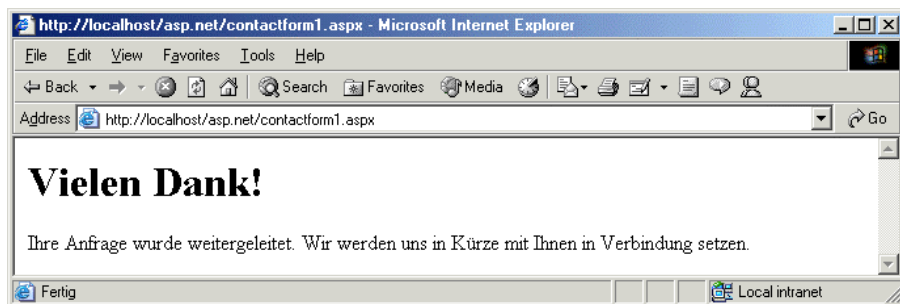


Abbildung 6.16 Die Anfrage wurde abgesendet.

Intern wurde die Anfrage des Interessenten als Email weitergeleitet. Dies ist notwendig, damit der Anbieter diese komfortabel im regulären Email-Programm beantworten kann. Hierzu wird ein Mail-Parser aus dem Rezept „... eine Email-Vorlage erstellen?“ verwendet. Das Ergebnis der Anfrage zeigt die folgende Abbildung.

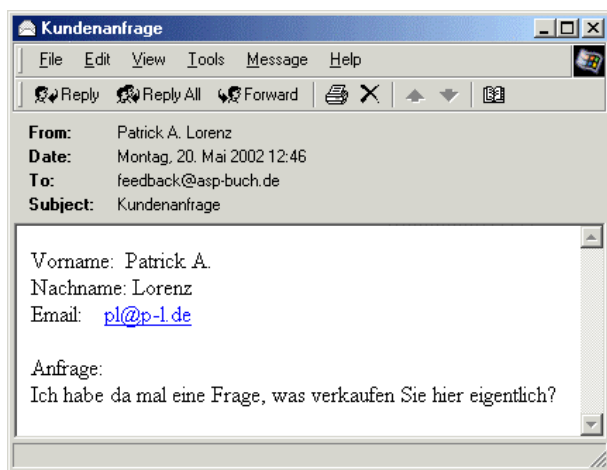


Abbildung 6.17 Die Anfrage wurde als Email weitergeleitet.

6.13 ... ein Formular mit Datei-Upload erstellen?

In vielen Fällen ist es durchaus hilfreich, dem Benutzer den Upload einer oder mehrerer Dateien direkt über den Browser anzubieten. Eine entsprechende Unterstützung ist im Protokoll HTTP vorgesehen, und auch das HTML-Format erlaubt die Angabe eines speziellen Eingabetyps „file“. Mit früheren Versionen von ASP war

es jedoch recht mäßig, die übertragenen Daten korrekt auszuwerten und beispielsweise auf dem Server abzuspeichern. Oftmals ließ sich dies nur mit Hilfe einer Komponente realisieren. Mit ASP.NET wird das nun ganz anders ...

Eine einzelne Datei übertragen

ASP.NET stellt ein HTML Server Control zur Verfügung, über das Sie dem Benutzer den Upload von Dateien anbieten können. Sie müssen lediglich das gewünschte Element über das input-Tag anlegen. Vergessen hierbei bitte nicht das runat- sowie das type-Attribut. Letzteres muss auf „file“ gesetzt sein. Das Listing zeigt den Upload einer Datei.

Listing 6.20 FileUpload1.aspx

```
<script runat="server">

void Button_Click(object sender, EventArgs e)
{
    HttpPostedFile file = upload.PostedFile;

    if(file != null)
    {
        Response.Write("<h1>Vielen Dank!</h1>");
        Response.Write("Dateiname: " + file.FileName + "<br>");
        Response.Write("Größe: " + file.ContentLength + " Byte <br>");
        Response.Write("Content-Type: " + file.ContentType + "<br>");
    }
}

</script>

<form runat="server" enctype="multipart/form-data">

    <input type="file" id="upload" runat="server"/>

    <ASP:Button runat="server" text="Uploaden" OnClick="Button_Click"/>

</form>
```

Im Listing ist das beschriebene serverseitige Control zu erkennen. Ein Klick auf den Button löst den Upload aus. Die übertragenen Daten sind über die Eigenschaft `PostedFile` des Controls in Form einer Instanz der Klasse `HttpPostedFile` zugänglich. Hier lassen sich Informationen wie ursprünglicher Dateiname, Größe und Art abfragen. Die Abbildung zeigt die Ausgaben anhand meiner `autoexec.bat`.

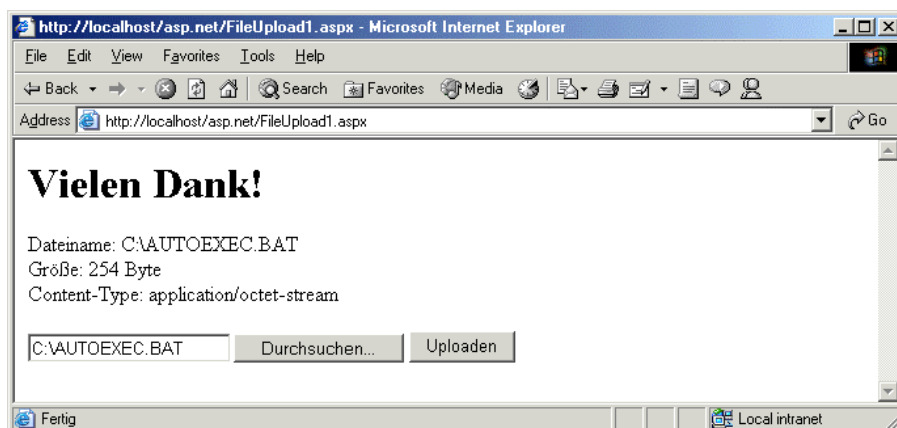


Abbildung 6.18 Ein Button-Klick, und die Datei wird zum Server übertragen.



Beachten Sie bitte, dass das serverseitige Formular *unbedingt* mit dem Zusatz `enctype="multipart/form-data"` versehen werden muss. Ansonsten können die Daten nicht übertragen werden, und es passiert schlichtweg nichts.

Eine Datei uploaden und verwenden

Selbstverständlich können Sie auf den Inhalt der Datei zugreifen, sobald diese übertragen wurde. Die Eigenschaft `InputStream` der Klasse `HttpPostedFile` liefert einen Stream, der wie üblich verwendet werden kann. Im zweiten Listing sehen Sie, wie dieser Stream mit Hilfe eines `StreamReader` ausgelesen und zurück an den Client übertragen wird.

Listing 6.21 FileUpload2.aspx

```
...  
void Button_Click(object sender, EventArgs e)  
{  
    HttpPostedFile file = upload.PostedFile;  
  
    if(file != null)  
    {  
        Response.Write("<h1>Vielen Dank!</h1>");  
        Response.Write("Dateiname: " + file.FileName + "<br>");  
        Response.Write("Größe: " + file.ContentLength + " Byte <br>");  
        Response.Write("Content-Type: " + file.ContentType + "<br>");  
  
        Response.Write("<h2>Inhalt:</h2>");  
        StreamReader streamreader = new StreamReader(file.InputStream);
```

```

string contents = streamreader.ReadToEnd();
contents = contents.Replace("\r\n", "<br>");
Response.Write(contents);
}
}
...

```

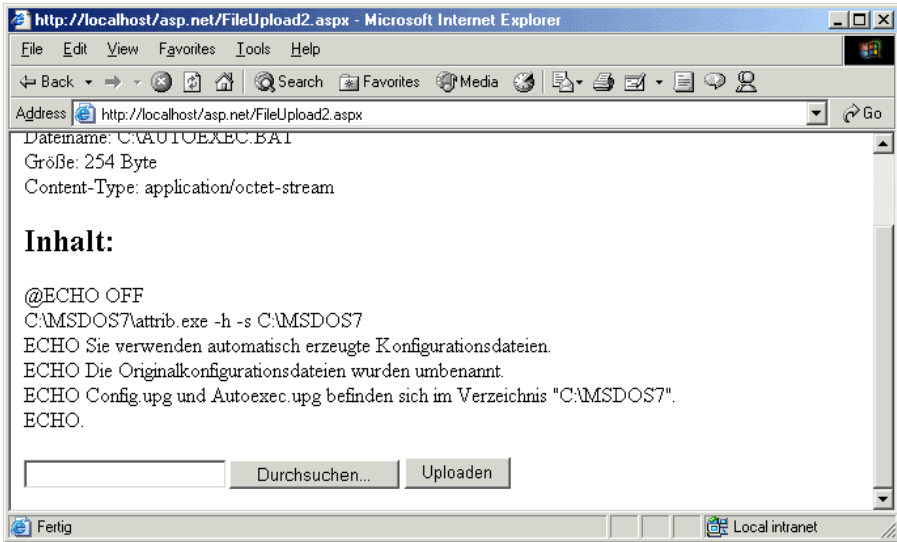


Abbildung 6.19 Der Inhalt der Datei ist als Stream verfügbar.

Natürlich können Sie den Stream auch anderweitig verwenden. So ist es beispielsweise denkbar, das Formular auf die Übertragung von Bildern einzuschränken. Eine entsprechende Abfrage kann nach dem Upload über die Dateieindung vorgenommen werden. Anschließend kann auf Basis des Streams eine neue Instanz der Klasse `Bitmap` erstellt werden.

Listing 6.22 FileUpload3.aspx

```

...
void Button_Click(object sender, EventArgs e)
{
    HttpPostedFile file = upload.PostedFile;

    if(file != null)
    {
        Response.Write("<h1>Vielen Dank!</h1>");
        Response.Write("Dateiname: " + file.FileName + "<br>");
        Response.Write("Größe: " + file.ContentLength + " Byte <br>");
        Response.Write("Content-Type: " + file.ContentType + "<br>");
    }
}

```

```
Bitmap b = new Bitmap(file.InputStream);  
// Weitere Verarbeitung des Bildes  
}  
}  
...
```



Sofern Sie bei der Instanziierung der Klasse `Bitmap` eine `ArgumentException` mit dem Hinweis „Invalid parameter used“ erhalten, verzweifeln Sie bitte nicht. In diesem Fall haben Sie vermutlich eine ungültige Grafikdatei übertragen. Die Klasse wirft die Ausnahme, sobald sie mit den Daten nicht zurechtkommt.

Eine Datei uploaden und abspeichern

Statt direkt auf die übertragenen Daten zuzugreifen, können Sie diese selbstverständlich auch auf dem lokalen Datenträger des Servers abspeichern. Sie müssen hierzu noch nicht einmal manuell auf den Stream zugreifen, sondern können ganz einfach den gewünschten lokalen Dateinamen der Methode `SaveAs` der Klasse `HttpPostedFile` übergeben.

Listing 6.23 FileUpload4.aspx

```
...  
void Button_Click(object sender, EventArgs e)  
{  
    HttpPostedFile file = upload.PostedFile;  
  
    if(file != null)  
    {  
        Response.Write("<h1>Vielen Dank!</h1>");  
        Response.Write("Dateiname: " + file.FileName + "<br>");  
        Response.Write("Größe: " + file.ContentLength + " Byte <br>");  
        Response.Write("Content-Type: " + file.ContentType + "<br>");  
  
        string filename = Path.GetFileName(file.FileName);  
        filename = Path.Combine(Server.MapPath(""), filename);  
        file.SaveAs(filename);  
        Response.Write("<p>Die Datei wurde abgespeichert</p>");  
    }  
}  
...
```


Die prinzipielle Anwendung der Methode `SaveAs` bedarf wohl keiner weiteren Beschreibung. Wichtig ist jedoch die Ermittlung des Dateinamens. Über die Eigenschaft `FileName` wird der ursprüngliche Pfad auf dem Client geliefert. Sollte dieser Pfad analog auf dem Server existieren, dürfte es sich um einen Zufall handeln. In jedem Fall hat die Anlage unter diesem Pfad wenig Sinn. Vielmehr sollte der Dateiname wie im Beispiel gezeigt extrahiert und mit einem lokalen Verzeichnis verbunden werden. Für beide Funktionen werden statische Methoden der Klasse `Path` aus dem zuvor zu importierenden Namespace `System.IO` verwendet.



Beachten Sie, dass der Benutzer-Account „ASPNET“ über alle zur Dateianlage benötigten Rechte auf Ebene der Windows ACL (Access Control List) verfügen muss. Der Benutzer verfügt initiiell über stark eingeschränkte Rechte, die Sie daher unbedingt noch einmal überprüfen und gegebenenfalls erweitern sollten.

Eine Datei uploaden und per Email versenden

Sie können eine übertragene Datei auch direkt per Email weitersenden. Dies kommt beispielsweise bei Kontaktformularen im Bereich der Produktunterstützung (Support) zum Tragen, wenn Benutzer zusätzliche Daten übermitteln sollen.

Die Klasse `MailMessage` aus dem Namespace `System.Web.Mail` ermöglicht das Versenden von Nachrichten über den eingebauten SMTP-Dienst. Auch eine Unterstützung für Attachment ist gegeben. Diese erlaubt jedoch ausschließlich die Angabe von Dateinamen, nicht aber das Übergeben von Streams. Das ist sehr schade und eine absolut inkonsequente Fortführung der .NET-Konzepte. Hoffentlich wird sich in zukünftigen Versionen des Frameworks hier etwas tun. Bis es so weit ist, bleibt leider nur das temporäre Abspeichern der Datei auf dem Server.

Das Listing zeigt eine Implementierung zum Versenden der Datei per Email. Es wird temporär ein Verzeichnis auf Basis der eindeutigen Session-ID angelegt. Dies ist notwendig, damit sich parallele Zugriffe nicht behindern. In das Verzeichnis wird die Datei abgespeichert und anschließend als Attachment mit der Email versendet. Nun wird das komplette Verzeichnis wieder entfernt.

Listing 6.24 FileUpload5.aspx

```
...
string directory = Path.Combine(Server.MapPath(""),
    Session.SessionID);
if(!Directory.Exists(directory))
    Directory.CreateDirectory(directory);

string filename = Path.GetFileName(file.FileName);
filename = Path.Combine(directory, filename);
```

```
file.SaveAs(filename);

MailMessage mail = new MailMessage();
mail.From = "pl@p-l.de";
mail.To = "pl@p-l.de";
mail.Subject = "Dateianhang";
mail.Body = "Anbei die hochgeladene Datei.";
mail.Attachments.Add(new MailAttachment(filename));
SmtpMail.Send(mail);

Directory.Delete(directory, true);

Response.Write("<p>Die Datei wurde per Email versendet</p>");
}
}
...
```

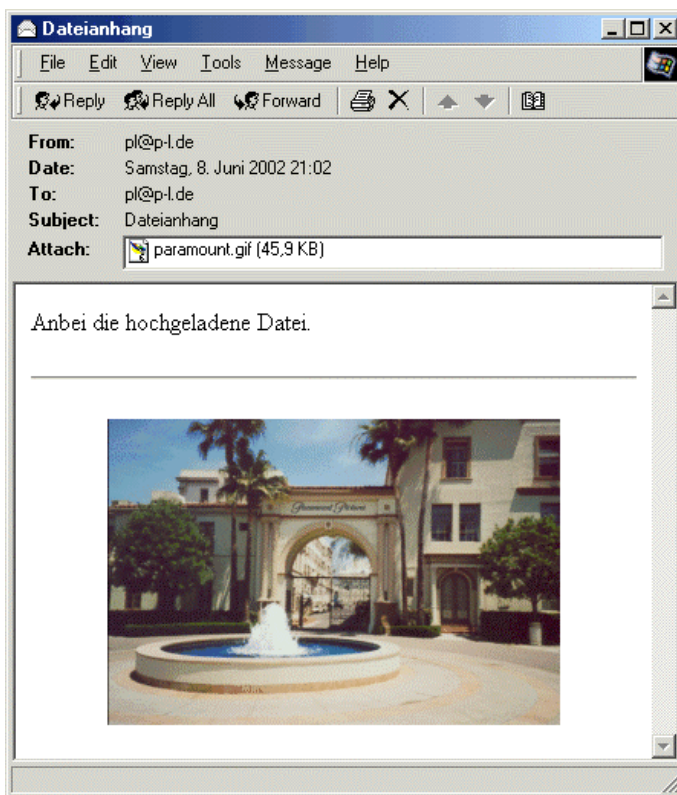


Abbildung 6.20 Die Datei wurde hochgeladen und versendet.



Beachten Sie bitte auch hier, dass der Benutzer-Account „ASPNET“ über die notwendigen Rechte zur Anlage von Verzeichnissen und Dateien verfügen und diese auch wieder löschen können muss. Ist dies nicht gegeben, erhalten Sie IO-Ausnahmen.

Mehrere Dateien uploaden

Auch die Übertragung von mehreren Dateien gleichzeitig ist keinerlei Problem. Sie können dazu einfach mehrere `HtmlInputFile-Controls` platzieren. Statt diese nun einzeln abzufragen, können Sie über die Eigenschaft `Files` der Klasse `HttpRequest` eine `HttpFileCollection` mit allen übertragenen Dateien erhalten. Leider ist die Klasse ein wenig unsauber implementiert. Der Enumerator liefert die internen Namen der einzelnen Controls als Zeichenkette. Erst mit dieser – für den Entwickler wenig relevanten – Information lässt sich die eigentliche Datei über den Index der Collection abfragen. Zudem kommt hinzu, dass nicht verwendete Eingabeelemente nicht erkannt werden und die Eigenschaft `Count` somit nicht die Anzahl der Dateien, sondern die Anzahl der Objekte liefert.

Im Listing sehen Sie nun mehrere Eingabeobjekte zur Übertragung von Dateien. Der Zugriff erfolgt über eine `foreach...in`-Schleife und somit über den Enumerator der Collection. Über die Eigenschaft `ContentLength` wird abgefragt, ob tatsächlich eine Datei angegeben wurde, bevor weitere Daten zu dieser im Browser ausgegeben werden.

Listing 6.25 FileUpload6.aspx

```
<script runat="server">

void Button_Click(object sender, EventArgs e)
{
    Response.Write("<h1>Vielen Dank!</h1>");
    Response.Write("Es wurden " + Request.Files.Count +
        " Dateien übertragen:<br>");

    foreach(string obj in Request.Files)
    {
        HttpPostedFile file = Request.Files[obj];
        if(file.ContentLength > 0)
            Response.Write("&bull; " + file.FileName + "<br>");
    }
}

</script>

<form runat="server" enctype="multipart/form-data">
```

```
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>
<input type="file" runat="server"/>

<ASP:Button runat="server" text="Uploaden" OnClick="Button_Click"/>

</form>
```

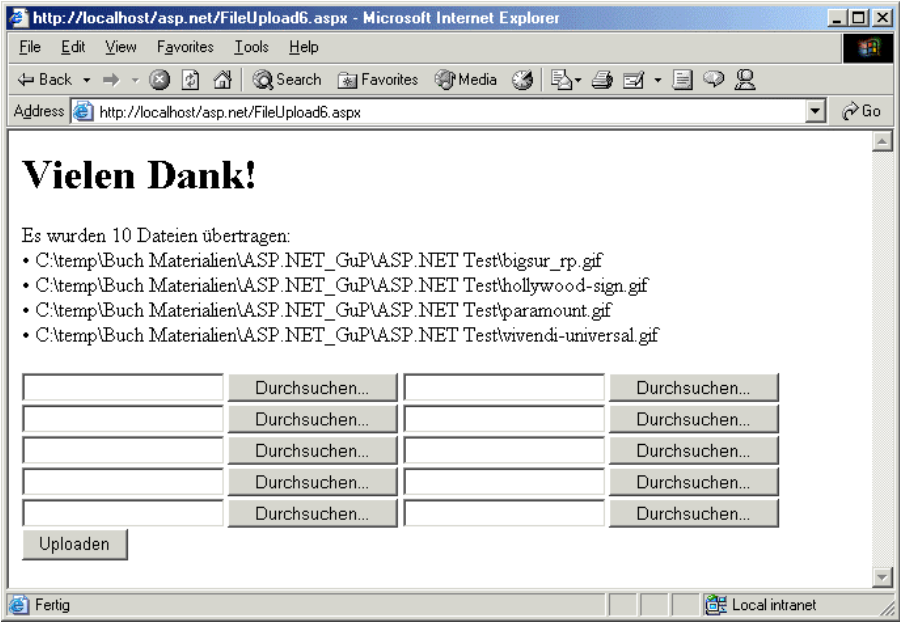


Abbildung 6.21 Die gelieferte Anzahl stimmt nicht mit dem tatsächlichen Wert überein.

6.14 ... eine Email-Vorlage erstellen?

Nahezu jede Website nutzt das Versenden von Emails. Es ergeben sich dabei zwei wichtige Hauptaufgaben. Einerseits gehen Emails mit Informationen an die Besucher, und andererseits werden über das Medium Daten zur Weiterverarbeitung an den Betreiber gesendet. Ein Beispiel hierfür ist das Kontaktformular aus dem Rezept „... ein einfaches Kontaktformular erstellen?“.

Das .NET Framework bietet mit der Klasse `MailMessage` eine einfache, objektorientierte Möglichkeit, eine Email zu senden. Wichtige Parameter wie Absender, Empfänger, Betreff und Inhalt werden über Eigenschaften angeboten. Gerade bei größeren Emails ist es jedoch relativ umständlich, diese mit Daten zu füllen. Mittels Zeichenkettenoperationen muss der Inhalt manuell hinterlegt werden. Insbesondere die Einbindung der Email in den Quellcode ist recht ungeschickt.

Eine saubere Trennung zwischen Email und Quellcode liefert der Mail-Parser, den ich Ihnen nachfolgend vorstellen möchte. Es handelt sich um eine Ableitung der Klasse `MailMessage`. Erweitert wurde diese um einen Konstruktor sowie die Methode `ParseTemplate`. Übergeben wird jeweils ein Dateiname und eine `NameValueCollection`. Die angegebene Datei enthält eine Vorlage mit Platzhaltern. Beim Aufruf der Methode wird der Text eingelesen, und die Platzhalter werden durch die entsprechenden Zeichenketten der übergebenen Collection ersetzt. Nun können die mit Echtdaten gefüllten Elemente den korrespondierenden Eigenschaften der Klasse zugewiesen werden.

Das Listing zeigt eine Implementierung der Klasse `MailMessageExt` in Form einer C#-Quellcode-Datei. Deutlich erkennbar ist das Durchlaufen der übergebenen Collection und das Ersetzen der Platzhalter innerhalb der Vorlage. Im Anschluss werden die Zeilen mittels eines Arrays einzeln untersucht. Enthalten diese zum Beginn Anweisungen wie „To:“ oder „From:“, werden die nachfolgenden Inhalte den entsprechenden Eigenschaften zugewiesen.

Listing 6.26 MailParser.cs

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.IO;
using System.Web;
using System.Web.Mail;

public class MailMessageExt : MailMessage
{
    public MailMessageExt(string filename, NameValueCollection variables)
    {
        this.ParseTemplate(filename, variables);
    }
}
```

```
public void ParseTemplate(string filename,
    NameValueCollection variables)
{
    StreamReader reader = File.OpenText(filename);
    string template = reader.ReadToEnd();
    reader.Close();

    foreach(string key in variables.Keys)
    {
        string placeholder = "<" + key + ">";
        string content = variables[key];
        template = template.Replace(placeholder, content);
    }

    string reduced = template.Replace("\r", "");
    string[] lines = reduced.Split(new char[] { '\n' });
    for(int i=0; i<lines.Length; i++)
    {
        string line = lines[i].ToLower();

        if(line.StartsWith("from:")) this.From = lines[i].Substring(6);
        if(line.StartsWith("to:")) this.To = lines[i].Substring(4);
        if(line.StartsWith("cc:")) this.Cc = lines[i].Substring(4);
        if(line.StartsWith("bcc:")) this.Bcc = lines[i].Substring(5);
        if(line.StartsWith("subject:"))
            this.Subject = lines[i].Substring(9);

        if(line == string.Empty)
        {
            this.Body = string.Join("\n", lines, i+1, lines.Length-i-1);
            return;
        }
    }

    this.Body = template;
}
```

Das Listing sieht recht einfach aus, und das ist es auch. Dennoch kann es unheimlich viel Zeit und Arbeit ersparen. Hierüber eine Email zu versenden, ist wirklich ein Kinderspiel. Nachfolgend finden Sie eine typische Email-Vorlage, die der Parser verarbeiten kann. In spitzen Klammern sind die Platzhalter hinterlegt, die später durch die jeweiligen Inhalte ersetzt werden sollen.

Listing 6.27 mail.txt

```
From: webmaster@asp-buch.de
To: <Email>
Subject: Ihre Anmeldung

Guten Tag <Anrede> <Name>,

vielen Dank fuer Ihre Anmeldung.

Anbei erhalten Sie die benoetigten Benutzerdaten zum Zugriff
auf die Website www.asp-buch.de. Die Daten lauten wie folgt:

Email: <Email>
Passwort: <PW>

Viel Spass!

Webmaster
```

Was nun noch fehlt, ist ein kleines Anwendungsbeispiel, das den Mail-Parser im Einsatz zeigt. Basierend auf der gezeigten Vorlage instanziiert die nachfolgende ASP.NET-Seite eine neue `NameValueCollection`. Über die `Add`-Methode werden die zur Verfügung stehenden Daten angemeldet. Anschließend werden diese dem Konstruktor der Klasse `MailMessageExt` übergeben. Intern wird nun bereits die Email geparkt und kann daher mit einem einfachen Aufruf der statischen Methode `SmtpMail.Send` versendet werden.

Listing 6.28 MailParser1.aspx

```
<% @Assembly src="MailParser.cs" %>
<% @Import Namespace="System.Web.Mail" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string filename = Server.MapPath("mail.txt");

    NameValueCollection variables = new NameValueCollection();
    variables.Add("Email", "pl@p-l.de");
    variables.Add("Anrede", "Herr");
    variables.Add("Name", "Lorenz");
    variables.Add("PW", "abcdefgh");

    MailMessageExt mail = new MailMessageExt(filename, variables);

    SmtpMail.Send(mail);
}
```

```
</script>
```

Die Email wurde gesendet!

Die Abbildung zeigt das Ergebnis, die gesendete Email. Alle Platzhalter wurden planmäßig durch die korrespondierenden Einträge der `NameValueCollection` ersetzt.

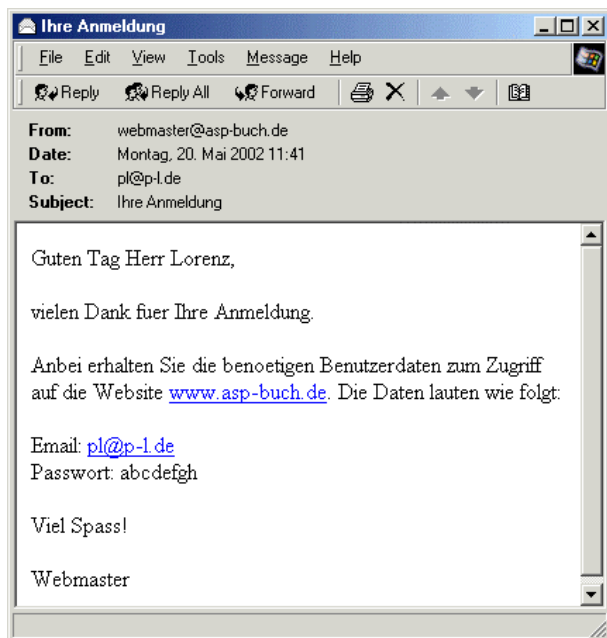


Abbildung 6.22 Die Email wurde erfolgreich geparkt und gesendet.

Um den Mail-Parser in eigenen Seiten zu verwenden, müssen Sie die Quellcode-Datei lediglich über die `@Assembly` referenzieren. Anschließend können Sie mit wenigen Zeilen auch komplexe Vorlagen auswerten und versenden. Der Inhalt der Email ist dabei immer vom Quellcode der Seite getrennt und kann zudem wiederverwendet werden.



Beachten Sie bitte, dass die vorliegende Implementierung des Mail-Parsers eine groß-klein-sensitive Übergabe der Variablen erwartet. Sie müssen die Variablen daher genau so verwenden, wie sie der `NameValueCollection` übergeben wurden.

6.15 ... mehrere Felder in einer DropDownList oder ListBox anzeigen?

Bei den datengebundenen Controls `DropDownList` und `ListBox` ist es häufig sinnvoll, den angezeigten Text aus mehreren Datenbankfeldern zusammenzusetzen. Die individuelle Formatierung ist hier analog zum `DataGrid`-Control über die Eigenschaft `DataTextFormatString` möglich. Hierüber lässt sich der Text individuell formatieren, mehrere können jedoch nicht zusammengefasst werden.

Die beiden Controls bieten keine Ereignisse zum Eingriff an, so dass mehrere Felder bereits vor der Datenbindung zusammengefasst werden müssen. Hierzu bieten sich die zwei – im Rezept „... mehrere Felder in einer DataGrid-Spalte anzeigen?“ beschriebenen – Möglichkeiten per SQL beziehungsweise der dynamischen Erweiterung der `DataTable`-Instanz an. Das Vorgehen entspricht dem dort vorgestellten Beispiel. Die beiden nachfolgenden Beispiele sind daher nicht weiter erklärt. Im ersten wird eine `DropDownList` und die SQL-Variante gezeigt, im zweiten eine `ListBox`, die per `DataTable` erweitert wird.

Listing 6.29 `DropDownList1.aspx`

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT ID, Firstname+' '+Lastname AS Fullname
        FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    ddl.DataSource = dataset;
    if(!IsPostBack) DataBind();
}

</script>
```

```
<form runat="server">

<ASP:DropDownList id="ddl" runat="server"
    DataTextField="Fullname"
    DataValueField="ID"/>

</form>
```

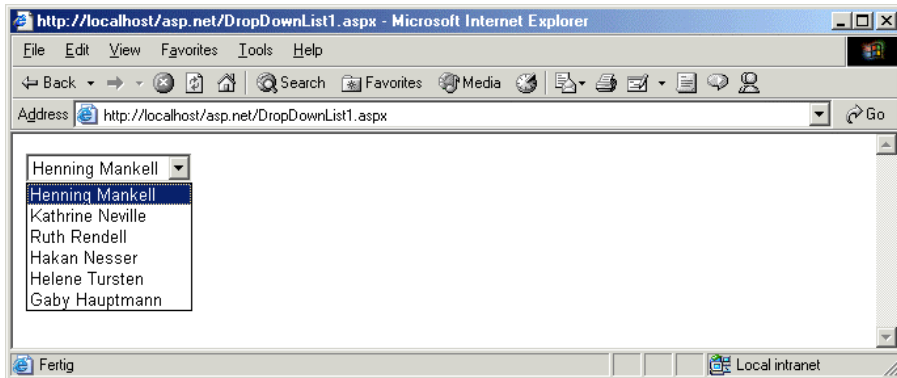


Abbildung 6.23 Das angezeigte Feld wurde über die SQL-Query dynamisch erstellt.

Listing 6.30 ListBox1.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    DataColumn column = new DataColumn("Fullname", typeof(string),
        "Firstname+ ' '+Lastname");
```

```

DataTable table = dataset.Tables[0];
table.Columns.Add(column);

lb.DataSource = dataset;
if (!IsPostBack) DataBind();
}

</script>

<form runat="server">

    <ASP:ListBox id="lb" runat="server"
        DataTextField="Fullname"
        DataValueField="ID"/>

</form>

```

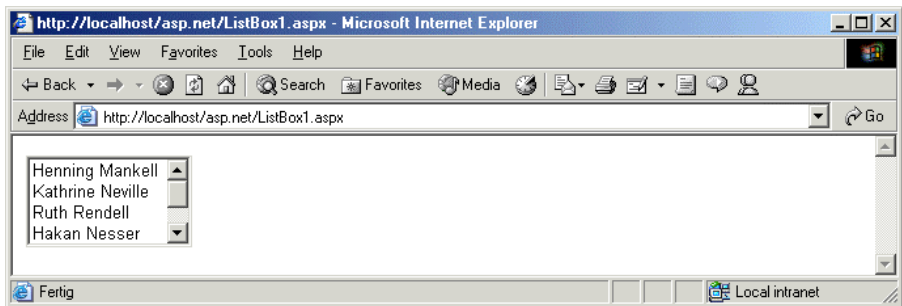


Abbildung 6.24 Der Inhalt der ListBox wurde über eine dynamische Spalte erzeugt.

6.16 ... AutoPostBack und SmartNavigation parallel benutzen

Der Artikel Q314206 aus der Microsoft Knowledge Base erklärt, dass sich AutoPostBack und SmartNavigation nicht korrekt parallel benutzen lassen. Als Problem wird dort die Tatsache beschrieben, dass das Eingabefeld nach dem automatischen Round Trip zum Server den Fokus verliert. Lösungsvorschläge bietet der Artikel nicht an, vielmehr heißt es dort: „This behavior is by design.“ Ganz so einfach mache ich es mir nicht, denn natürlich gibt es auch für dieses Problem eine Lösung.

Das Problem

Bevor ich Ihnen die Lösung zeige, möchte ich das Problem noch einmal erläutern. Ich verwende dazu das Beispiel aus der Microsoft Knowledge Base. Dieses enthält zwei TextBox-Controls. Beide lösen bei einer Änderung automatisch einen Round Trip zum Server aus (AutoPostBack). Für die Seite ist zusätzlich die SmartNavigation aktiviert. Dieses System sorgt dafür, dass die aktuelle Seitenposition nach dem Round Trip erhalten bleibt.

Listing 6.31 smartnavigation1.aspx

```
<% @Page language="C#" SmartNavigation="true" %>

<script runat="server">

</script>

<html>
<head/>
<body>

  <form runat="server">

    <asp:TextBox AutoPostBack=True id=TextBox1 runat="server"/>
    <asp:TextBox AutoPostBack=True id=TextBox2 runat="server"/>

  </form>

</body>
</html>
```

Das Listing zeigt das Beispiel. Nach der Eingabe eines Textes in das erste Feld löst der Wechsel mittels der Tabulator-Taste einenPostBack aus. Im Anschluss sollte der Cursor sich im zweiten Feld befinden. Dies ist jedoch nicht der Fall, vielmehr steht der Cursor erneut im ersten Feld. Das ist falsch und genau das Problem.

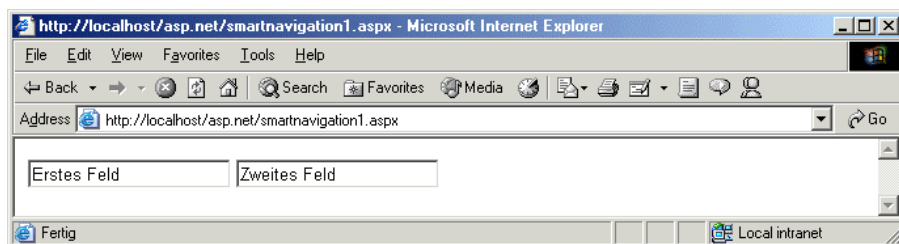


Abbildung 6.25 Das Beispiel enthält zwei TextBox-Controls.

Die Lösung

Des Rätsels Lösung ist ein kleiner clientseitiger Eingriff mittels JavaScript. Nach dem Round Trip zum Server wird über das Ereignis `TextChanged` ein kleines Script registriert, das beim Laden der Seite das zweite Eingabefeld aktiviert. Hierzu verwendet werden die Methode `IsStartupScriptRegistered` sowie `RegisterStartupScript` der Klasse `Page`. Clientseitig kommt der JavaScript-Befehl `focus` zum Einsatz. Das Listing zeigt die Erweiterung des Beispiels, das nun korrekt den Fokus wechselt.

Listing 6.32 smartrnavigation2.aspx

```
<% @Page language="C#" SmartNavigation="true" %>

<script runat="server">

void tb1_changed(object sender, EventArgs e)
{
    if(!IsStartupScriptRegistered("focus"))
    {
        string focus = "<script language=\"javascript\">" +
            "document.all.TextBox2.focus();<" +
            "/script>";
        this.RegisterStartupScript("focus", focus);
    }
}

</script>

<html>
<head/>
<body>

    <form runat="server">

        <asp:TextBox AutoPostBack=True id=TextBox1 runat="server"
            OnTextChanged="tb1_changed"/>
        <asp:TextBox AutoPostBack=True id=TextBox2 runat="server"/>

    </form>

</body>
</html>
```

6.17 ... eine MessageBox im Browser ausgeben?

Da ASP.NET vollständig auf dem Server abläuft, haben Sie keine Möglichkeit, mit den eingebauten Mitteln eine Benutzermeldung wie eine MessageBox anzuzeigen. Sie müssen sich dazu der clientseitigen Programmierung mit JavaScript bedienen. Hier steht Ihnen die Funktion `alert` zur Verfügung, die clientseitig die gewünschte Meldung anzuzeigen vermag.

Um die notwendigen Zeilen clientseitigen Scripts an den Browser zu übertragen, fassen Sie diese in einer Zeichenkette zusammen. Anschließend können Sie diese Zeichenkette mittels der Methode `Page.RegisterClientScriptBlock` registrieren und an den Server übertragen lassen. ASP.NET kümmert sich automatisch um das korrekte Positionieren innerhalb des Quelltextes der Seite.

Listing 6.33 alert1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string message = "Hallo Welt!";
    string alertScript = "<script language=JavaScript>";
    alertScript += "alert('" + message + "');</script" + ">";

    if(!IsClientScriptBlockRegistered("alert"))
        this.RegisterClientScriptBlock("alert", alertScript);
}

</script>

<form runat="server">
</form>
```

Um zu verhindern, dass möglicherweise mehrere Meldungen mit dem gleichen Inhalt angezeigt werden, sollten Sie mittels `IsClientScriptBlockRegistered` überprüfen, ob unter demselben Namen bereits ein Script registriert wurde. Dies ist insbesondere wichtig, wenn die Meldung über ein User Control angezeigt wird.



Abbildung 6.26 Die Meldung wird beim Laden der Seite angezeigt.

Damit das Script an den Client übertragen wird, ist es notwendig, dass die Seite ein serverseitiges Formular enthält. In dessen Kontext wird der übergebene Script-Block abgelegt:

```
<form name="_ctl0" method="post" action="alert1.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="..." />
  <script language=JavaScript>alert('Hallo Welt!');</script>
</form>
```

Damit die gezeigte Meldung nur beim ersten Aufruf der Seite angezeigt wird, können Sie das Script bedingt in Abhängigkeit vomPostBack-Status an den Browser übergeben. Das zweite Listing zeigt eine derartige Implementierung. Nach dem Round Trip zum Server wird die Meldung nicht mehr angezeigt.

Listing 6.34 alert2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        string message = "Herzlich willkommen!";
        string alertScript = "<script language=JavaScript>";
        alertScript += "alert('" + message + "');</script> + ">";

        if(!IsClientScriptBlockRegistered("alert"))
            this.RegisterClientScriptBlock("alert", alertScript);
    }
}

</script>

<form runat="server">
```

```
<ASP:TextBox runat="server"/>
<ASP:Button runat="server" text="ok"/>

</form>
```

Die nachfolgenden Rezepte zeigen weitere Möglichkeiten in Verbindung mit client-seitigem Script.

6.18 ... eine JavaScript-Funktion vor dem Absenden eines Formulars ausführen?

Es kann durchaus nützlich sein, vor dem Absenden eines Eingabeformulars client-seitigen Code auszuführen. Sie können dem Benutzer zum Beispiel schon einmal für seine Angaben danken oder bei längeren Operationen den Absende-Button deaktivieren, um ein mehrmaliges „Anschubsen“ zu verhindern.

Bei regulären HTML-Seiten verwenden Sie das Submit-Ereignis des form-Tags, das Sie mit JavaScript wie im folgenden Listing gezeigt nutzen können. Auch mit ASP.NET lässt sich dieses clientseitige Ereignis benutzen, indem es wie gewohnt dem serverseitigen form-Tag zugewiesen wird. Das Listing zeigt dies in Verbindung mit dem JavaScript-Befehl `alert`, der eine MessageBox im Browser anzeigt. Wie in der Abbildung zu sehen, wird vor jedem Round Trip an den Server die Meldung ausgegeben – vorausgesetzt natürlich, dass der Benutzer JavaScript nicht deaktiviert hat.

Listing 6.35 submit1.aspx

```
<form runat="server" onsubmit="alert('hallo welt');">

  <ASP:TextBox id="tb" runat="server"/>

  <p> <ASP:Button runat="server" text="ok"/> </p>

</form>
```




Abbildung 6.27 Vor dem Round Trip zum Server wird eine Meldung angezeigt.

Dieser Ansatz hat jedoch einen Haken. Sofern innerhalb des Web-Formulars Validation-Controls verwendet werden, benutzen diese das Ereignis ebenfalls. Es wird hier die clientseitige Validierung der Benutzereingaben vorgenommen. Dies sieht beispielsweise so aus:

```
<form name="_ctl10" method="post" action="submit1b.aspx"
  language="javascript" onsubmit="ValidatorOnSubmit();" id="_ctl10">
```

Da eine doppelte Zuweisung des Ereignisses in dieser Form nicht möglich ist, kommt es zu einem Fehler. Was also tun, wenn Validierung und eigener clientseitiger Code ausgeführt werden soll?

Die Lösung ist im Prinzip relativ einfach. Erinnern Sie sich daran, was das form-Tag ist. Es handelt sich um ein Server Control, das über die Klasse `HtmlForm` repräsentiert wird. Sofern Sie eine ID vergeben, können Sie das Control darüber ansprechen, und selbstverständlich können Sie auch auf Ereignisse des Controls reagieren. Eines dieser von der Basis `Control` geerbten Ereignisse heißt `PreRender`. Dieses wird unmittelbar vor dem Rendern der Seite ausgelöst. Sie haben hier beispielsweise die Möglichkeit, die Attribute des Controls zu verändern und den von den Validation Controls eingefügten clientseitigen Quellcode zu erweitern.

Das zweite Listing zeigt eine derartige Erweiterung. Das Attribut `onSubmit` wird um die Ausgabe der alert-Meldung erweitert. Nun wird nach der clientseitigen Validierung aber vor dem Round Trip zum Server eine Meldung im Browserfenster ausgegeben.

Listing 6.36 submit2.aspx

```
<script runat="server">

void form_PreRender(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        HtmlForm form = (HtmlForm) sender;
```

```
    form.Attributes["onsubmit"] += "alert('hallo welt');";
  }
}

</script>

<form runat="server" OnPreRender="form_PreRender">

  <ASP:TextBox id="tb" runat="server"/>

  <ASP:RequiredFieldValidator runat="server"
    ControlToValidate="tb"
    Display="Dynamic"
    Text="Bitte Text eingeben."/>

  <p> <ASP:Button runat="server" text="ok"/> </p>

</form>
```

Ein wenig störend erscheint die Tatsache, dass die Meldung in jedem Fall ausgegeben wird. Auch wenn die Validierung eine fehlerhafte Benutzereingabe erkannt hat und kein Round Trip zum Server stattfindet, wird dem Benutzer die alert-MessageBox angezeigt. Mit einer kleinen Erweiterung ist auch dies zu beseitigen. Das dritte Listing zeigt eine Erweiterung des übergebenen Quellcodes. Hier wird die Eigenschaft `ReturnValue` des event-Objekts abgefragt. Nur wenn dieses `true` liefert, war die Validierung erfolgreich, und die Meldung wird angezeigt.

Listing 6.37 submit3.aspx

```
<script runat="server">

void form_PreRender(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        HtmlForm form = (HtmlForm) sender;
        form.Attributes["onsubmit"] +=
            "if(event.returnValue) alert('hallo welt');";
    }
}

</script>

...
```

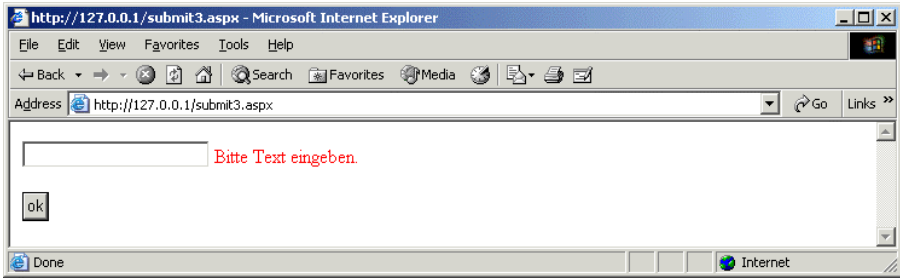


Abbildung 6.28 Nur wenn die Eingaben korrekt sind, wird die Meldung ausgegeben.

Statt der alert-Meldung können Sie selbstverständlich auch andere oder eigene JavaScript-Funktionen aufrufen. Der hierzu notwendige Quelltext sollte getrennt übergeben werden. Wie dies geht, lesen Sie im Kapitel „Basics“.

6.19 ... das Absenden eines Formulars bestätigen lassen?

Unnötige Sicherheitsabfragen und ständige Nachfragen der Textverarbeitung, ob nun man auch *wirklich* sicher ist, dass man diesen Satz *wirklich* schreiben will, sind gerade bei professionellen Anwendern verpönt. Je nach Zielgruppe und Art der Web-Applikation bietet sich eine zusätzliche Benutzerinteraktion dennoch durchaus an. JavaScript stellt hierzu die Funktion `confirm` zur Verfügung, die den übergebenen Text samt OK- und Abbrechen-Button anzeigt. Die Wahl des Benutzers wird in Form eines booleschen Wertes zurückgeliefert.

Das Listing demonstriert den Einsatz der Funktion. Das Beispiel basiert auf dem vorherigen Rezept. Sobald der Benutzer das gezeigte Formular abzusenden versucht und die clientseitige Validierung Ihr Einverständnis gibt, wird die Sicherheitsabfrage eingeblendet. Nur wenn der Benutzer diese bestätigt, wird das Formular tatsächlich abgesendet. Ansonsten passiert schlichtweg nichts.

Listing 6.38 submit4.aspx

```
<script runat="server">

void form_PreRender(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        HtmlForm form = (HtmlForm) sender;
        form.Attributes["onsubmit"] += "if(event.returnValue)
            return(confirm('Sind Sie sicher, dass Sie fortfahren möchten?'))";
    }
}
```

```
</script>

<form runat="server" OnPreRender="form_PreRender">

  <ASP:TextBox id="tb" runat="server"/>

  <ASP:RequiredFieldValidator runat="server"
    ControlToValidate="tb"
    Display="Dynamic"
    Text="Bitte Text eingeben."/>

  <p> <ASP:Button runat="server" text="ok"/> </p>

</form>
```

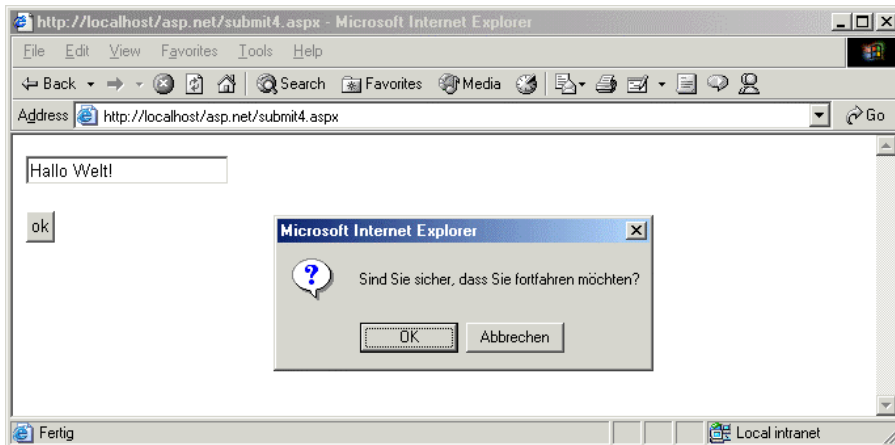


Abbildung 6.29 Das Formular wird erst nach der Bestätigung abgesendet.

Sie können die Sicherheitsmeldung alternativ auch einem bestimmten Button zuordnen. Nun wird diese nur angezeigt, wenn beispielsweise der OK-Button angeklickt wurde. In diesem Fall sollten Sie statt eines Web Control ein HTML Server Control verwenden, da Sie die clientseitige Ereignisbehandlung hier direkt zuweisen können.

Listing 6.39 submit5.aspx

```
<form runat="server">

  <ASP:TextBox id="tb" runat="server"/>

  <ASP:RequiredFieldValidator runat="server"
    ControlToValidate="tb"
```

```

    Display="Dynamic"
    Text="Bitte Text eingeben."/>

<p>
  <input type="submit" runat="server" value="ok"
    onclick="return(confirm('Sind Sie sicher, dass Sie fortfahren
    möchten?'));" />

  <input type="submit" runat="server" value="abbrechen"
    causesvalidation="false" />
</p>

</form>

```

Während der OK-Button eine Abfrage auslöst, resultiert der Abbrechen-Button in jedem Fall in einen Round Trip zum Server. Durch Deaktivieren der Option `CausesValidation` wird zudem keine Validierung durchgeführt. Hier zeigt sich jedoch der Nachteil dieses Ansatzes gegenüber dem ersten, denn die Sicherheitsabfrage wird auch dann angezeigt, wenn die clientseitige Validierung einen Fehler in den Benutzereingaben erkennt. Dies lässt sich zu diesem Zeitpunkt jedoch noch nicht abfragen.



Wie immer, wenn es um JavaScript geht, ist auch in diesem Fall nicht sichergestellt, dass die Befehle tatsächlich ausgeführt werden. Unterstützt der Browser die Scriptsprache nicht oder hat der Benutzer deren Verwendung deaktiviert, werden die Anweisungen schlichtweg ignoriert. Sie sollten daher entweder tolerant reagieren oder zuvor sicherstellen, dass JavaScript wirklich aktiviert ist. Wie dies geht, erfahren Sie im Rezept „... überprüfen, ob JavaScript aktiviert ist?“ im Kapitel „Basics“.

6.20 ... Seitenvariablen über einenPostBack hinweg speichern?

Eingabeelemente haben die angenehme Angewohnheit, ihren Status über mehrere Seitenaufrufe hinweg zu behalten. Es halten sich um eines *der* Kern-Features von ASP.NET. Die Objekte verwenden zur Ablage der Daten den ViewState der Seite. Es handelt sich hierbei um ein Dictionary. Die dort abgelegten Daten werden serialisiert innerhalb der Seite an den Client übertragen. Es wird ein verstecktes Formularfeld verwendet, so dass die Daten bei einemPostBack zurück an den Server übertragen, dort serialisiert werden und anschließend wie vor demPostBack im ViewState verfügbar sind.

Variablen, die Sie individuell in Ihrer Seite verwenden, werden hingegen nicht abgespeichert. Das ist aufgrund der Architektur einerseits kein Wunder und andererseits auch gar nicht notwendig. Eine Serialisierung und Übertragung aller Daten wäre indes auch eine nicht akzeptable Belastung der Performance.

Ist die Persistenz einer Variablen über mehrere Seitenaufrufe hinweg erforderlich, so können Sie diese im Session-Scope ablegen. Dies ist mit den bekannten Nachteilen und Einschränkungen verbunden. Viel einfacher ist dagegen die individuelle Ablage im ViewState. Über die gleichnamige und als `protected` markierte Eigenschaft der Klasse `Page` erhalten Sie Zugriff auf das beschriebene Dictionary vom Typ `StateBag`. Sie können wie gewohnt Daten in Verbindung mit einem eindeutigen Schlüssel ablegen und diese später wieder abrufen.

Das Listing zeigt die individuelle Verwendung des ViewState-StateBag. Ein Element „count“ wird beim ersten Aufruf der Seite mit 0 initialisiert. Bei jedem Klick auf den eingefügten Button wird dieser Wert inkrementiert und das Ergebnis im Browser angezeigt.

Listing 6.40 ViewState1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        ViewState["count"] = 0;
        Response.Write("Bitte Button anklicken");
    }
}

void Button_Click(object sender, EventArgs e)
{
    int count = (int) ViewState["count"];
    count++;
    ViewState["count"] = count;
    Response.Write("Sie haben den Button bereits ");
    Response.Write(count.ToString());
    Response.Write(" Mal angeklickt.");
}

</script>

<form runat="server">

<ASP:Button runat="server" text="Klick mich" OnClick="Button_Click"/>

</form>
```

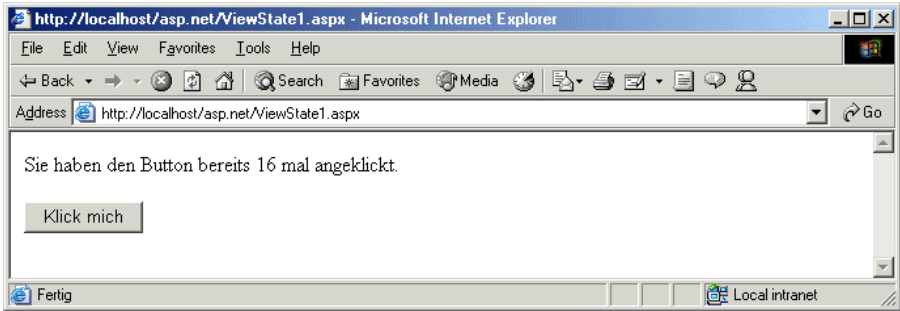


Abbildung 6.30 Jeder Klick auf den Button wird gezählt.

Die Abbildung zeigt deutlich, dass jeder Klick auf den Button unnachgiebig gezählt und der Inhalt des ViewState-StateBag somit über mehrere Seitenaufrufe hinweg gespeichert wird. Nachfolgend sehen Sie die dazu an den Client übertragenen Daten. Das versteckte Formularfeld mit dem Namen „__VIEWSTATE“ enthält in serialisierter Form unter anderem den aktuellen Wert des Zählers.

```
Sie haben den Button bereits 16 Mal angeklickt.<form name="_ctl0"
method="post" action="ViewState1.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" va-
lue="dDwtMTA1NTU5ODU0Mjt0PHA8bDxjb3VudDs+O2w8aTwxNj47Pj47Oz47PoPMmjuNY
HPaxr5/sv3J+osD10EV" />
<input type="submit" name="_ctl1" value="Klick mich" />
</form>
```

Da der Inhalt innerhalb der übertragenen Seite gespeichert wird, ist dieses Verfahren mit jedem gängigen Browser und vor allem jeden Sicherheitseinstellungen verwendbar. Es wird kein Cookie benötigt, und auch auf dem Server werden keine zusätzlichen Daten gespeichert. Alles steckt im versteckten Formularfeld. Daraus resultierend ist es durchaus möglich, dass ein Benutzer die gleiche Seite mehrfach mit unterschiedlichen Werten anzeigt.



Beachten Sie bitte, dass die Verwendung des ViewStates nur in Verbindung mit einem serverseitigen Formular möglich ist. Ansonsten kann das benötigte Formularfeld nicht angelegt werden. Ohne dieses Formular ergibt die Verwendung der Technik jedoch ohnehin wenig Sinn, da kein PostBack ausgelöst werden könnte.

6.21 ... Web Controls zur Laufzeit rendern?

Was passiert eigentlich, wenn ein Web Control gerendert wird? Klar, es wird im Browser dargestellt. Doch was passiert intern? Hier wird rekursiv ausgehend von der Seite für alle Controls die Methode `RenderControl` aufgerufen. Diese sorgt dafür, dass die als `protected` markierte Methode `Render` aufgerufen wird und das Control gerendert und als HTML an den Client übertragen wird. Die HTML-Daten werden dabei an eine Instanz der Klasse `HtmlTextWriter` übergeben.

Sie können sich dieses System zunutze machen und Controls individuell rendern. Sie müssen im Grunde nur eine eigene Instanz der Klasse `HtmlTextWriter` an das Control übergeben, und schon wird das Control für Sie und nicht für den Client gezeichnet.

Im Listing sehen Sie, wie ein Calendar-Control individuell gerendert wird. Die Basis bildet eine Instanz der Klasse `StringWriter`. Hier übergibt der ebenfalls neu instanziierte `HtmlTextWriter` die Daten, wenn die Methode `RenderControl` des Kalenders aufgerufen wird. Anschließend können die erzeugten HTML-Ausgaben wie im Beispiel kodiert an den Browser übertragen werden. Die Abbildung zeigt somit statt des Controls dessen Quelltext.

Listing 6.41 `RenderControl1.aspx`

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    StringWriter stringwriter = new StringWriter();
    HtmlTextWriter htmlwriter = new HtmlTextWriter(stringwriter);

    Calendar calendar = new Calendar();
    calendar.RenderControl(htmlwriter);

    string html = stringwriter.ToString();
    Response.Write(Server.HtmlEncode(html));
}

</script>
```

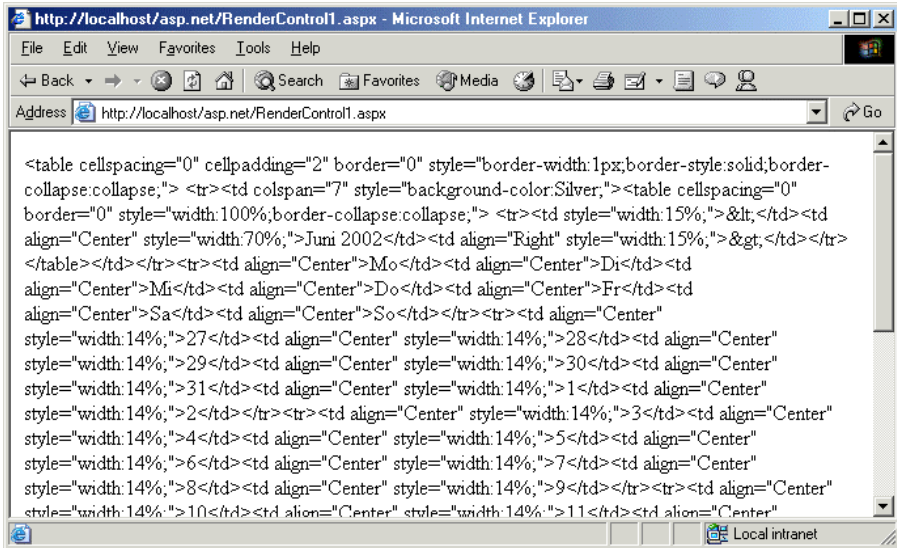



Abbildung 6.31 Das Control wurde individuell gerendert.

Über den praktischen Sinn und Zweck dieses Vorgehens lässt sich sicherlich streiten. Es gibt vermutlich Anwendungsfälle, doch sind diese rar. In jedem Fall aber zeigt das Beispiel auf interessante Weise, wie das Framework intern die Abarbeitung einer Seite steuert.

6.22 ... ein formatiertes Eingabefeld anbieten?

Die Möglichkeiten eines regulären Eingabefeldes sind zwar meistens ausreichend, manchmal bedarf es jedoch mehr Funktionalität. Haben Sie nicht auch schon einmal davon geträumt, Eingaben wie in einer kleinen Textverarbeitung vornehmen zu können? Sie glauben, das geht nicht? Falsch gelegen, denn es geht sehr wohl. Schauen Sie sich bitte einmal das folgende Listing an.

Listing 6.42 rtf1.aspx

```
<%@ Register TagPrefix="RTB" Namespace="RichTextBoxControl"
    Assembly="RichTextBox" %>

<form runat="server">

    <RTB:RichTextBox id="myRTB" runat="server"/>

</form>
```

Das Listing sieht zugegebenermaßen nicht sehr spannend aus, aber warten Sie einmal ab, bis Sie die Abbildung des Controls sehen ...

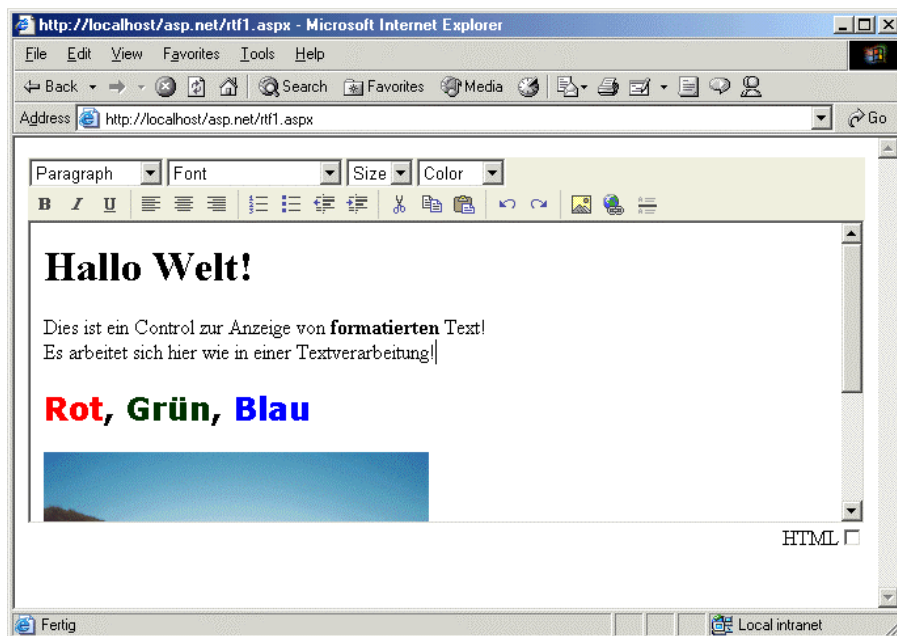


Abbildung 6.32 Die Seite stellt eine kleine Textverarbeitung zur Verfügung.

Wow, ist das nicht eindrucksvoll? Das sieht aus wie eine kleine Textverarbeitung und lässt sich auch genauso verwenden. Super, oder? Nun ja, dahinter steht natürlich kein Hexenwerk, sondern eine recht umfangreiche Komponente. Diese nutzt die verschiedenen Möglichkeiten des Internet Explorers ab Version 5. Insbesondere kommt ein editierbares `IFRAME` sowie jede Menge JavaScript zum Einsatz. Es verwundert daher nicht, dass der an den Client übertragene Quelltext ein „wenig“ umfangreicher ist als die originale ASP.NET-Seite. Die zweite Abbildung zeigt Notepad mit einem kleinen Auszug der Seite.

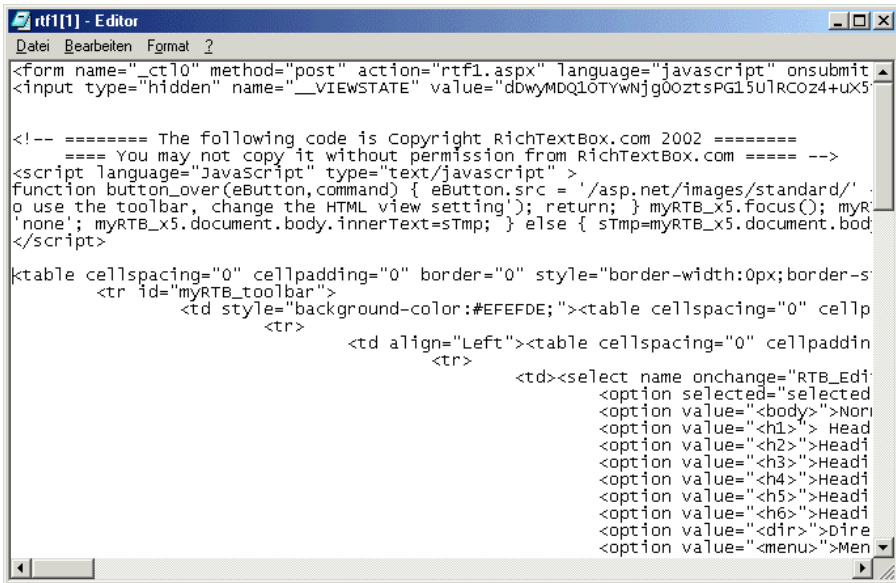


Abbildung 6.33 Der übertragene Quelltext der Seite ist ziemlich umfangreich.

Das Control wird vom Hersteller RichTextBox.com angeboten und unter der gleichnamigen Internet-Adresse vertrieben. Kostenlos ist das Control also nicht, aber dafür sehr leistungsfähig. Eine Vielzahl von in der Dokumentation ausführlich beschriebenen Eigenschaften ermöglicht es, das Eingabefeld sehr individuell anzupassen. Das Ergebnis wird in jedem Fall als HTML-Stream geliefert, der in einer Datenbank abgespeichert oder anderweitig verarbeitet werden kann. Eine Auswertung als Text im „echten“ Format RTF ist hingegen bisher noch nicht vorgesehen.

Weitere Informationen sowie eine Demoversion erhalten Sie direkt beim Hersteller unter folgender Adresse:

<http://www.richtextbox.com>

6.23 ... eine sortierte ListBox anzeigen?

Ganz ehrlich? Ich weiß nicht, warum die Entwickler bei Microsoft nicht an eine Sortierung zumindest der beiden Controls `ListBox` und `DropDownList` gedacht haben. Es wäre doch wirklich absolut trivial. Nun gut, wie immer heißt es auch in diesem Fall: „Selbst ist der Mann“ oder die Frau.

Die Implementierung einer sortierten `ListBox` ist nicht weiter schwierig, bedarf aber einer Ableitung der Klasse. Ich stelle mir dazu zwei Eigenschaften vor. Über `SortMode` kann die Art der Sortierung gewählt werden. Mögliche Werte der gleichnamigen Enumeration sind `None` für keine Sortierung sowie `Text` und `Value` zur Sortierung nach dem angezeigten Text oder dem internen Wert der Listeneinträge. Die zweite Eigenschaft `SortDirection` bestimmt, ob die Sortierung auf- oder absteigend erfolgen soll.

Damit die Funktionalität wieder verwendet werden kann, habe ich eine Schnittstelle `ISortedListControl` angelegt, die die notwendigen Mitglieder definiert. Diese Schnittstelle sowie die Enumerationen für die zwei vorgestellten Eigenschaften sehen Sie im folgenden ersten Teil der C#-Quellcode-Datei.

Listing 6.43 SortedListBox.cs – Teil 1

```
public enum ListSortMode
{
    None,
    Text,
    Value
}

public enum ListSortDirection
{
    Asc,
    Desc
}

internal interface ISortedListControl
{
    ListItemCollection Items
    {
        get;
    }
    ListSortMode SortMode
    {
        get;
        set;
    }
    ListSortDirection SortDirection
    {
        get;
        set;
    }
}
```

Das Control selbst ist schnell implementiert. Es handelt sich wie beschrieben um eine Ableitung der Klasse `ListBox`. Zusätzlich wird die gezeigte Schnittstelle `ISortedListControl` unterstützt, so dass die beiden Eigenschaften `SortMode` und `SortDirection` implementiert werden müssen.

Listing 6.44 SortedListBox.cs – Teil 2

```
public class SortedListBox : ListBox, ISortedListControl
{

    public SortedListBox() : base()
    {
    }

    protected override void CreateChildControls()
    {
        if (SortMode != ListSortMode.None)
        {
            ArrayList ctllist = new ArrayList(this.Items);
            ctllist.Sort(new ListItemComparer(this));

            this.Items.Clear();
            foreach (ListItem ctl in ctllist)
            {
                this.Items.Add(ctl);
            }
        }
        this.CreateControlCollection();
    }

    public ListSortMode SortMode
    {
        get
        {
            if (ViewState["SortMode"] != null)
                return ((ListSortMode) ViewState["SortMode"]);
            else
                return (ListSortMode.Text);
        }
        set { ViewState["SortMode"] = value; }
    }

    public ListSortDirection SortDirection
    {
        get
        {
            if (ViewState["SortDirection"] != null)
                return ((ListSortDirection) ViewState["SortDirection"]);
            else
                return (ListSortDirection.Ascending);
        }
        set { ViewState["SortDirection"] = value; }
    }
}
```

```
        return(ListSortDirection.Asc);  
    }  
    set { ViewState["SortDirection"] = value; }  
    }  
}
```

Die Arbeit wird in der überschriebenen und als `protected` markierten Methode `CreateChildControls` verrichtet. Auf Basis der bereits vorhandenen `ItemsCollection` wird eine `ArrayList` angelegt. Diese wird sortiert, anschließend wird die bisherige `Collection` gelöscht, und die einzelnen Elemente werden in der korrekt sortierten Reihenfolge wieder angefügt. Zur Sortierung wird die Klasse `ListItemsComparer` verwendet, die die Schnittstelle `Comparer` unterstützt. Mangels Vorlage musste auch diese Klasse neu geschrieben werden, was sich allerdings nicht als sonderlich umfangreich erwiesen hat.

Listing 6.45 SortedListBox.cs – Teil 3

```
internal class ListItemComparer : IComparer  
{  
    ISortedListControl _owner;  
  
    private ListItemComparer()  
    {  
    }  
  
    public ListItemComparer(ISortedListControl owner)  
    {  
        _owner = owner;  
    }  
  
    public int Compare(object x, object y)  
    {  
        int retVal = 0;  
  
        ListItem _x = (ListItem) x;  
        ListItem _y = (ListItem) y;  
  
        if(_owner.SortMode == ListSortMode.Text)  
            retVal = string.Compare(_x.Text, _y.Text);  
        else  
            retVal = string.Compare(_x.Value, _y.Value);  
  
        if(_owner.SortDirection == ListSortDirection.Desc)  
            retVal *= -1;  
  
        return(retVal);  
    }  
}
```

So viel zur grauen Theorie. Wollen Sie das neue Control im Einsatz sehen? Ich schon! Vorher muss dieses jedoch noch mittels `csc.exe` kompiliert und die so erzeugte DLL im bin-Verzeichnis der Web-Applikation abgelegt werden. Anschließend kann das nachfolgende Beispiel ausprobiert werden. Es enthält eine `ListBox` mit den Einträgen aus der Ländertabelle. Über zwei `RadioButtonList`-Controls kann die Sortierart und deren -richtung bestimmt werden.

Listing 6.46 SortedListBox1.aspx

```
<% @Import Namespace="PAL.WebControls" %>
<% @Register TagPrefix="PAL" Namespace="PAL.WebControls"
    Assembly="SortedListBox" %>

<script language="C#" runat=server>

...

void UpdateView(object sender, EventArgs e)
{
    string sortmode = tbl_sortmode.SelectedItem.Value;
    countrylist.SortMode = (ListSortMode)
        Enum.Parse(typeof(ListSortMode), sortmode);

    string sortdirection = tbl_sortdirection.SelectedItem.Value;
    countrylist.SortDirection = (ListSortDirection)
        Enum.Parse(typeof(ListSortDirection), sortdirection);
}

</script>

<form runat="server">

<p> Bitte wählen Sie Ihr Land aus: <br>
    <PAL:SortedListBox id="countrylist" runat="server"
        DataValueField="ID"
        DataTextField="TextDE"
        Rows="10"/>
</p>

<p> Sortierart:
    <ASP:RadioButtonList id="tbl_sortmode" runat="server"
        AutoPostBack="true"
        OnSelectedIndexChanged="UpdateView"
        RepeatColumns="3">
        <ASP:ListItem Value="None" Text="Ohne"/>
        <ASP:ListItem Value="Text" Text="Text-Feld" Selected="true"/>
        <ASP:ListItem Value="Value" Text="Value-Feld"/>
    </ASP:RadioButtonList>
```

```
</p>

<p> Sortierrichtung:
<ASP:RadioButtonList id="tbl_sortdirection" runat="server"
  AutoPostBack="true"
  OnSelectedIndexChanged="UpdateView"
  RepeatColumns="2">
  <ASP:ListItem Value="Asc" Text="Aufwärts" Selected="true"/>
  <ASP:ListItem Value="Desc" Text="Abwärts"/>
</ASP:RadioButtonList>
</p>

</form>
```

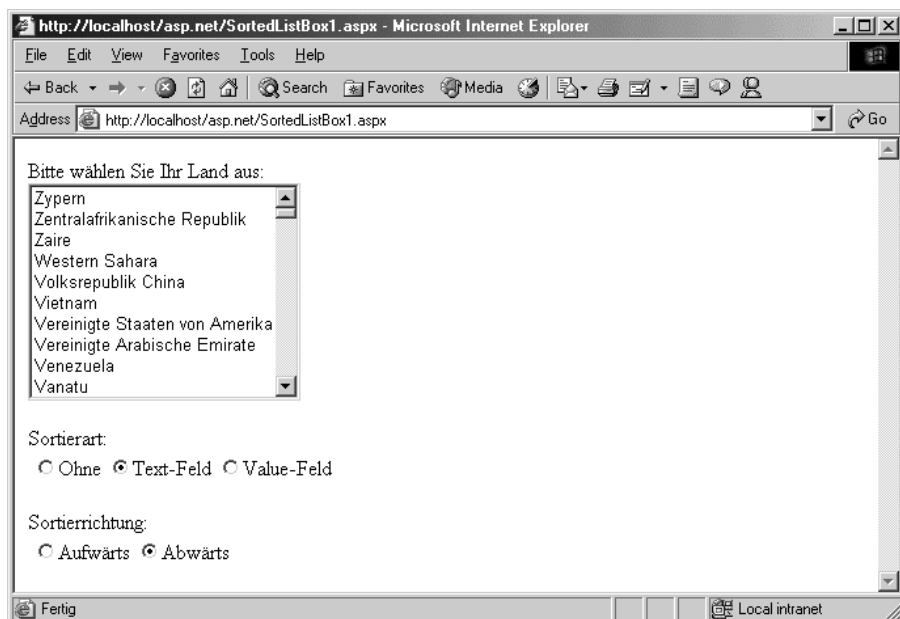


Abbildung 6.34 Der Inhalt der ListBox wird automatisch sortiert.



Aus Gründen der Performance sollten Sie überlegen, wann Sie automatisch Sortierung verwenden und wann Sie diese Aufgabe lieber direkt einer Datenbank-Engine überlassen. Im vorliegenden Fall wäre eine ORDER BY-Query sicherlich effektiver.



Selbstverständlich finden Sie das Listing auf der begleitenden Buch-CD. Die dort abgelegte Version enthält analoge Ableitungen für die drei an-

deren List-Controls `DropDownList`, `CheckBoxList` und `RadioButtonList`. Auch diese können Sie in Zukunft also automatisch sortiert anzeigen.

6.24 ... einen Listeneintrag mit einem bestimmten Wert selektieren?

Sie kennen das bestimmt. Sie setzen eine `ListBox` oder auch eine `DropDownList` ein und wollen einen Eintrag mit einem bekannten Wert selektieren. Was tun? Eine Schleife durch alle Einträge ist eine Möglichkeit, die sicher naheliegend, aber dennoch nicht so schön wie das eingebaute Gegenstück ist. Alle Listen-Controls unterstützen durch die untergeordnete `ListItemCollection` eine Methode `FindByValue`. Zurückgeliefert wird der erste Eintrag, der den übergebenen Wert aufweist. Sofern ein solcher nicht existiert, wird `null` geliefert.

Das Beispiel zeigt den Einsatz in Verbindung mit einem datengebundenen `DropDownList`-Control, das den Inhalt der Länderliste anzeigt. Ein Land soll vorselektiert werden. Die Abbildung zeigt, dass das gewünschte Land mit der ID „D“ beim ersten Aufruf der Seite bereits vorselektiert ist.

Listing 6.47 FindByValue1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\countries.mdb");
    conn.Open();

    string SQL = "SELECT ID, TextDe FROM Countries;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    countrylist.DataSource =
        cmd.ExecuteReader(CommandBehavior.CloseConnection);

    if(!IsPostBack)
    {
        countrylist.DataBind();
        ListItem li = countrylist.Items.FindByValue("D");
        if(li != null) li.Selected = true;
    }
}

</script>

<form runat="server">
```

```
<p> Bitte wählen Sie Ihr Land aus:  
  
<ASP:DropDownList id="countrylist" runat="server"  
    DataTextField="TextDE"  
    DataValueField="ID"/>  
  
</p>  
  
</form>
```

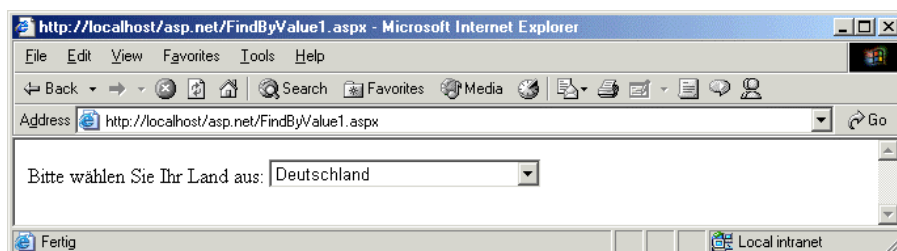


Abbildung 6.35 Das Element würde über den Inhalt gefunden.

Analog zu `FindByValue` können Sie auch `FindByText` verwenden, um das Element mit dem übergebenen Text zu finden. Im obigen Fall müssten Sie also „Deutschland“ übergeben. Sie können selbstverständlich nicht nur die Selektion vornehmen, vielmehr haben Sie vollen Zugriff auf alle Eigenschaften des herausgesuchten Listeneintrags, so dass Sie beispielsweise auch den internen Wert oder den angezeigten Text ändern können.

6.25 ... verhindern, dass die Selektion beimPostBack zurückgesetzt wird?

Bei dieser Frage handelt es sich um eine echte FAQ, die in der Microsoft-Newsgruppe zahlreich in unterschiedlichen Variationen gestellt wurde. Nachfolgend demonstriere ich Ihnen das Problem anhand der Länderliste und einem ListBox-Control. Das Listing sieht auf den ersten Blick in Ordnung aus, und die Liste wird auch korrekt angezeigt.

Listing 6.48 IsPostBack1.aspx

```

<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\countries.mdb");
    conn.Open();

    string SQL = "SELECT ID, TextDe FROM Countries;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    countrylist.DataSource =
        cmd.ExecuteReader(CommandBehavior.CloseConnection);

    countrylist.DataBind();
}

</script>

<form runat="server">

<p> Bitte wählen Sie Ihr Land aus: <br>
    <ASP:ListBox id="countrylist" runat="server"
        DataTextField="TextDE"
        DataValueField="ID"/>
</p>

<p> <ASP:Button runat="server" Text="OK"/> </p>

</form>

```

Die Abbildung zeigt das Beispiel nach der Auswahl eines Eintrags und einem Klick auf den Button. Ganz offensichtlich ist von der Selektion nichts mehr zu sehen. Diese wurde durch den PostBack zurückgesetzt. Dies bestätigt auch die Ausgabe der ListBox-Eigenschaft `SelectedIndex`. Dieser liefert -1, also keine Selektion.

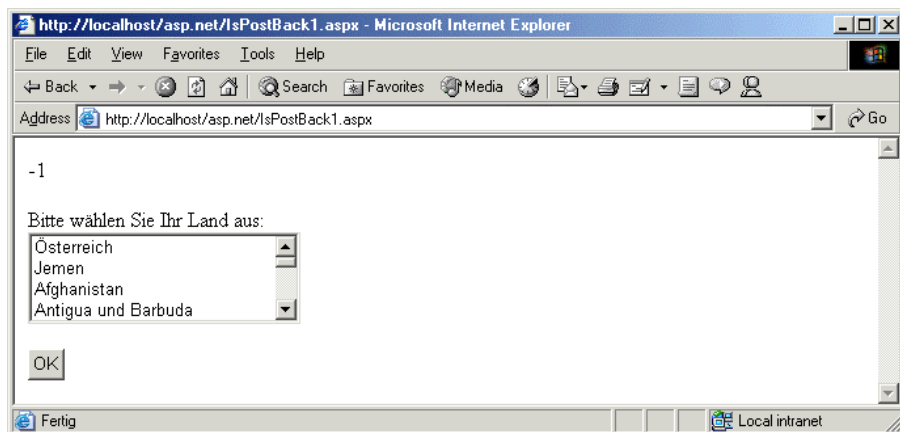


Abbildung 6.36 Die Selektion geht nach dem PostBack verloren.

Die Lösung für dieses Problem ist genauso einfach wie leicht zu übersehen und zu vergessen. Die Datenbindung spielt Ihnen einen Streich, und zwar, weil sie unbedingt in der Behandlung des Page_Load-Ereignisses aufgerufen wird. Somit wird das ListBox-Control bei jedem Laden der Seite komplett neu gebunden und neu aufgebaut. Dass die Benutzerauswahl dabei verloren geht, ist nur logisch.

Abhilfe schafft eine bedingte Datenbindung unter Verwendung der Page-Eigenschaft `IsPostBack`. Über den zurückgelieferten booleschen Wert kann abgefragt werden, ob es sich um den ersten Aufruf der Seite handelt oder um einen nachfolgenden PostBack. Das in dieser Hinsicht erweiterte Beispiel funktioniert korrekt. Die Bindung wird nur beim ersten Aufruf durchgeführt, wozu die Eigenschaft negiert werden muss.

Listing 6.49 `IsPostBack2.aspx`

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\countries.mdb");
    conn.Open();

    string SQL = "SELECT ID, TextDe FROM Countries;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    countrylist.DataSource =
        cmd.ExecuteReader(CommandBehavior.CloseConnection);

    if(!IsPostBack)
    {
```

```

countrylist.DataBind();
}
}

void bt_click(object sender, EventArgs e)
{
    int index = countrylist.SelectedIndex;
    Response.Write(index.ToString());
}

</script>

...

```

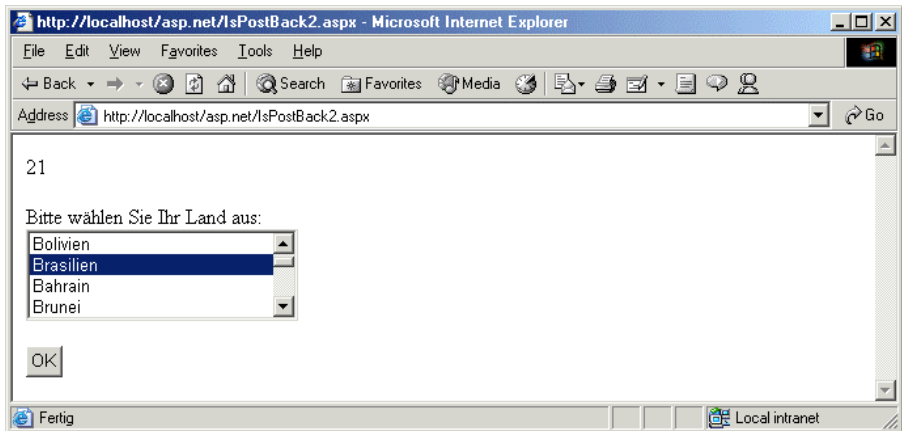


Abbildung 6.37 Der gewählte Eintrag Nummer 21 ist auch nach dem PostBack selektiert.

6.26 ... verhindern, dass ein File-Upload immer null ergibt?

Die Möglichkeit, auf einfache Art und Weise Dateien vom Client auf den Server zu übertragen, ist ein kleiner, aber angenehmer Pluspunkt von ASP.NET. Das Control, das die Arbeit übernimmt, hört auf den Namen `HtmlInputFile` und gehört somit der Gattung der HTML Server Controls an.

Ich möchte Sie an dieser Stelle nicht mit der prinzipiellen Verwendung des Controls langweilen. Allerdings möchte ich Ihnen zeigen, warum der Einsatz manchmal schlicht nicht klappen will und die `PostedFile`-Eigenschaft unmissverständlich `null` liefert. Im folgenden Beispiel ist genau das der Fall.

Listing 6.50 HtmlInputFile1.aspx

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    HttpPostedFile file = inputfile.PostedFile;

    if(file != null)
        lb.Text = "Vielen Dank!";
    else
        lb.Text = "Sie haben keine Datei ausgewählt ...";
}

</script>

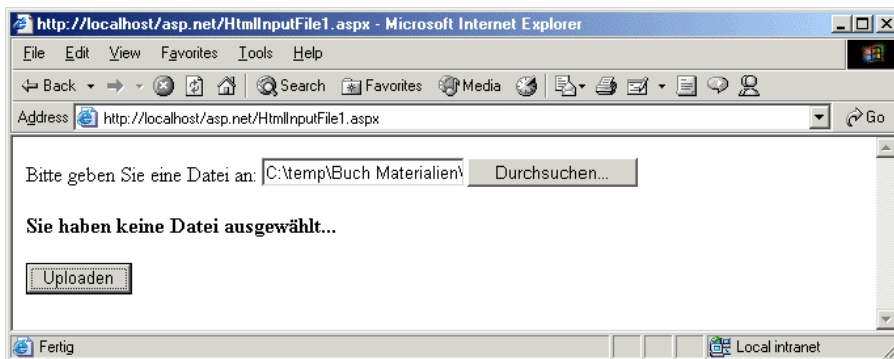
<form runat="server">

<p> Bitte geben Sie eine Datei an:
    <input type="file" id="inputfile" runat="server"/>
</p>

<p><b> <ASP:Label id="lb" runat="server"/> </b></p>

<p>
    <ASP:Button runat="server" Text="Uploaden" OnClick="bt_click"/>
</p>

</form>
```

**Abbildung 6.38** Grimmel, das habe ich wohl ...

Obwohl das Listing korrekt aussieht, funktioniert es ganz offensichtlich nicht. Der Grund hierfür ist die Übertragung der Formulardaten. Diese muss MIME-kodiert und in mehreren Teilen erfolgen: ein Teil für die reinen Textdaten möglicher ande-

rer Eingabefelder und n weitere Teile für die einzelnen Dateien oder wie in diesem Fall nur eine. Standardmäßig werden jedoch alle Daten in ein MIME-Paket geschnürt, und daher kann die übertragene Datei nicht herausgefiltert werden. Die Lösung für das Problem ist wirklich simpel, denn Sie müssen lediglich einen alternativen Übertragungstyp angeben. Dies geschieht im Rahmen des (serverseitigen) Formulars:

Listing 6.51 HtmlInputFile2.aspx

```
...

<form runat="server" enctype="multipart/form-data">

    <p> Bitte geben Sie eine Datei an:
    <input type="file" id="inputfile" runat="server"/>
    </p>

...
```

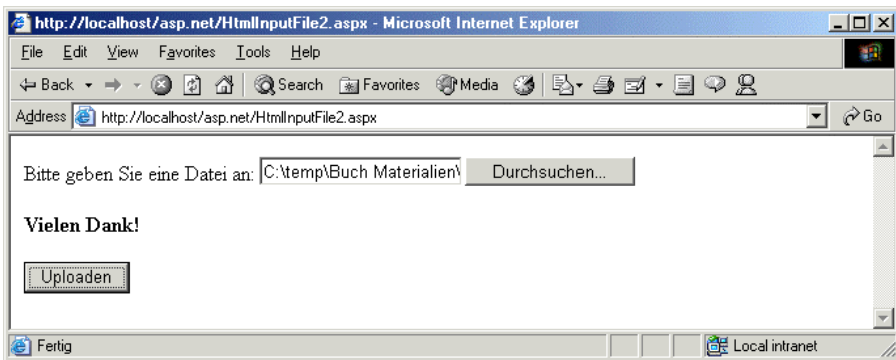


Abbildung 6.39 Na also, es geht doch!

6.27 ... eine TextBox mit einem Farbverlauf versehen?

Diese Frage hat an sich mit ASP.NET reichlich wenig zu tun, taucht aber dennoch erstaunlich oft auf. Die Aufgabenstellung lässt sich sehr einfach mit Cascading Style Sheets lösen. Grundlage ist eine Grafik, die den gewünschten Verlauf enthält. Über das Style-Attribut wird diese als Hintergrund des TextBox-Controls angegeben. Das Listing zeigt's.

Listing 6.52 GradientTextBox1.aspx

```
<form runat="server">

  <ASP:TextBox runat="server"
    Font-Name="Verdana"
    Font-Bold="true"
    Style="background: url('gradient.gif');"
    Text="Hallo Welt"/>

</form>
```

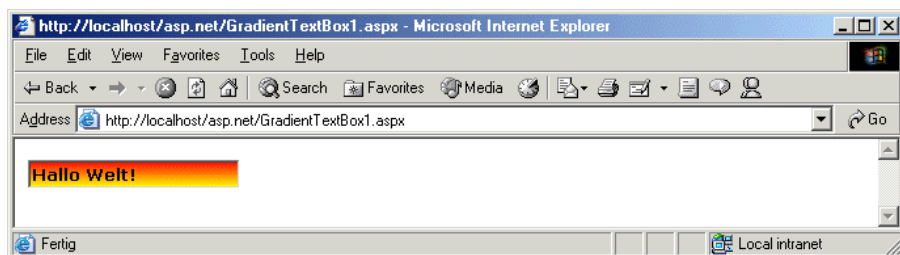


Abbildung 6.40 Der Hintergrund des Eingabefeldes besteht aus einem Farbverlauf.

Mit ein wenig Fantasie können Sie übrigens ganz erstaunliche Effekte zaubern. So können Sie mit Hilfe einer animierten Grafik den Hintergrund eines TextBox-Controls noch mehr aufpeppen. Nachfolgend ein kleines Beispiel, wenngleich die Animation in gedruckter Form nicht wirklich eindrucksvoll ist.

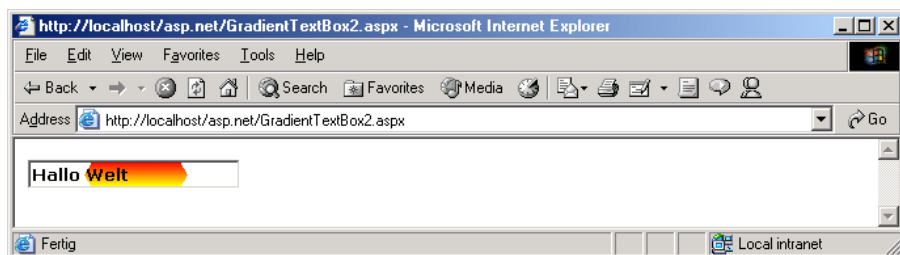


Abbildung 6.41 Auch Animationen sind möglich.

Weitere Beispiele in dieser Richtung finden Sie auch bei Dr.Web:

<http://www.drweb.de>

6.28 ... die Zeichenlänge einer TextBox limitieren?

Standardmäßig ist die Eingabelänge einer TextBox nicht beschränkt. In manchen Fällen ist dies jedoch wichtig, insbesondere wenn die Inhalte in eine Datenbank geschrieben werden, wo eine maximale Länge festgelegt ist. In diesem Fall können Sie die Eigenschaft `MaxLength` auf den gewünschten Wert setzen.

Listing 6.53 MaxLength1.aspx

```
<form runat="server">

  <p> Bitte geben Sie Ihren Vornamen ein (max. 20 Zeichen):<br><br>
  <ASP:TextBox id="tb" runat="server" MaxLength="20" />
</p>

</form>
```

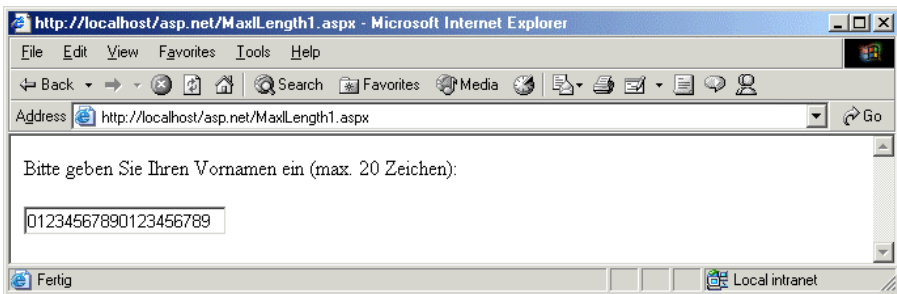


Abbildung 6.42 Es können maximal 20 Zeichen eingegeben werden.

Beachten Sie bitte, dass die Eigenschaft bei mehrzeiligen Eingabefeldern (`TextMode=TextBoxMode.MultiLine`) keine Auswirkungen hat. In diesem Fall können Sie das `onChange`-Ereignis clientseitig mittels JavaScript behandeln. Stellen Sie hier fest, dass die zulässige Länge überschritten wird, schneiden Sie den überflüssigen Teil einfach ab.

Listing 6.54 MaxLength2.aspx

```
<form runat="server">

  <p> Bitte geben Sie Ihren Vornamen ein (max. 20 Zeichen):<br><br>
  <ASP:TextBox id="tb" runat="server"
    TextMode="MultiLine"
    onChange="if (this.value.length>20)
      this.value=this.value.substring(0,20);"/>
</p>

</form>
```

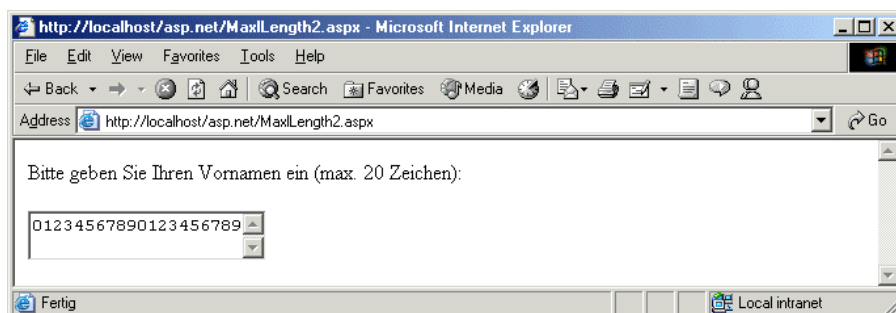


Abbildung 6.43 Auch das mehrzeilige Eingabefeld nimmt maximal 20 Zeichen auf.

6.29 ... Zahlen und Dati in Vorlagen formatieren?

Die statische Methode `DataBinder.Eval` wird insbesondere innerhalb von Vorlagen bei den Data-Controls `Repeater`, `DataList` und `DataGrid` verwendet. Der Methode wird das `DataItem` des Containers sowie das abzufragende Feld übergeben, dessen Inhalt zurückgeliefert wird. In vielen Fällen reicht es aus, die Daten wie vorhanden auszugeben. Manchmal ist es aber erforderlich, diese individuell zu formatieren. Die Abbildung zeigt einige Datensätze, die in einem `DataList`-Control ausgegeben werden sollen.

NumericDate : Tabelle			
	ID	Numeric	Date
▶	1	4545	01.01.2002
	2	5,555	30.06.2002 12:50:00
	3	1,1	31.07.2002 12:40:15
*	(AutoWert)	0	

Datensatz: 1 von 3

Abbildung 6.44 Die Datensätze der Tabelle `NumericDate`

Listing 6.55 `DataBinder1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb");
    conn.Open();
}
```

```
string SQL = "SELECT * FROM NumericDate";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
dl.DataSource = cmd.ExecuteReader();

DataBind();

conn.Close();
}

</script>

<ASP:DataList id="dl" runat="server">

  <ItemTemplate>
    Zahl:
    <%=# DataBinder.Eval(Container.DataItem, "Numeric") %>

    <br> Datum:
    <%=# DataBinder.Eval(Container.DataItem, "Date") %>
  </ItemTemplate>

</ASP:DataList>
```

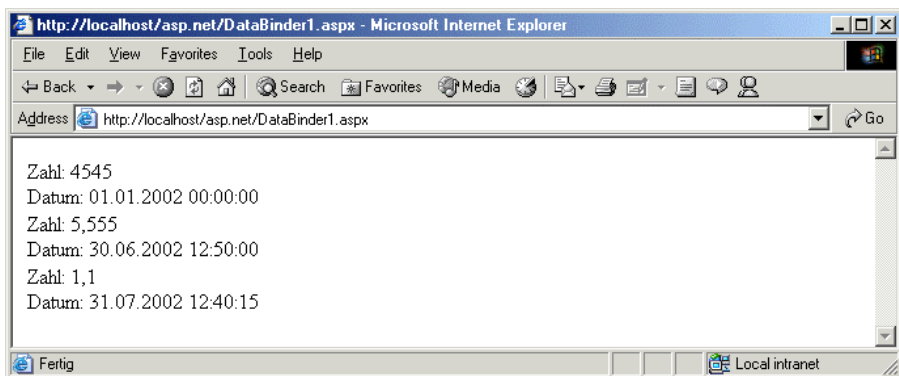


Abbildung 6.45 Die Ausgabe sieht aus wie Kraut und Rüben.

Das Listing zeigt den Zugriff auf die Beispieltabelle und deren Ausgabe in einem DataList-Control. Das hinterlegte ItemTemplate fragt die Daten mit Hilfe der Methode DataBinder.Eval ab. Wie die Abbildung deutlich zeigt, ist das Ergebnis wenig befriedigend. Zum Glück bietet die Methode eine Möglichkeit, individuelle Formatierungen zuzuweisen, ganz so, wie Sie es von der Methode string.Format kennen. Die gewünschte Formatierungszeichenkette übergeben Sie schlicht als dritten Parameter.

Die Ausgabe des aktualisierten Listings ist schon etwas übersichtlicher. Alle Zahlen haben zwei Nachkommastellen, und das Datumsfeld wird getrennt in Datum und Uhrzeit angegeben. Änderungen am Quelltext waren dazu nicht notwendig.

Listing 6.56 DataBinder2.aspx

```
...  
  
<ASP:DataList id="dl" runat="server">  
  
  <ItemTemplate>  
    Zahl:  
    <%=# DataBinder.Eval(Container.DataItem, "Numeric", "{0:0.00}") %>  
  
    <br> Datum:  
    <%=# DataBinder.Eval(Container.DataItem, "Date", "{0:d}", Uhrzeit:  
{0:t}") %>  
  </ItemTemplate>  
  
</ASP:DataList>
```

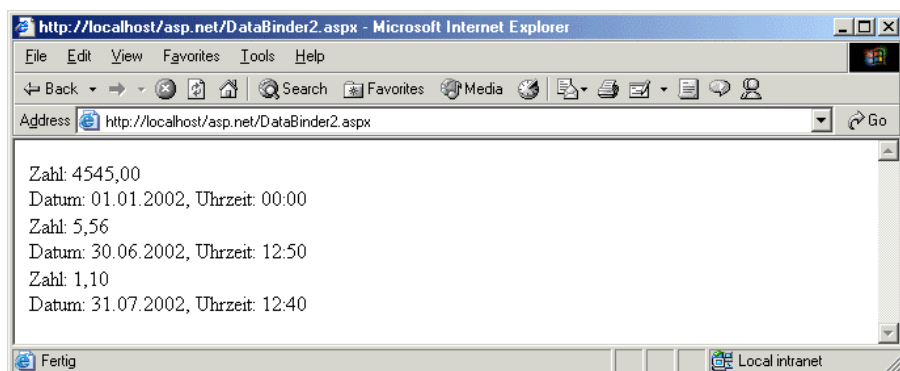


Abbildung 6.46 Mit Formatierung sieht die Ausgabe viel übersichtlicher aus.

6.30 ... die Datensatz-ID unsichtbar in einem DataGrid-Control mitführen?

Um einen Datensatz eindeutig identifizieren zu können, wird in Datenbanken ein eindeutiger Primärindex vergeben. Oftmals handelt es sich um einen automatisch entweder inkrementell oder zufällig erstellten Zahlwert. Aber auch individuelle

Schlüssel sind möglich. In aller Regel handelt es sich jedoch um ein internes Feld, das dem Benutzer nicht angezeigt werden soll. Die ID sollte daher nach Möglichkeit unsichtbar mitgeführt werden können.

Die von der gemeinsamen Basis `BaseDataList` abgeleiteten Controls `DataList` und `DataGrid` verfügen über eine Eigenschaft `DataKeyField`. Hier kann bei der Definition des Controls der Name des ID-Feldes zugeordnet werden. Über den Index der einzelnen Zeilen kann die ID später mit Hilfe der von der Eigenschaft `DataKeys` zurückgelieferten `DataKeyCollection` abgefragt werden.

Das Beispiel zeigt die Verwendung der Technik. Das `DataGrid`-Control wird an eine Datenquelle gebunden, aus der zwei Felder explizit in Form von `ButtonColumn`-Spalten angezeigt werden. Das dritte Feld wird der Eigenschaft `DataKeyField` zugewiesen. Ein Klick auf einen der Buttons löst das zentrale `ItemCommand`-Ereignis aus. Hier wird die ID des gewählten Datensatzes abgefragt und ausgegeben.

Listing 6.57 DataKey1.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    dg.DataSource = dataset;
    if(!IsPostBack) DataBind();
}

void dg_ItemCommand(object sender, DataGridCommandEventArgs e)
{
    int index = e.Item.ItemIndex;
    int id = (int) dg.DataKeys[index];
    Response.Write("Datensatz-ID: " + id.ToString());
}
```

```
}  
  
</script>  
  
<form runat="server">  
  
<ASP:DataGrid id="dg" runat="server"  
  AutoGenerateColumns="false"  
  DataKeyField="ID"  
  OnItemCommand="dg_ItemCommand">  
  
  <Columns>  
  
    <ASP:ButtonColumn  
      HeaderText="Vorname"  
      DataTextField="Firstname"/>  
  
    <ASP:ButtonColumn  
      HeaderText="Nachname"  
      DataTextField="Lastname"/>  
  
  </Columns>  
  
</ASP:DataGrid>  
  
</form>
```

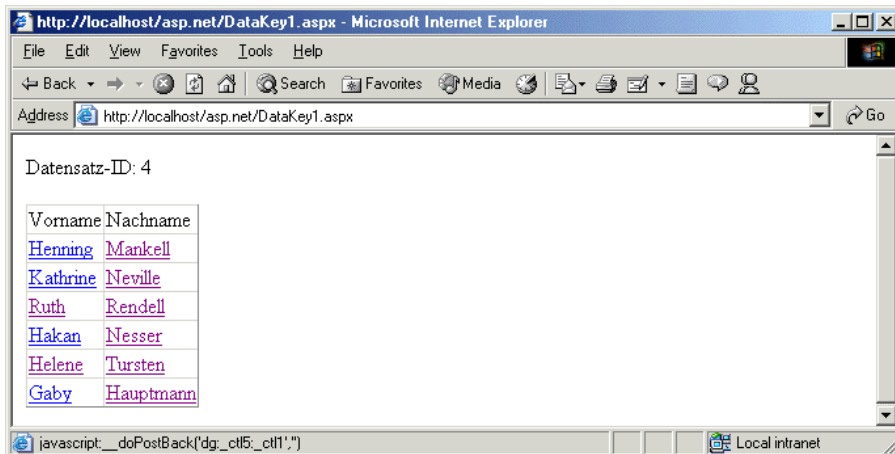


Abbildung 6.47 Die ID wird unsichtbar mitgeführt.

Die Ausgabe des Beispiels zeigt, dass das System funktioniert; beim Klick wird die korrekte ID ausgegeben. Dies ist insbesondere sehr nützlich, wenn die Daten interaktiv durch den Benutzer in der Datenbank geändert werden sollen. In diesem Fall ist ein Zugriff auf die ID unerlässlich.

Alternativ könnte die ID auch in einer zusätzlichen Spalte abgelegt werden, die dem Benutzer mittels `visible="false"` vorenthalten wird. In diesem Fall müsste jedoch auch der Zugriff auf die ID leicht angepasst werden. Empfehlenswert ist das Vorgehen indes nicht, wenngleich sich das Ergebnis im Browser nicht unterscheidet.

Listing 6.58 DataKey2.aspx

```
...

void dg_ItemCommand(object sender, DataGridCommandEventArgs e)
{
    int id = int.Parse(e.Item.Cells[0].Text);
    Response.Write("Datensatz-ID: " + id.ToString());
}

...

<ASP:BoundColumn
    DataField="ID"
    Visible="false"/>

...
```

6.31 ... die automatisch erstellten Spalten eines DataGrid-Controls anpassen?

Die standardmäßig eingeschaltete Eigenschaft `AutoGenerateColumns` sorgt dafür, dass für die zugewiesenen Daten automatisch entsprechende Spalten für das DataGrid-Control angelegt werden. Als Entwickler haben Sie scheinbar keinerlei Möglichkeit, explizit Einfluss auf die weitere Darstellung zu nehmen. Oder etwa doch? Ja, denn das Control bietet zwei Ereignisse an, über die Sie individuell eingreifen können. Das Ereignis `ItemCreated` wird ausgelöst, sobald eine Zeile samt den enthaltenen Spalten angelegt wurde. Das zweite Ereignis `ItemDataBound` wird *nach* der Bindung an die Datenquelle, aber noch vor dem Zeichnen des Controls ausgelöst.

ItemCreated

Das erste Ereignis `ItemCreated` eignet sich insbesondere, um die Tabellenzeile oder -spalte, nicht aber deren Inhalt anzupassen. Das Beispiel zeigt dies. Über das Ereignis wird hier die Hintergrundfarbe des Controls gesetzt. Diese Aufgabe könnte auch anders gelöst werden, zeigt aber deutlich, wie das Control intern funktioniert. Über die Ereignisargumente wird eine `DataGridItem`-Instanz geliefert, die die aktuelle Zeile repräsentiert. Hier können zahlreiche Informationen abgefragt werden, darunter auch die Art der Zeile wie `Header`, `Footer`, `Item`, `AlternatingItem` und so weiter. Daneben erhalten Sie auch Zugriff auf die Datenquelle sowie – wie im Beispiel zu sehen – die einzelnen Zellen.

Listing 6.59 ItemCreated1.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>
<% @Import Namespace="System.Drawing" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    dg.DataSource = dataset;
    if(!IsPostBack) DataBind();
}

void dg_ItemCreated(object sender, DataGridItemEventArgs e)
{
    if(e.Item.ItemType == ListItemType.Item)
    {
        TableCell FirstCell = e.Item.Cells[0];
        FirstCell.BackColor = Color.Red;
    }
}
```



```

</script>

<ASP:DataGrid id="dg" runat="server"
OnItemCreated="dg_ItemCreated"/>

```

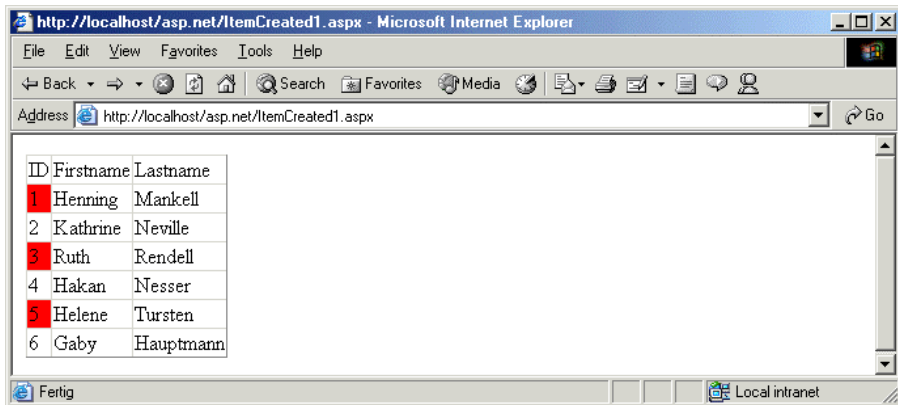


Abbildung 6.48 Die Hintergrundfarbe wurde über das Ereignis geändert.

ItemDataBound

Das zweite Ereignis wird *nach* der Bindung an die Datenquelle ausgelöst. Im Unterschied zu `ItemCreated` stehen daher die tatsächlichen Daten und nicht nur das Grundgerüst zur Verfügung. Diese können nun sehr einfach angepasst werden. Hierzu greifen Sie auf die Eigenschaft `Text` der jeweiligen `TableCell`-Instanz zu, die jede einzelne Zelle der Zeile repräsentiert. Im Beispiel wird auf diese Weise jede Zelle der ersten Spalte mit einigen zusätzlichen Daten versehen. Darunter auch der fortlaufende, 0-basierte Index der Zeile. Zusätzliche Zeilen wie der Kopf haben den Wert `-1`.

Listing 6.60 ItemDataBound1.aspx

```

...

void dg_DataBound(object sender, DataGridItemEventArgs e)
{
    TableCell FirstCell = e.Item.Cells[0];
    FirstCell.Text = "ID: " + FirstCell.Text + ", Index: " +
        e.Item.ItemIndex.ToString();
}

</script>

```

```
...  
<ASP:DataGrid id="dg" runat="server"  
  OnItemDataBound="dg_DataBound"/>  
...
```

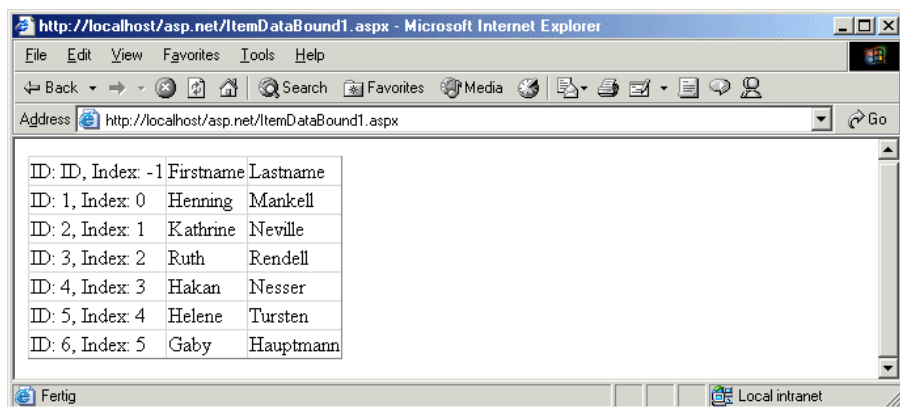


Abbildung 6.49 Die gebundenen Daten wurden individuell angepasst.

6.32 ... eine datengebundene DataGrid-Spalte individuell formatieren?

Manchmal ist es erforderlich, eine datengebundene Spalte individuell zu formatieren oder zu ergänzen. Denken Sie beispielsweise an eine Preisangabe, bei der der numerische Datenbankinhalt mit dem korrekten Währungssymbol versehen werden soll.

Die Quelle eines Feldes wird über die Eigenschaft `DataField` zugewiesen. Optional kann eine Formatierung angegeben werden, die der Eigenschaft `DataFormatString` übergeben wird. Hier kann ein Formatierungsstring analog zu dem bekannten `string.Format` eingesetzt werden. Intern verwendet das `DataGrid`-Control eben diese Methode und übergibt das aktuelle Datenfeld als Parameter 0.

Das Beispiel zeigt die Ergänzung einer Spalte mit zusätzlichem Text. Auf das Datenfeld wird über die Sequenz `{0}` zugegriffen. Hier können auch weitere numerische Formatierungen vorgenommen werden. Die Sequenz wurde erweitert, so dass das Feld `ID` mit führenden Nullen angezeigt wird.

Listing 6.61 ExtendField1.aspx

```

...

<form runat="server">

  <ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false">

    <Columns>

      <ASP:BoundColumn
        HeaderText="DB-ID"
        DataField="ID"
        DataFormatString="Die ID lautet: {0:0000} :-)" />

      <ASP:BoundColumn
        HeaderText="Vorname"
        DataField="Firstname" />

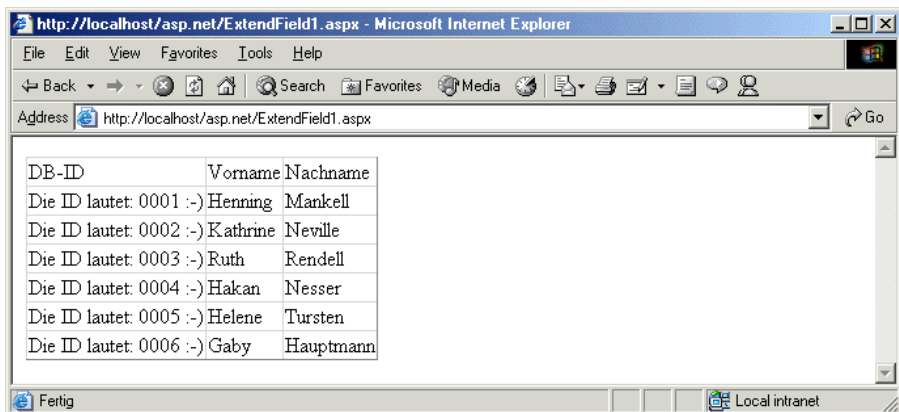
      <ASP:BoundColumn
        HeaderText="Nachname"
        DataField="Lastname" />

    </Columns>

  </ASP:DataGrid>

</form>

```

**Abbildung 6.50** Die Spalte ID wurde individuell formatiert.

6.33 ... mehrere Felder in einer DataGrid-Spalte anzeigen?

Die Anlage von datengebundenen Spalten ist beim DataGrid-Control mehr als einfach gelöst. Einfach den Namen des zugehörigen Datenfeldes und optional weitere Angaben wie Text der Kopfzeile hinterlegen, und schon ist die Spalte fertig. Auch die Zuweisung einer zusätzlichen Formatierungsanweisung ist möglich.

Das System hat leider einen Haken: Es kann maximal ein Datenfeld pro Spalte angezeigt werden. Dies ist nicht immer akzeptabel. Bei der Autorentabelle aus der Beispieldatenbank sind Vor- und Nachname getrennt hinterlegt. Dies ist vor dem Hintergrund der Atomisierung einer Datenbank sinnvoll. Dennoch ist die gemeinsame Ausgabe in einer Spalte „Name“ optisch ansprechender.

Variante I: TemplateColumn

Eine Möglichkeit, die Verknüpfung zweier Felder zu realisieren, ist der Verzicht auf die gebundene Spalte und die Verwendung einer vorlagenbasierten Spalte `TemplateColumn`. Hier bleibt die Implementierung der Spalte inklusive der Kopfzeile Ihnen überlassen. Über die gewohnte `DataBinding`-Syntax können Sie auf die Datenquelle zugreifen und diese nach Belieben verknüpfen.

Listing 6.62 MultipleFields1.aspx

```
...

<form runat="server">

  <ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false">

    <Columns>

      <ASP:BoundColumn
        HeaderText="DB-ID"
        DataField="ID"/>

      <ASP:TemplateColumn>

        <HeaderTemplate>
          Vorname und Nachname
        </HeaderTemplate>

        <ItemTemplate>
          <%= DataBinder.Eval(Container.DataItem, "Firstname") %>
          <%= DataBinder.Eval(Container.DataItem, "Lastname") %>
        </ItemTemplate>

      </Columns>

    </ASP:DataGrid>

  </form>
```

```

</ASP:TemplateColumn>

</Columns>

</ASP:DataGrid>

</form>

```

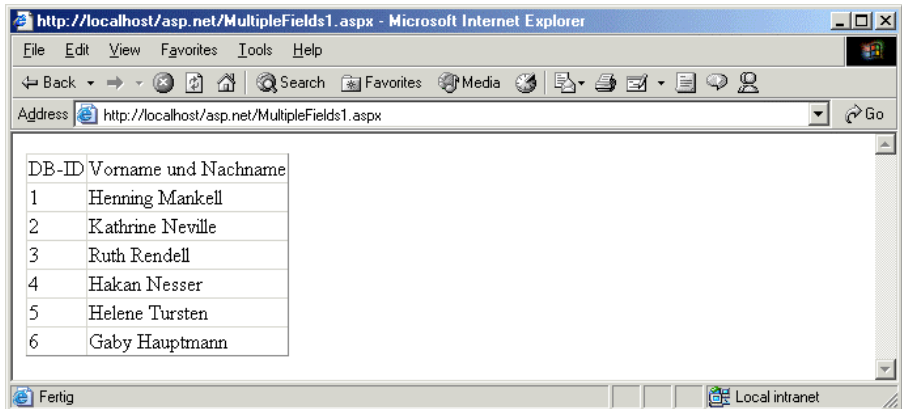


Abbildung 6.51 Die beiden Felder wurden in der Spaltenvorlage zusammengefasst.

Wie die Abbildung zeigt, funktioniert die Variante problemlos. Dennoch ist sie ein wenig umständlich. Insbesondere die quasi manuelle Nachbildung der Kopfzeile ist eigentlich überflüssige Arbeit. Drei weitere Varianten schaffen hier Abhilfe und erzeugen eins zu eins die gleiche Ausgabe im Browser.

Variante II: SQL

Die Abfragesprache SQL ist ausgesprochen leistungsfähig. Die dynamische Erstellung zusätzlicher Spalten auf Basis mehrerer bestehender ist noch die kleinste Herausforderung. Aber genau dies können Sie sich zur Anzeige von mehreren Feldern in einer DataGrid-Spalte zunutze machen.

Das Listing zeigt das abgeänderte Beispiel. Die Namensspalte greift nun auf das reguläre Datenfeld „Fullname“ zu, das zuvor dynamisch innerhalb der SQL-Query angelegt wurde.

Listing 6.63 MultipleFields2.aspx

```
...

string SQL = "SELECT ID, Firstname+' '+Lastname AS Fullname
FROM AUTHORS";
OleDbCommand cmd = new OleDbCommand(SQL, conn);

...

<form runat="server">

<ASP:DataGrid id="dg" runat="server"
AutoGenerateColumns="false">

<Columns>

<ASP:BoundColumn
HeaderText="DB-ID"
DataField="ID"/>

<ASP:BoundColumn
HeaderText="Vorname und Nachname"
DataField="Fullname"/>

</Columns>

</ASP:DataGrid>

</form>
```

Variante III: DataTable

Statt per SQL können Sie auch die `DataTable`-Instanz verwenden, um dynamisch eine Spalte anzulegen. Die Spalte wird hier über die Klasse `DataColumn` repräsentiert. Im Konstruktor kann der Name, der Datentyp sowie ein Ausdruck angegeben werden. Dieser Ausdruck fasst im Beispiel die beiden bestehenden Spalten „First-name“ und „Lastname“ zusammen. Die instanziierte Spalte wird der `Columns`-Collection der zuständigen `DataTable` angefügt.

Listing 6.64 MultipleFields3.aspx

```
...

string SQL = "SELECT * FROM AUTHORS";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
```

```

OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
DataSet dataset = new DataSet();
adapter.Fill(dataset, "Authors");
conn.Close();

DataColumn column = new DataColumn("Fullname",
    typeof(string), "Firstname+' '+Lastname");
DataTable table = dataset.Tables[0];
table.Columns.Add(column);

dg.DataSource = dataset;
if(!IsPostBack) DataBind();

...

```

Variante IV: ItemDataBound

Die vierte Möglichkeit, eine aus mehreren Datenfeldern bestehende Spalte in einem DataGrid-Control auszugeben, besteht aus dem ItemDataBound-Ereignis, das bereits in einem vorherigen Rezept vorgestellt wurde. Für die Namensspalte wird hier keinerlei Datenfeld angegeben. Die Daten werden erst im genannten Ereignis explizit zugewiesen.

Listing 6.65 MultipleFields4.aspx

```

...

void dg_DataBound(object sender, DataGridItemEventArgs e)
{
    if(e.Item.ItemType == ListItemType.Item ||
        e.Item.ItemType == ListItemType.AlternatingItem)
    {
        TableCell cell = e.Item.Cells[1];
        DataRowView row = (DataRowView) e.Item.DataItem;
        cell.Text = row["Firstname"] + " " + row["Lastname"];
    }
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        OnItemDataBound="dg_DataBound">

        <Columns>

```

```
<ASP:BoundColumn
  HeaderText="DB-ID"
  DataField="ID"/>

<ASP:BoundColumn
  HeaderText="Vorname und Nachname"/>

</Columns>

</ASP:DataGrid>

</form>
```



Beachten Sie unbedingt die Abfrage des Zeilentyps. Das Ereignis wird auch für zusätzliche Typen wie Kopf- und Fußzeile ausgelöst, hier ist die Datenquelle jedoch nicht gesetzt und entspricht `null`.

Alternativ zu dem Ereignis `ItemDataBound` könnten Sie im vorliegenden Fall übrigens auch das zweite Ereignis `ItemCreated` verwenden. Die der Spalte hier übergebenen Daten würden nicht überschrieben, da der Spalte keinerlei Datenfeld zugewiesen wurde.

Variante V: Ein neuer Spaltentyp

Eine Möglichkeit ist das Einführen eines neuen Spaltentyps, der die gebundene Ausgabe mehrerer Felder zulässt. Eine derartige Implementierung finden Sie im Rezept „... neue Spaltentypen für das `DataGrid` entwickeln?“.

6.34 ... auf Click-Ereignisse in einem `DataGrid`-Control reagieren?

Gerade bei einer Vielzahl von Button-Controls wie `Button`, `LinkButton` und `ImageButton` innerhalb eines `DataGrid`-Control ist eine individuelle Behandlung der auslösenden Ereignisse mitunter müßig. Eleganter ist da schon die zentralisierte Behandlung über das `DataGrid` selbst. Dank Event Bubbling ist das kein Problem.

Der Name beschreibt die Tatsache, dass Ereignisse von den einzelnen Controls an ihren Naming Container – das `DataGrid`-Control – weitergereicht werden. Über das `ItemCommand`-Ereignis ist nun eine individuelle, aber zentrale Behandlung

möglich. Um die hierzu notwendige Differenzierung des auslösenden Controls zu ermöglichen, kann jedem Button-Objekt über die Eigenschaft `CommandName` der Name der auszuführenden Aktion zugewiesen werden.

Listing 6.66 ItemCommand1.aspx

```
void dg_ItemCommand(object sender, DataGridCommandEventArgs e)
{
    string cmdName = e.CommandName;
    Response.Write("Gewählte Aktion: " + cmdName + "<br>");

    int ID = (int) dg.DataKeys[e.Item.ItemIndex];
    Response.Write("Datensatz-ID: " + ID.ToString() + "<br>");

    /*
    switch(cmdName)
    {
        case "delete" :
            // ...
        }
    */

    DataBind();
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        DataKeyField="ID"
        OnItemCommand="dg_ItemCommand">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Vorname"
                DataField="Firstname"/>

            <ASP:BoundColumn
                HeaderText="Nachname"
                DataField="Lastname"/>

            <ASP:TemplateColumn
                HeaderText="Buttons">
```

```

<ItemTemplate>

    <ASP:Button runat="server" Text="Bearbeiten" CommandName="edit"/>

    <ASP:LinkButton runat="server" Text="Details anzeigen"
        CommandName="showdetails"/>

    <ASP:ImageButton runat="server" ImageUrl="delete.gif"
        CommandName="delete"/>

</ItemTemplate>

</ASP:TemplateColumn>

</Columns>

</ASP:DataGrid>

</form>

```

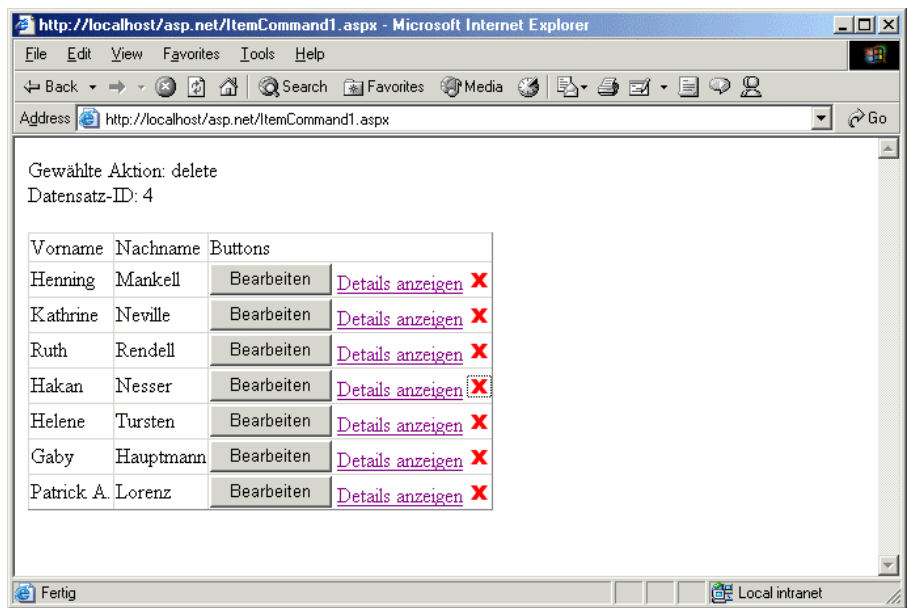


Abbildung 6.52 Es werden die Aktion und die ID des Datensatzes ausgegeben.

Im Beispiel ist die Verwendung des ItemCommand-Ereignisses in Verbindung mit drei unterschiedlichen Button(-typen) zu sehen. Alle drei lösen die gleiche Ereignisbehandlung aus und unterscheiden sich nur im übergebenen `CommandName`. Dieser kann beispielsweise über eine im Listing angedeutete switch-Abfrage aufgeschlüsselt werden.

Die Abfrage der zum Ereignis gehörenden Zeile geschieht über die `DataKeys-Collection`. Es wird die Datensatz-ID der gewählten Zeile geliefert. Informationen hierzu finden Sie auch im Rezept „... die Datensatz-ID unsichtbar in einem DataGrid-Control mitführen?“.

6.35 ... die Selektion eines DataGrid-Controls explizit aufheben?

Das DataGrid-Control bietet eine komfortable Möglichkeit, eine Zeile zu selektieren. Hierzu muss lediglich ein enthaltenes Control wie ein Button ein select-Command auslösen. Doch wie kann die Selektion wieder zurückgesetzt werden? Ganz einfach, setzen Sie die Eigenschaft einfach wie im Beispiel gezeigt auf `-1`.

Listing 6.67 DataGridSelection1.aspx

```
<script runat="server">

...

void resetselection(object sender, EventArgs e)
{
    dg.SelectedIndex = -1;
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        DataKeyField="ID">

        <SelectedItemStyle
            BackColor="Red"
            ForeColor="White"/>

        <Columns>

            <ASP:ButtonColumn
                CommandName="select"
```

```

    HeaderText="Vorname"
    DataTextField="Firstname"/>

    <ASP:ButtonColumn
        CommandName="select"
        HeaderText="Nachname"
        DataTextField="Lastname"/>

</Columns>

</ASP:DataGrid>

<p> <ASP:Button runat="server" text="Selektion aufheben"
    onclick="resetselection"/> </p>

</form>

```

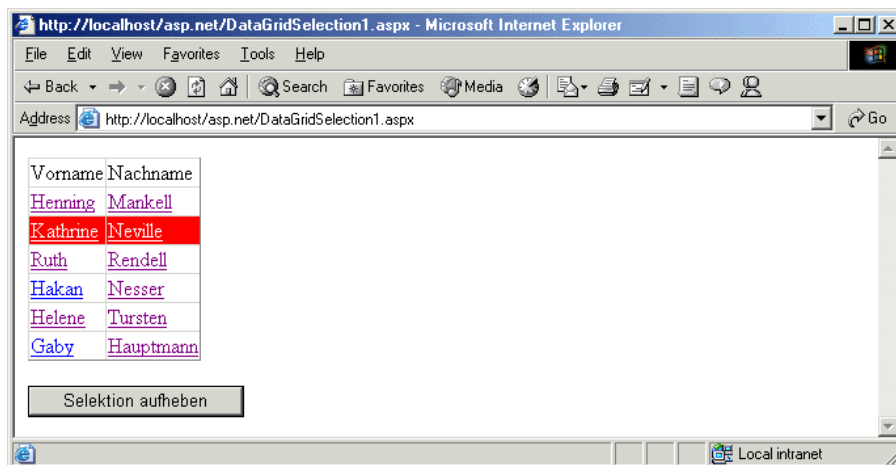


Abbildung 6.53 Die Selektion lässt sich ganz einfach aufheben.

Analog können Sie übrigens auch vorgehen, wenn Sie den Bearbeitungsmodus des DataGrid-Controls verlassen wollen. Hier erfolgt die Zuweisung von `-1` in der Regel in den beiden Ereignissen `CancelCommand` und `UpdateCommand`.

Listing 6.68 DataGridSelection2.aspx

```

<script runat="server">

...

void dg_Edit(object sender, DataGridCommandEventArgs e)
{

```

```

    dg.EditItemIndex = e.Item.ItemIndex;
    DataBind();
}

void dg_Cancel(object sender, DataGridCommandEventArgs e)
{
    dg.EditItemIndex = -1;
    DataBind();
}

void dg_Update(object sender, DataGridCommandEventArgs e)
{
    // ...
    dg.EditItemIndex = -1;
    DataBind();
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        DataKeyField="ID"
        OnEditCommand="dg_Edit"
        OnCancelCommand="dg_Cancel"
        OnUpdateCommand="dg_Update">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Vorname"
                DataField="Firstname"/>

            <ASP:BoundColumn
                HeaderText="Nachname"
                DataField="Lastname"/>

            <asp:EditCommandColumn
                EditText="Bearbeiten"
                CancelText="Abbrechen"
                UpdateText="Übernehmen"/>

        </Columns>

    </ASP:DataGrid>

</form>

```

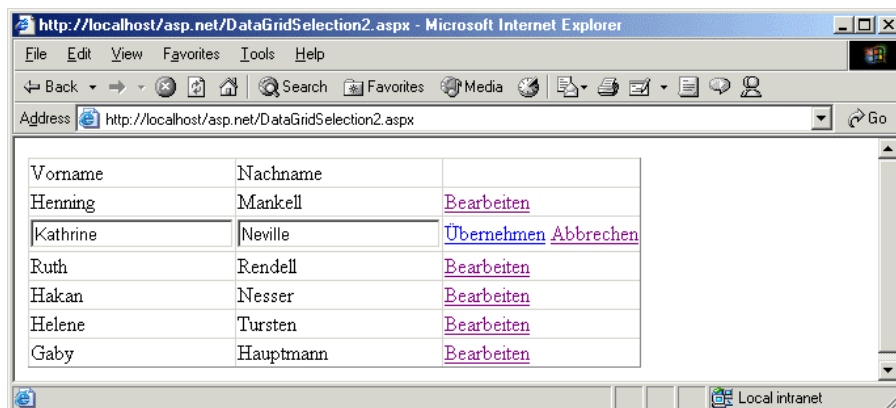


Abbildung 6.54 Der Bearbeitungsmodus kann ebenfalls sehr einfach verlassen werden.

6.36 ... ein editierbares DataGrid-Control erstellen?

Ein editierbares DataGrid-Control zu erstellen ist wirklich ziemlich einfach, denn das Control übernimmt einen Großteil der Arbeit für Sie. So werden beispielsweise automatisch Buttons zum Bearbeiten, Übernehmen und Abbrechen angezeigt, sobald Sie eine neue Spalte vom Typ `EditCommandColumn` hinzufügen. Über drei Eigenschaften können Sie den anzuzeigenden Text individuell lokalisieren.

Sobald der Benutzer einen der drei Buttons anklickt, wird eines der Ereignisse `EditCommand`, `CancelCommand` beziehungsweise `UpdateCommand` ausgelöst. Die ersten beiden übernehmen die (De-)Selektion des gewünschten Eintrags, indem die `EditItemIndex` auf den Index der zu editierenden Zeile oder aber `-1` gesetzt wird. Auf diese Weise werden automatisch TextBox-Felder für die datengebundenen Spalten angezeigt.

Die Behandlung von `UpdateCommand` ermöglicht eine Aktualisierung der unterliegenden Datenbank. Die ID des Datensatzes wird über die `DataKeys`-Collection ermittelt. Alles Weitere ist eine Frage von SQL. Im Anschluss an die Behandlung jedes der drei Ereignisse muss die Datenbindung erneut ausgeführt werden, damit die Ansicht aktualisiert werden kann.

Das Beispiel zeigt das Editieren der Autorentabelle.

Listing 6.69 DataGridEdit1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        ExecuteDataBinding();
}

void ExecuteDataBinding()
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    dg.DataSource = dataset;
    DataBind();
}

void dg_Edit(object sender, DataGridCommandEventArgs e)
{
    dg.EditItemIndex = e.Item.ItemIndex;
    ExecuteDataBinding();
}

void dg_Cancel(object sender, DataGridCommandEventArgs e)
{
    dg.EditItemIndex = -1;
    ExecuteDataBinding();
}

void dg_Update(object sender, DataGridCommandEventArgs e)
{
    TextBox tb_fn = (TextBox) e.Item.Cells[0].Controls[0];
    TextBox tb_ln = (TextBox) e.Item.Cells[1].Controls[0];

    int ID = (int) dg.DataKeys[e.Item.ItemIndex];

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
```

```
@ "Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
conn.Open();

string SQL = "UPDATE Authors SET Firstname=@Firstname,
    Lastname=@Lastname WHERE ID=@ID;";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
cmd.Parameters.Add("@Firstname", tb_fn.Text);
cmd.Parameters.Add("@Lastname", tb_ln.Text);
cmd.Parameters.Add("@ID", ID);
cmd.ExecuteNonQuery();

conn.Close();

dg.EditItemIndex = -1;
ExecuteDataBinding();
}

</script>

<form runat="server">

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID"
    OnEditCommand="dg_Edit"
    OnCancelCommand="dg_Cancel"
    OnUpdateCommand="dg_Update">

    <Columns>

    <ASP:BoundColumn
        HeaderText="Vorname"
        DataField="Firstname"/>

    <ASP:BoundColumn
        HeaderText="Nachname"
        DataField="Lastname"/>

    <asp:EditCommandColumn
        EditText="Bearbeiten"
        CancelText="Abbrechen"
        UpdateText="Übernehmen"/>

    </Columns>

</ASP:DataGrid>

</form>
```

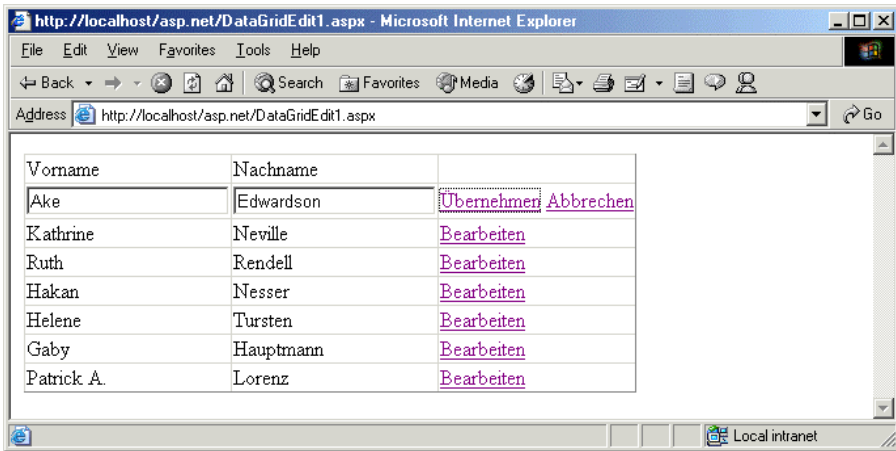



Abbildung 6.55 Auch schwedische Krimiautoren sind austauschbar.

Weitere Informationen zum Thema finden Sie auch in den folgenden Rezepten:

- „... die Datensatz-ID unsichtbar in einem DataGrid-Control mitführen?“
- „... auf die TextBox-Elemente eines DataGrid-Controls zugreifen?“

6.37 ... auf die TextBox-Elemente eines DataGrid-Controls zugreifen?

Ein editierbares DataGrid-Control zu erstellen ist sehr einfach. Sie müssen lediglich innerhalb einer Spalte vom Typ `EditCommandColumn` die Eigenschaft `EditItemIndex` auf den Index der gewünschten Zeile setzen. Alle datengebundenen Spalten dieser Zeile werden nun automatisch in Form einer `TextBox` visualisiert und laden den Benutzer zur Änderung ein. Doch wie kann im Zuge der Aktualisierung auf den geänderten Inhalt dieser `TextBox` zugegriffen werden? Hierzu müssen Sie lediglich das erste Objekt der entsprechenden Tabellenzelle abfragen. Innerhalb einer Ereignisbehandlung sieht das beispielsweise so aus:

```
TextBox tb = (TextBox) e.Item.Cells[0].Controls[0];
```

Das Beispiel zeigt diese Zeile im Kontext eines editierbaren DataGrid-Control. Die Übernahme und Speicherung der geänderten Zeile in der Datenbank wurde in diesem Fall vernachlässigt. Wie dies geht, lesen Sie im Rezept „... ein editierbares DataGrid-Control erstellen?“.

Listing 6.70 DataGridEdit1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    dg.DataSource = dataset;
    if(!IsPostBack) DataBind();
}

void dg_Edit(object sender, DataGridCommandEventArgs e)
{
    dg.EditItemIndex = e.Item.ItemIndex;
    DataBind();
}

void dg_Cancel(object sender, DataGridCommandEventArgs e)
{
    dg.EditItemIndex = -1;
    DataBind();
}

void dg_Update(object sender, DataGridCommandEventArgs e)
{
    TextBox tb_fn = (TextBox) e.Item.Cells[0].Controls[0];
    TextBox tb_ln = (TextBox) e.Item.Cells[1].Controls[0];
    Response.Write(string.Format("<p>Ihre Eingabe: {0} {1}</p>",
    tb_fn.Text, tb_ln.Text));

    // ...
    dg.EditItemIndex = -1;
    DataBind();
}

</script>

<form runat="server">
```

```

<ASP:DataGrid id="dg" runat="server"
  AutoGenerateColumns="false"
  DataKeyField="ID"
  OnEditCommand="dg_Edit"
  OnCancelCommand="dg_Cancel"
  OnUpdateCommand="dg_Update">

  <Columns>

    <ASP:BoundColumn
      HeaderText="Vorname"
      DataField="Firstname"/>

    <ASP:BoundColumn
      HeaderText="Nachname"
      DataField="Lastname"/>

    <asp:EditCommandColumn
      EditText="Bearbeiten"
      CancelText="Abbrechen"
      UpdateText="Übernehmen"/>

  </Columns>

</ASP:DataGrid>

</form>

```

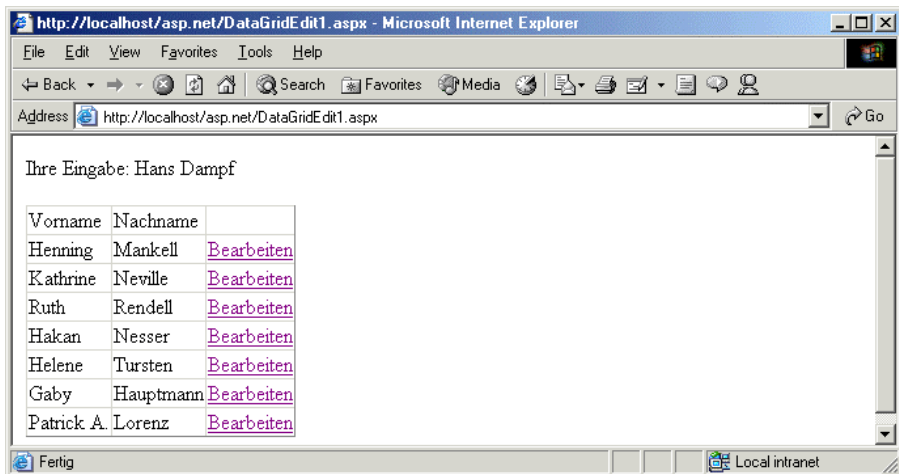


Abbildung 6.56 Die Eingabe des Benutzers wird über die TextBox abgefragt.

Die Abbildung zeigt das Ergebnis nach Änderung der ersten Zeile. Die Benutzereingabe wird im Browser ausgegeben, mangels Aktualisierung der Datenbank jedoch nicht übernommen.

Das beschriebene Vorgehen funktioniert in Verbindung mit Spalten vom Typ `BoundColumn` problemlos. Bei Verwendung von vorlagenbasierten Spalten vom Typ `TemplateColumn` ist jedoch unter Umständen eine Anpassung notwendig. Hier kann es passieren, dass vor dem `TextBox`-Control noch andere Objekte platziert wurden, so dass eine Abfrage über den Index der `Controls`-Collection recht unsicher wäre. In diesem Fall sollten Sie dem Eingabeelement eine ID zuweisen und das Objekt anhand dieser über die `FindControl`-Methode ausfindig machen. Das zweite Listing zeigt diesen Ansatz.

Listing 6.71 DataGridEdit2.aspx

```
...

void dg_Update(object sender, DataGridCommandEventArgs e)
{
    TextBox tb_fn = (TextBox) e.Item.Cells[0].FindControl("tb_fn");
    TextBox tb_ln = (TextBox) e.Item.Cells[0].FindControl("tb_ln");

    Response.Write(string.Format("<p>Ihre Eingabe: {0} {1}</p>",
        tb_fn.Text, tb_ln.Text));

    // ...
    dg.EditItemIndex = -1;
    DataBind();
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        DataKeyField="ID"
        OnEditCommand="dg_Edit"
        OnCancelCommand="dg_Cancel"
        OnUpdateCommand="dg_Update">

        <Columns>

            <ASP:TemplateColumn
                HeaderText="Name">

                <ItemTemplate>
                    <%%# DataBinder.Eval(Container.DataItem, "Firstname") %>
```

```

    <#% DataBinder.Eval(Container.DataItem, "Lastname") %>
  </ItemTemplate>

  <EditItemTemplate>
    Ihre neue Eingabe:
    <ASP:TextBox id="tb_fn" runat="server"
      text=<#% DataBinder.Eval(Container.DataItem, "Firstname") %>/>
    <ASP:TextBox id="tb_ln" runat="server"
      text=<#% DataBinder.Eval(Container.DataItem, "Lastname") %>/>
  </EditItemTemplate>

</ASP:TemplateColumn>

<asp:EditCommandColumn
  EditText="Bearbeiten"
  CancelText="Abbrechen"
  UpdateText="Übernehmen" />

</Columns>

</ASP:DataGrid>

</form>

```

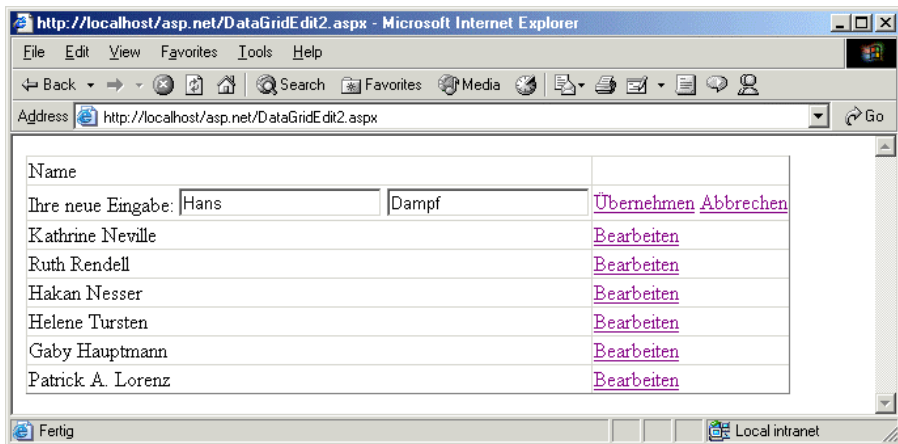


Abbildung 6.57 Bei vorlagenbasierten Spalten kann die Methode FindControl genutzt werden.

6.38 ... das Eingabefeld eines editierbaren DataGrid-Controls verändern?

Das zum Editieren einer Spalte automatisch angezeigte TextBox-Control sieht in aller Regel ziemlich langweilig aus. Wer die üblichen Formatierungsmöglichkeiten wie Schrift- und Rahmenstil einsetzen möchte, muss eine vorlagenbasierte Spalte vom `TemplateColumn` verwenden. Wirklich? Nein, mit einem Trick geht es auch einfacher. Sie müssen das Eingabefeld dazu lediglich im `ItemCreated`-Ereignis „abfangen“ und mit den gewünschten Eigenschaften versehen. Da das Ereignis für jede Zeile ausgelöst wird, müssen Sie zuvor über die `ItemType`-Eigenschaft der übergebenen `DataGridItem`-Instanz den aktuellen Spaltentyp ermitteln.

Listing 6.72 DataGridEdit2.aspx

```
...  
  
void dg_ItemCreated(object sender, DataGridItemEventArgs e)  
{  
    if(e.Item.ItemType == ListItemType.EditItem)  
    {  
        TextBox tb_fn = (TextBox) e.Item.Cells[0].Controls[0];  
        tb_fn.Font.Name = "Verdana";  
        tb_fn.Font.Size = 15;  
        tb_fn.Font.Bold = true;  
  
        TextBox tb_ln = (TextBox) e.Item.Cells[1].Controls[0];  
        tb_ln.Font.Name = "Comic Sans MS";  
        tb_ln.Font.Bold = true;  
        tb_ln.BorderStyle = BorderStyle.Dotted;  
        tb_ln.BorderWidth = 3;  
    }  
}  
  
...
```

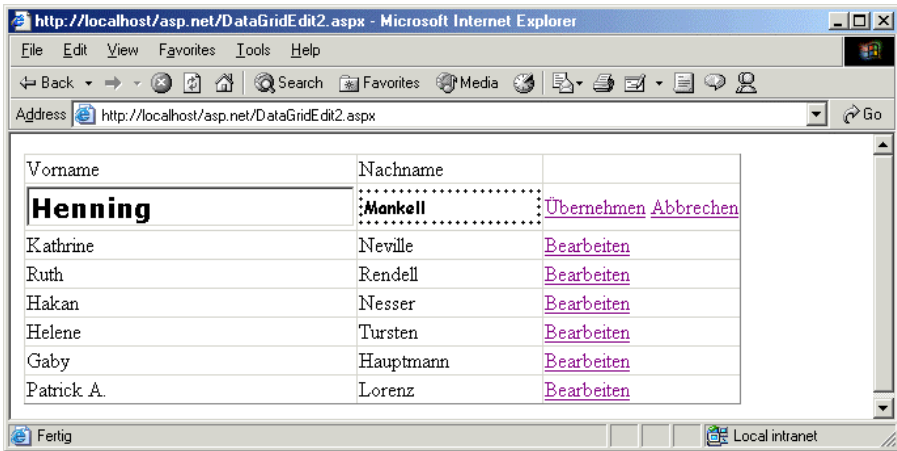


Abbildung 6.58 Über Geschmack lässt sich bekanntlich sehr gut streiten.

Die Abbildung macht deutlich, dass auch ausgefallene Designs mit einfachen Mitteln realisierbar sind. Es stehen die regulären Möglichkeiten eines TextBox-Controls in vollem Umfang zur Verfügung. Das bedeutet auch, dass Sie auf diese Weise ein mehrzeiliges Eingabefeld erzeugen können:

Listing 6.73 DataGridEdit3.aspx

```
...

void dg_ItemCreated(object sender, DataGridItemEventArgs e)
{
    if(e.Item.ItemType == ListItemType.EditItem)
    {
        TextBox tb_fn = (TextBox) e.Item.Cells[0].Controls[0];
        tb_fn.TextMode = TextBoxMode.MultiLine;

        TextBox tb_ln = (TextBox) e.Item.Cells[1].Controls[0];
        tb_ln.TextMode = TextBoxMode.MultiLine;
    }
}

...
```

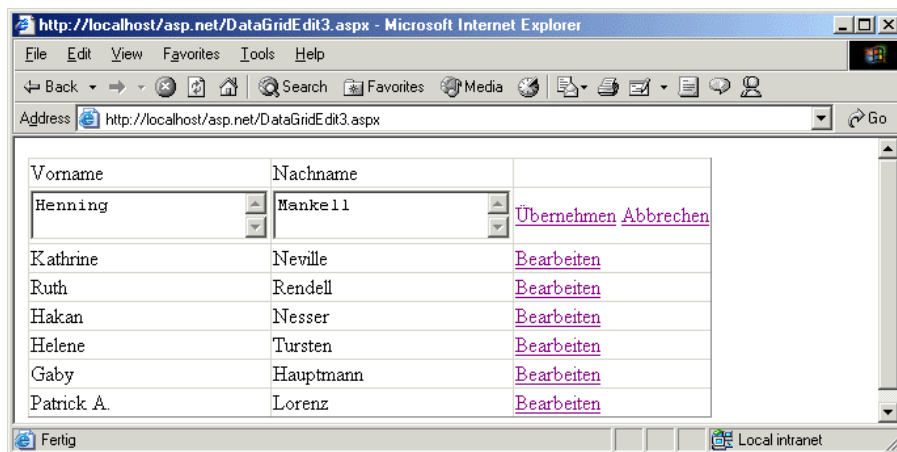


Abbildung 6.59 Ein mehrzeiliges Eingabefeld im DataGrid-Control

6.39 ... die Eingaben eines editierbaren DataGrid-Controls validieren?

Validation Controls sind eine tolle Sache, die auch beim Editieren von Spalten eines DataGrid-Controls sinnvoll einsetzbar sind. Allerdings gibt es keine Möglichkeit, diese so einfach zuzuweisen. Wieder einmal bedarf es eines kleinen Tricks, und wieder einmal bildet die Basis dafür das ItemCreated-Ereignis. Hier wird die TextBox abgefangen und um die benötigten Validierungen ergänzt.

Das Beispiel zeigt die Autorentabelle. Sobald eine Zeile geändert werden soll, werden im genannten Ereignis die Eingabefelder abgefragt. Zur Identifizierung wird eine eindeutige ID vergeben. Zudem wird die maximale Länge der Eingabe gesetzt, damit diese später nicht den zur Verfügung stehenden Platz in der Datenbank überschreitet. Anschließend wird das gewünschte Validation Control instanziiert, über die ID angebunden und programmatisch mit den benötigten Parametern versehen. Natürlich sind auch mehrere Validierungen für ein einzelnes Eingabefeld möglich.

Listing 6.74 DataGridEdit4.aspx

```
...  
  
void dg_ItemCreated(object sender, DataGridItemEventArgs e)  
{  
    if(e.Item.ItemType == ListItemType.EditItem)  
    {  
        TextBox tb;  
        RequiredFieldValidator validator;  

```



```

// Vorname
tb = (TextBox) e.Item.Cells[0].Controls[0];
tb.ID = "tb_fn";
tb.MaxLength = 20;

validator = new RequiredFieldValidator();
validator.ControlToValidate = "tb_fn";
validator.Display = ValidatorDisplay.Dynamic;
validator.Text = "<img src='exclamation.gif' />";
validator.ErrorMessage = "Bitte geben Sie einen Vornamen an!";
e.Item.Cells[0].Controls.Add(validator);

// Nachname
tb = (TextBox) e.Item.Cells[1].Controls[0];
tb.ID = "tb_ln";
tb.MaxLength = 20;

validator = new RequiredFieldValidator();
validator.ControlToValidate = "tb_ln";
validator.Display = ValidatorDisplay.Dynamic;
validator.Text = "<img src='exclamation.gif' />";
validator.ErrorMessage = "Bitte geben Sie einen Nachnamen an!";
e.Item.Cells[1].Controls.Add(validator);
}
}
...

```

Im Listing ist zu erkennen, dass leere Eingaben nicht angenommen werden. Eine Beschreibung der Fehler wird über ein `ValidationSummary`-Control ausgegeben. Die Behandlung des `UpdateCommand`-Ereignisses wird so lange aufgehalten, bis die Eingaben dem festgelegten Muster entsprechen. Es ist allerdings möglich, den Bearbeitungsmodus über den Abbrechen-Button zu verlassen. Die Entwickler haben sinnvollerweise daran gedacht, die Eigenschaft `CausesValidation` dieses Buttons auf `false` zu setzen.

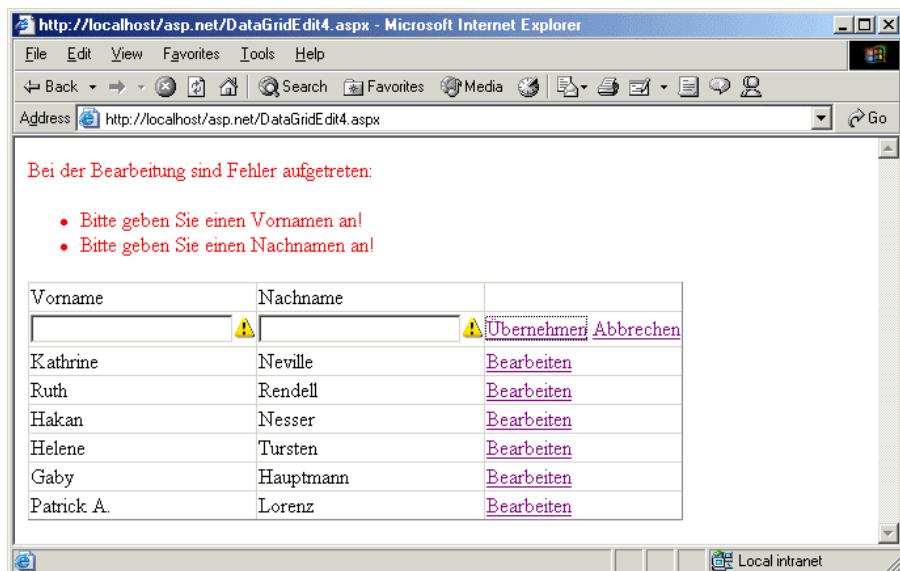


Abbildung 6.60 Nur korrekte Eingaben werden akzeptiert.

6.40 ... einen booleschen Wert in einem DataGrid-Control editieren?

Das Editieren von Zeichenfeldern, numerischen Werten und Daten mag über ein einfaches Eingabefeld wie das TextBox-Control sinnvoll möglich sein. Für das Editieren eines booleschen Werts gilt dies allerdings nicht. Hier gibt es ein Spezialobjekt, die *CheckBox*. In diesem Rezept möchte ich Ihnen einige Möglichkeiten vorstellen, ein solches Control beim Editieren einer DataGrid-Zeile zu nutzen.

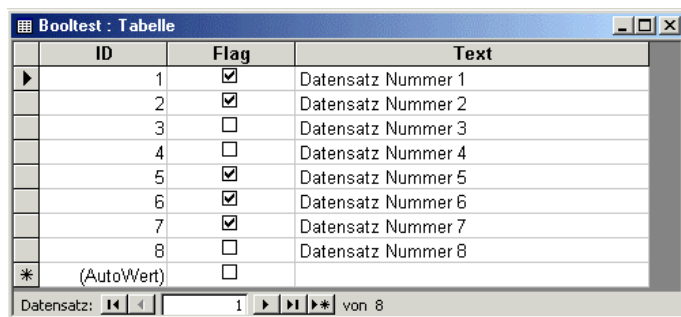


Abbildung 6.61 Die Beispieltabelle „Booltest“

Alle Beispiele gehen von der Tabelle `Booltest` aus der Beispieldatenbank `misc.mdb` aus. Diese enthält ein boolesches und ein Zeichenfeld.

Editieren in selektierter Zeile

Die einfachste Variante zur Darstellung eines editierbaren, booleschen Flags stellt die Verwendung einer vorlagenbasierten Spalte vom Typ `TemplateColumn` dar. Im Normalzustand werden zwei Image-Controls zur Darstellung eines Symbols verwendet. Die Controls werden gegensätzlich über den Wert des Feldes ein- beziehungsweise ausgeblendet. Soll die Zeile editiert werden, kommt die als `EditItemTemplate` hinterlegte zweite Vorlage zum Tragen. Hier ist das benötigte `CheckBox`-Control hinterlegt, das bei der Aktualisierung der Datenquelle analog zu den anderen Eingabefeldern ausgewertet wird. Das Beispiel zeigt's.

Listing 6.75 CheckBox1.aspx

```
...

void dg_Update(object sender, DataGridCommandEventArgs e)
{
    CheckBox cb = (CheckBox) e.Item.Cells[0].FindControl("cb");
    TextBox tb = (TextBox) e.Item.Cells[1].Controls[0];

    int ID = (int) dg.DataKeys[e.Item.ItemIndex];

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb");
    conn.Open();

    string SQL = "UPDATE Booltest SET Flag=@Flag, [Text]=@Text
        WHERE ID=@ID;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Flag", cb.Checked);
    cmd.Parameters.Add("@Text", tb.Text);
    cmd.Parameters.Add("@ID", ID);
    cmd.ExecuteNonQuery();

    conn.Close();

    dg.EditItemIndex = -1;
    ExecuteDataBinding();
}

</script>
```

```
<form runat="server">

<ASP:ValidationSummary runat="server"
  HeaderText="Bei der Bearbeitung sind Fehler aufgetreten:"/>

<ASP:DataGrid id="dg" runat="server"
  AutoGenerateColumns="false"
  DataKeyField="ID"
  OnEditCommand="dg_Edit"
  OnCancelCommand="dg_Cancel"
  OnUpdateCommand="dg_Update">

  <Columns>

    <ASP:TemplateColumn
      HeaderText="Flag">

      <ItemTemplate>
        <ASP:Image runat="server"
          ImageUrl="checked.gif"
          Visible=<## DataBinder.Eval(Container.DataItem, "Flag") %>/>

        <ASP:Image runat="server"
          ImageUrl="unchecked.gif"
          Visible=
            <## ! (bool) DataBinder.Eval(Container.DataItem, "Flag") %>/>
        </ItemTemplate>

        <EditItemTemplate>
          <ASP:CheckBox runat="server" id="cb"
            Checked=<## DataBinder.Eval(Container.DataItem, "Flag") %>/>
          </EditItemTemplate>

        </ASP:TemplateColumn>

    <ASP:BoundColumn
      HeaderText="Text"
      DataField="Text"/>

    <asp:EditCommandColumn
      EditText="Bearbeiten"
      CancelText="Abbrechen"
      UpdateText="Übernehmen"/>

  </Columns>

</ASP:DataGrid>

</form>
```

Die Abbildung zeigt den Einsatz des Listings. Zu jedem Datensatz wird der Wert des Flags über die entsprechende Grafik dargestellt. Im Bearbeitungsmodus einer Zeile wird stattdessen ein CheckBox-Control dargestellt. Beim Aktualisieren des Datensatzes wird der aktuelle Status übernommen und anschließend über die Grafik dargestellt.

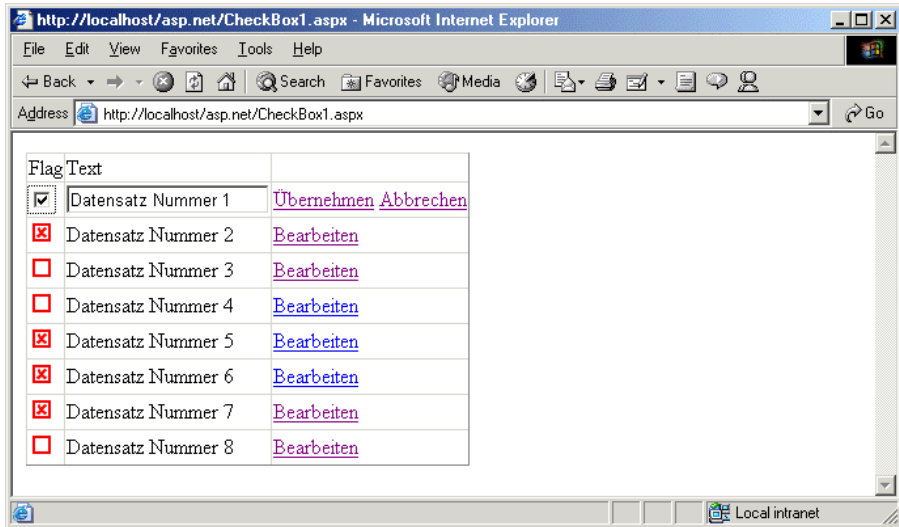


Abbildung 6.62 Das Flag lässt sich ganz einfach editieren.

Editieren in jeder Zeile

Eine alternative Möglichkeit ist das direkte Editieren des Flags in *jeder* Zeile. Es ist dann nicht mehr notwendig, in den Bearbeitungsmodus zu wechseln. Möglicherweise muss dieser auch gar nicht mehr angeboten werden. So auch im folgenden, angeänderten Beispiel. Über eine `TemplateColumn` wird für jede Zeile eine `CheckBox` angezeigt, die mit dem aktuellen Wert „gefüttert“ wurde.

Ein Klick löst dank eingeschaltetem `AutoPostBack` direkt das zentrale hinterlegte Ereignis aus. An dieser Stelle besteht der Kniff in der Abfrage des sender-Parameters. Darüber lässt sich die auslösende `CheckBox` ermitteln. Über deren Eigenschaft `NamingContainer` kann wiederum die zugehörige `DataGridItem`-Instanz und somit indirekt die ID des korrespondierenden Datensatzes abgefragt werden. Diese wird schließlich verwendet, um die Datenquelle mit dem neuen Wert zu aktualisieren.

Listing 6.76 CheckBox2.aspx

```
...

void cb_CheckedChanged(object sender, EventArgs e)
{
    CheckBox cb = (CheckBox) sender;
    DataGridItem dgi = (DataGridItem) cb.NamingContainer;

    int ID = (int) dg.DataKeys[dgi.ItemIndex];

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb");
    conn.Open();

    string SQL = "UPDATE Booltest SET Flag=@Flag WHERE ID=@ID;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Flag", cb.Checked);
    cmd.Parameters.Add("@ID", ID);
    cmd.ExecuteNonQuery();

    conn.Close();

    ExecuteDataBinding();
}

</script>

<form runat="server">

    <ASP:ValidationSummary runat="server"
        HeaderText="Bei der Bearbeitung sind Fehler aufgetreten:"/>

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        DataKeyField="ID">

        <Columns>

            <ASP:TemplateColumn
                HeaderText="Flag">

                <ItemTemplate>
                    <ASP:CheckBox runat="server" id="cb"
                        Checked=<%=# DataBinder.Eval(Container.DataItem, "Flag") %>
                        OnCheckedChanged="cb_CheckedChanged"
                        AutoPostBack="true"/>
                </ItemTemplate>
            </ASP:TemplateColumn>
        </Columns>
    </ASP:DataGrid>
</form>
```

```

</ASP:TemplateColumn>

<ASP:BoundColumn
  HeaderText="Text "
  DataField="Text"/>

</Columns>

</ASP:DataGrid>

</form>

```

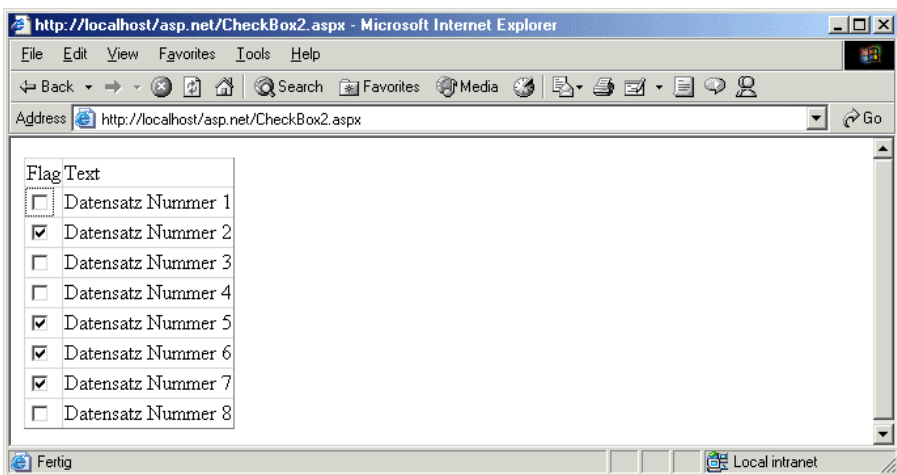


Abbildung 6.63 Das Editieren ist in jeder Zeile möglich.

Was in der Abbildung nicht zu sehen ist, ist die Leichtigkeit, mit der das Flag editiert werden kann. Ein einfacher Klick löst automatisch denPostBack und somit die Aktualisierung der Datenbank aus. Zu sehen ist von diesem Prozess allerdings kaum etwas ...

ManuellerPostBack

Anders sieht es vielleicht aus, wenn die Seite etwas umfangreicher oder die Anbindung etwas schlechter ist. Für solche Fälle bietet es sich an, dem automatischenPostBack eine manuelle Variante vorzuziehen. So kann der Benutzer in Ruhe die gewünschten Einträge vornehmen und die Änderungen bei Bedarf per Buttonklick mit dem Server abgleichen. Der notwendige Ansatz unterscheidet sich deutlich von den bisherigen, denn nun müssen alle Einträge auf einen Schlag und nicht wie bisher einzeln abgefragt werden.

Die Ereignisbehandlung des Buttons durchläuft alle Elemente des DataGrid-Control, also deren Zeilen. Jede einzelne wird über eine Instanz der Klasse `DataGridItem` repräsentiert. Darüber lässt sich wie bereits oben gezeigt das jeweilige `CheckBox`-Control sowie die Datensatz-ID ermitteln.

Um nicht für jedes Element eine separate Abfrage durchführen zu müssen, verwende ich zwei `ArrayList`-Instanzen für markierte und nicht markierte Elemente, denen ich je nach Wert der `Checked`-Eigenschaft die ID des Datensatzes zuweise. Im Anschluss an die Schleife werden die Collections in Arrays und schließlich in eine mit Kommata separierte Zeichenkette umgewandelt. Diese bildet die Basis für die notwendige SQL-Query zur Aktualisierung der Daten.

Listing 6.77 CheckBox3.aspx

```
...

void bt_click(object sender, EventArgs e)
{
    ArrayList al_checked = new ArrayList();
    ArrayList al_unchecked = new ArrayList();

    for(int i=0; i<dg.Items.Count; i++)
    {
        DataGridItem dgi = dg.Items[i];
        CheckBox cb = (CheckBox) dgi.FindControl("cb");

        string ID = dg.DataKeys[dgi.ItemIndex].ToString();

        if(cb.Checked)
            al_checked.Add(ID);
        else
            al_unchecked.Add(ID);
    }

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb");
    conn.Open();

    string[] SQLArray;
    string SQL;
    OleDbCommand cmd;

    // CheckBox checked
    if(al_checked.Count > 0 )
    {
        SQLArray = new string[al_checked.Count];
        al_checked.CopyTo(SQLArray);
```



```

SQL = string.Format("UPDATE Booltest SET Flag=TRUE WHERE ID IN
({0})", string.Join(" ", SQLArray));
cmd = new OleDbCommand(SQL, conn);
cmd.ExecuteNonQuery();
}

// CheckBox unchecked
if(al_unchecked.Count > 0 )
{
    SQLArray = new string[al_unchecked.Count];
    al_unchecked.CopyTo(SQLArray);

    SQL = string.Format("UPDATE Booltest SET Flag=FALSE WHERE ID IN
({0})", string.Join(" ", SQLArray));
    cmd = new OleDbCommand(SQL, conn);
    cmd.ExecuteNonQuery();
}

conn.Close();

ExecuteDataBinding();
}

</script>

<form runat="server">

<ASP:ValidationSummary runat="server"
    HeaderText="Bei der Bearbeitung sind Fehler aufgetreten:"/>

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID">

<Columns>

<ASP:TemplateColumn
    HeaderText="Flag">

    <ItemTemplate>
        <ASP:CheckBox runat="server" id="cb"
            Checked=<%=# DataBinder.Eval(Container.DataItem, "Flag") %>/>
        </ItemTemplate>

    </ASP:TemplateColumn>

<ASP:BoundColumn
    HeaderText="Text "

```

```
DataField="Text"/>

</Columns>

</ASP:DataGrid>

<p>
  <ASP:Button runat="server" Text="Aktualisieren" OnClick="bt_click"/>
</p>

</form>
```

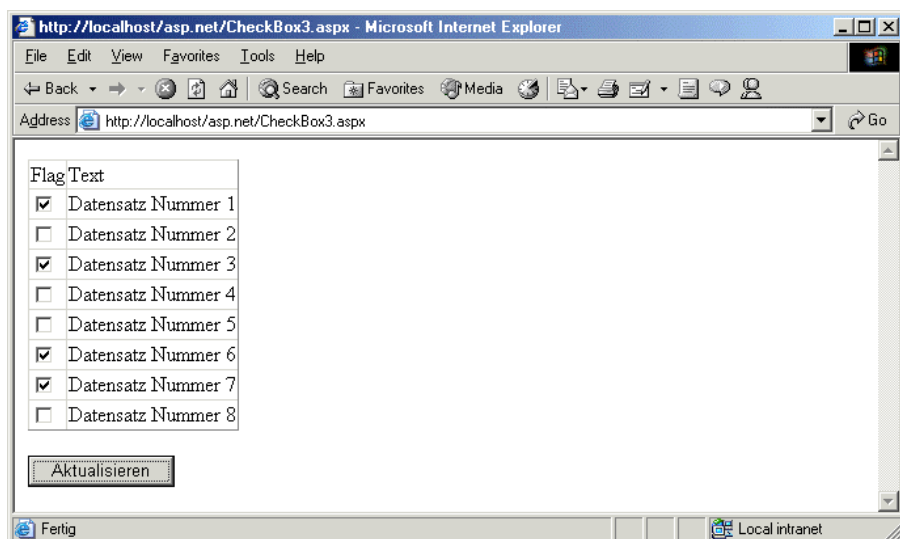


Abbildung 6.64 Die Änderungen werden auf Buttonklick gesammelt übernommen.

Selektion aller Einträge

Zu guter Letzt möchte ich Ihnen noch eine Möglichkeit vorstellen, wie Sie alle Elemente auf einmal selektieren können – ein nicht selten gewünschtes Feature. Realisiert wird dies durch ein einfaches JavaScript, das beim Klick auf eine zusätzliche `CheckBox` deren Status auf alle anderen überträgt. Der initiale Status der `CheckBox` ist deaktiviert. Nur wenn alle Einträge der Datenbank aktiviert sind, wird auch diese globale `CheckBox` aktiviert. Hierzu wird mittels `ExecuteScalar` eine Query gegen die Datenbank gefahren.

Listing 6.78 CheckBox4.aspx

```

...

void ExecuteDataBinding()
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Booltest";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    OleDbDataReader reader = cmd.ExecuteReader();
    dg.DataSource = reader;
    DataBind();
    reader.Close();

    SQL = "SELECT TOP 1 ID FROM Booltest WHERE Flag=FALSE;";
    cmd = new OleDbCommand(SQL, conn);
    cb_all.Checked = (cmd.ExecuteScalar() == null);

    conn.Close();
}

...

<script language="javascript">

    function selectAll(value)
    {
        var form = document.forms[0];
        for(i=0; i<form.length; i++)
        {
            if(form.elements[i].type == "checkbox")
                form.elements[i].checked = value;
        }
    }

</script>

<p>
    <ASP:CheckBox id="cb_all" runat="server" Text="Alle selektieren"
        onClick="selectAll(cb_all.checked);"/>
</p>

<p>
    <ASP:Button runat="server" Text="Aktualisieren" OnClick="bt_click"/>
</p>

```

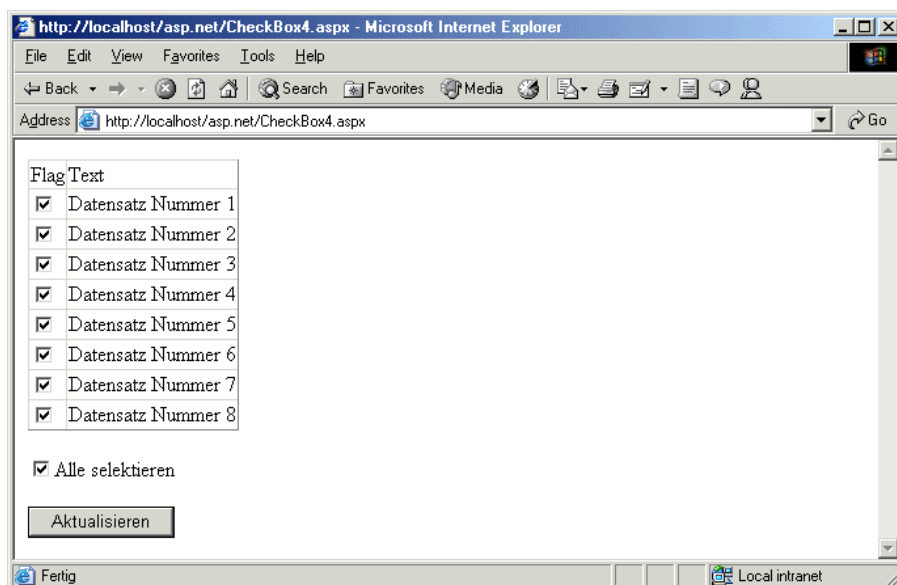


Abbildung 6.65 Ein Klick genügt, um alle Datensätze zu markieren.

Die Abbildung zeigt die Selektion aller Einträge durch einen einfachen Klick auf die zentrale `CheckBox`. Da die Selektion ausschließlich clientseitig erfolgt, muss anschließend noch der Button zum Abgleich mit der Datenbank angeklickt werden. Dies ist gewollt. Wer dies jedoch nicht möchte, kann wie im weiter gezeigten Beispiel `AutoPostBack` zusammen mit einer serverseitigen Behandlung des `CheckBox`-Ereignisses `CheckedChanged` einsetzen.

6.41 ... dem `DataGrid`-Control dynamisch Spalten anfügen?

In aller Regel werden die Spalten eines `DataGrid`-Controls entweder selbstständig durch dieses erzeugt, oder aber diese werden im Layout-Bereich der Seite explizit hinterlegt. Die dritte Möglichkeit wird eher selten verwendet, obwohl sie eine für die Praxis unschlagbare Flexibilität bietet. Die Rede ist von dem programmatischen Zugriff auf die `Columns`-Collection. Über diese können Sie bestehende Spalten modifizieren und löschen sowie neue anfügen – ganz so, wie Sie es von einer `Collection` gewohnt sind.

Bei der Anlage neuer Spalten stehen Ihnen die insgesamt fünf Spaltentypen uneingeschränkt zur Verfügung. Es handelt sich um `BoundColumn`, `ButtonColumn`, `EditCommandColumn`, `HyperLinkColumn` und `TemplateColumn`. Sie können eine neue Instanz des gewünschten Typs erstellen und diese der Methode `Add` der `Co`

lums-Collection übergeben. Die Reihenfolge der Übergabe bestimmt direkt das spätere Erscheinungsbild, da die Spalten in genau dieser Reihenfolge dargestellt werden.

Das Beispiel demonstriert den Einsatz der Collection anhand zweier datengebundener Spalten, die dem Control innerhalb von `Page_Load` angefügt werden. Die automatische Generierung der Spalten wurde deaktiviert, und auch im Layout-Bereich sind keine Vorlagen hinterlegt. Die Deklaration des DataGrid-Controls ist entsprechend kurz geraten.

Listing 6.79 Columns1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    BoundColumn col_fn = new BoundColumn();
    col_fn.HeaderText = "Vorname";
    col_fn.DataField = "Firstname";
    dg.Columns.Add(col_fn);

    BoundColumn col_ln = new BoundColumn();
    col_ln.HeaderText = "Nachname";
    col_ln.DataField = "Lastname";
    dg.Columns.Add(col_ln);

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();

    conn.Close();
}

</script>

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID"/>

</form>
```

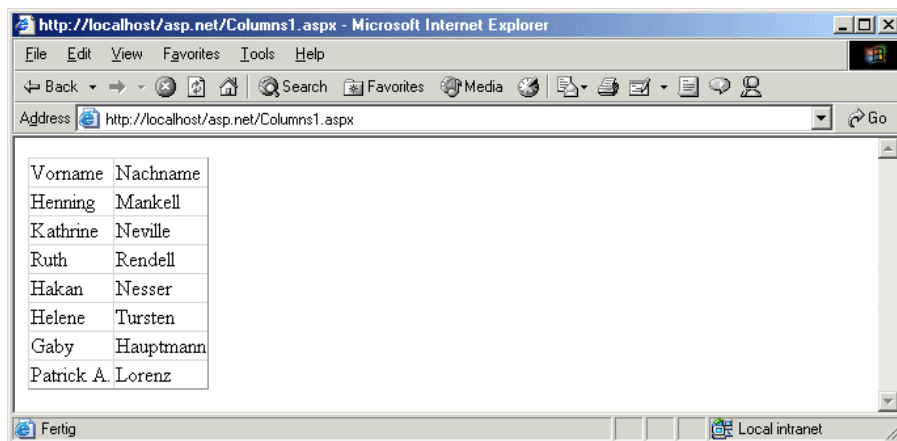


Abbildung 6.66 Die Spalten wurden programmatisch angelegt.

Sofern Sie die Reihenfolge der Spalten explizit festlegen wollen, verwenden Sie beim Hinzufügen einfach die Methode `AddAt` und übergeben als ersten Parameter den gewünschten Index. Natürlich können Sie bestehende Spalten auf analoge Weise auch löschen, indem Sie die Methoden `Remove` beziehungsweise `RemoveAt` aufrufen.

Sie können auf diese Weise beliebige Spalten anlegen, auch die vorlagenbasierte `TemplateColumn`. Doch woher die Vorlage nehmen, wenn nicht stehen? Das Rezept „... ein Template dynamisch laden?“ aus dem Kapitel „User Controls“ verrät es. Sie können die Vorlage mittels `LoadTemplate` aus einer Datei einladen.

Listing 6.80 Columns2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    TemplateColumn col = new TemplateColumn();
    col.HeaderText = "Name";
    col.ItemTemplate = this.LoadTemplate("template.ascx");
    dg.Columns.Add(col);

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();
}
```

```

    conn.Close();
}

</script>

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID" />

</form>

```

Die Vorlage ist einer zweiten Datei hinterlegt und sieht wie folgt aus:

Listing 6.81 template.ascx

```

<%# DataBinder.Eval(Container, "DataItem[Firstname] ") %>
<%# DataBinder.Eval(Container, "DataItem[Lastname] ") %>

```

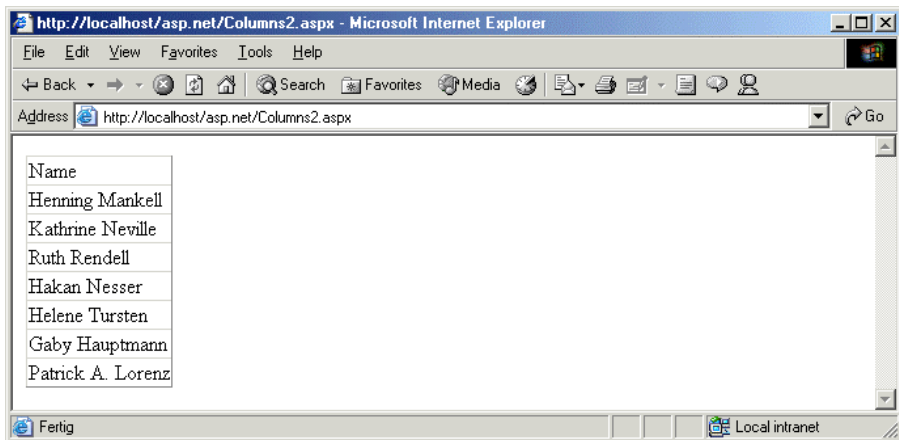


Abbildung 6.67 Die Vorlage für die Spalte wurde aus einer externen Datei eingelesen.



Theoretisch können Sie programmatisch erzeugte Spalten problemlos in Verbindung mit automatisch generierten verwenden. Beachten Sie bitte jedoch, dass die automatischen Spalten nicht in der Columns-Collection geführt werden und prinzipiell immer *nach* den programmatisch hinzugefügten Spalten dargestellt werden.

6.42 ... neue Spaltentypen für das DataGrid entwickeln?

Statt immer wieder ähnliche Implementierungen über eine vorlagenbasierte Spalte vom Typ `TemplateColumn` vorzunehmen, kann es mitunter hilfreich sein, eigene Spaltentypen zu implementieren. Ich denke da beispielsweise an die Ausgabe mehrerer Felder innerhalb einer datengebundenen Spalte. Im Rezept „... mehrere Felder in einer DataGrid-Spalte anzeigen?“ habe ich Ihnen mehrere Realisierungsmöglichkeiten vorgestellt, doch so richtig prickelnd ist keine davon. In diesem Rezept möchte ich Ihnen vorstellen, wie Sie mit wenigen Handgriffen Ihre eigenen Spaltentypen für das DataGrid-Control entwickeln können.

Alle vorhandenen Typen leiten sich von der abstrakten Basisklasse `DataGridColumn` ab. Die Klasse beinhaltet bereits eine ganze Menge Grundfunktionalität und auch Eigenschaften für Kopfzeilentext, Style und so weiter. Sie können sich bei der Implementierung eines eigenen Spaltentyps daher ganz auf die eigentliche Hauptaufgabe konzentrieren.

Ihre Arbeit beginnt mit dem Überschreiben der öffentlichen Methode `InitializeCell`. Dieser wird nacheinander jede Tabellenzelle übergeben, in die sich die Spalte zeichnen soll. Sie können an dieser Stelle die notwendigen Initialisierungen vornehmen, sollten dabei aber unbedingt auch die Basisimplementierung der Methode aufrufen.

Das folgende Beispiel zeigt eine Ableitung der Klasse `DataGridColumn`. Die neue Klasse `MultipleFieldsColumn` soll mehrere datengebundene Felder innerhalb einer Spalte darstellen. Sie enthält zwei Eigenschaften `DataFields` und `DataFormatString`, denen eine mit Kommata separierte Liste der Felder und eine Formatierungszeichenkette zugewiesen wird.

Die überschriebene Methode `InitializeCell` ruft zunächst die Basismethode auf. Hier wird beispielsweise der Kopfzeilentext ausgegeben. Anschließend wird der Zeilentyp ermittelt, der ebenfalls als Parameter übergeben wird. Handelt es sich um eine reguläre Spalte, so wird das `DataBinding`-Ereignis der zu verwendenden Tabellenzelle an die lokale Methode `cell_DataBinding` gebunden.

Die Methode wird für jede einzelne Zelle aufgerufen, sobald diese an eine Datenquelle gebunden wird. Über die `NamingContainer` kann die darüber liegende `DataGridItem`-Instanz und darüber das aktuelle Datenelement abgefragt werden. Die `Split`-Methode wandelt die Zeichenkette in ein Array. Über dieses lässt sich ein object-Array mit dem Inhalt der auszugebenden Felder erstellen. Dieses wird der Methode `string.Format` übergeben, welche für die Darstellung verantwortlich ist. Die erhaltene Zeichenkette wird über ein neu instanziiertes `Literal`-Control der Zelle hinzugefügt. Uff, das war's ...

Listing 6.82 MultipleFieldsColumn.cs

```

using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace PAL.Projects.AspNetKochbuch
{
    public class MultipleFieldsColumn : DataGridColumn
    {
        public string DataFields
        {
            get { return((string) ViewState["DataFields"]); }
            set { ViewState["DataFields"] = value; }
        }

        public string DataFormatString
        {
            get { return((string) ViewState["DataFormatString"]); }
            set { ViewState["DataFormatString"] = value; }
        }

        public override void InitializeCell(TableCell cell,
            int columnIndex, ListItemType itemType)
        {
            base.InitializeCell(cell, columnIndex, itemType);

            if(itemType == ListItemType.Item ||
                itemType == ListItemType.AlternatingItem)
            {
                cell.DataBinding += new EventHandler(cell_DataBinding);
            }
        }

        protected void cell_DataBinding(object sender, EventArgs e)
        {
            if(DataFields == null)
                throw(new ArgumentException("Property DataFields
                    can not be null"));

            if(DataFormatString == null)
                throw(new ArgumentException("Property DataFormatString
                    can not be null"));

            TableCell cell = (TableCell) sender;
            DataGridItem dgi = (DataGridItem) cell.NamingContainer;

            string[] fields = this.DataFields.Split(',');
            object[] values = new object[fields.Length];

```

```
for(int i=0; i<fields.Length; i++)
{
    values[i] = DataBinder.Eval(dgi.DataItem, fields[i]);
}

string contents = string.Format(this.DataFormatString, values);

Literal lit = new Literal();
lit.Text = contents;
cell.Controls.Add(lit);
}
}
}
```

Nachdem die Beschreibung etwas länger geraten ist, erscheint das Listing selbst relativ kurz. Aber das ist auch alles, was zur Darstellung mehrerer Felder in einer individuellen, datengebundenen Spalte notwendig ist. Das folgende Beispiel soll den Beweis bringen. Zuvor muss die Quellcode-Datei jedoch noch mittels `csc.exe` kompiliert und im `bin`-Verzeichnis der Web-Applikation abgelegt werden.

Listing 6.83 MultipleFieldsColumn1.aspx

```
<% @Import Namespace="PAL.Projects.AspNetKochbuch" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    MultipleFieldsColumn col = new MultipleFieldsColumn();
    col.HeaderText = "Name";
    col.DataFields="Firstname, Lastname";
    col.DataFormatString="{1}, {0}";
    dg.Columns.Add(col);

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();

    conn.Close();
}
```

```

</script>

<form runat="server">

  <ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID"/>

</form>

```

Das Beispiel zeigt die Anlage der benutzerdefinierten Spalte über die Columns-Collection. Über die Eigenschaft `DataFields` werden die beiden Felder zugewiesen. Der Eigenschaft `DataFormatString` werden die Formatierungsanweisungen übergeben. Es soll erst der Nachname und dann – mit einem Komma getrennt – der Vorname ausgegeben werden. Ob das klappt, verrät die Abbildung.

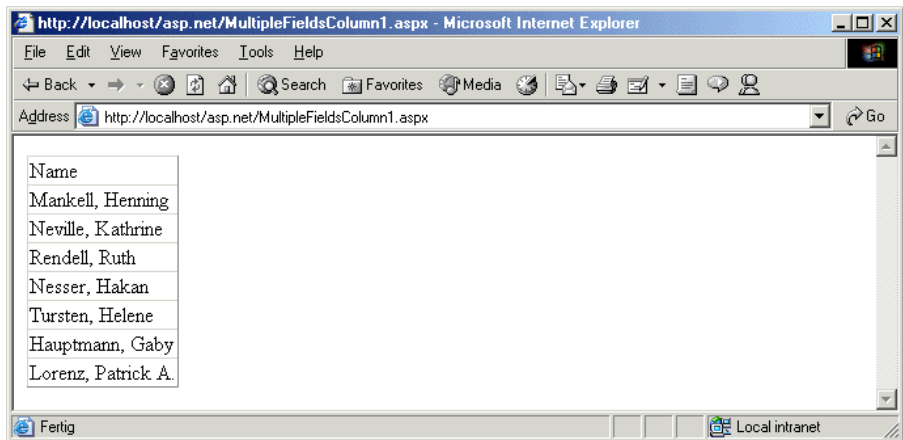


Abbildung 6.68 Der neue Spaltentyp kann mehrere datengebundene Felder darstellen.

Statt die Spalte programmatisch anzulegen, möchten Sie diese vermutlich wie gewohnt innerhalb des Layout-Bereichs anlegen. Auch das ist kein Problem, der Namespace muss lediglich über die `@Register`-Direktive mit einem Tag-Präfix versehen werden. Das Listing zeigt's; die Ausgabe entspricht der vorherigen Abbildung.

Listing 6.84 MultipleFieldsColumn2.aspx

```
<% @Register TagPrefix="PAL" Namespace="PAL.Projects.AspNetKochbuch"
    Assembly="MultipleFieldsColumn" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();

    conn.Close();
}

</script>

<form runat="server">

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID">

    <Columns>

        <PAL:MultipleFieldsColumn
            HeaderText="Name"
            DataFields="Firstname, Lastname"
            DataFormatString="{1}, {0}"/>

    </Columns>

</ASP:DataGrid>

</form>
```

Eine weitere, interessante Idee für einen Spaltentyp ist der Typ `DetailColumn`. Ähnlich wie im Rezept „... einen hierarchischen Datensatz über das `DataGrid`-Control anzeigen?“ im Kapitel „Datenbanken“ über eine `TemplateColumn` realisiert, könnte diese Spalte über eine `DataRelation` verknüpfte Unterdatensätze anzeigen.

Das folgende Listing realisiert genau das. Der Ansatz entspricht dem vorherigen, allerdings wird diesmal innerhalb jeder Zelle ein DataList-Control platziert. Die hierzu notwendige ItemTemplate-Vorlage wird über eine analoge Eigenschaft der Spalte durchgereicht.

Listing 6.85 DetailColumn.cs

```
using System;
using System.Collections;
using System.Data;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace PAL.Projects.AspNetKochbuch
{
    public class DetailColumn : DataGridColumn
    {
        protected ITemplate _ItemTemplate;

        public string DataRelation
        {
            get { return((string) ViewState["DataRelation"]); }
            set { ViewState["DataRelation"] = value; }
        }

        public ITemplate ItemTemplate
        {
            get { return(_ItemTemplate); }
            set { _ItemTemplate = value; }
        }

        public override void InitializeCell(TableCell cell,
            int columnIndex, ListItemType itemType)
        {
            base.InitializeCell(cell, columnIndex, itemType);

            if(itemType == ListItemType.Item ||
                itemType == ListItemType.AlternatingItem)
            {
                DataList dl = new DataList();
                dl.ItemTemplate = this.ItemTemplate;
                cell.Controls.Add(dl);

                cell.DataBinding += new EventHandler(cell_DataBinding);
            }
        }

        protected void cell_DataBinding(object sender, EventArgs e)
```

```
{
    if(this.DataRelation == null)
        throw(new ArgumentException("Property DataRelation
            can not be null"));

    TableCell cell = (TableCell) sender;
    DataList dl = (DataList) cell.Controls[0];
    DataGridItem dgi = (DataGridItem) cell.NamingContainer;
    DataRowView row = (DataRowView) dgi.DataItem;
    DataRow[] rows = row.Row.GetChildRows(this.DataRelation);
    dl.DataSource = rows;
}
}
```

Die Spalte ist genauso effektiv wie einfach zu implementieren. Auch die Nutzung ist absolut trivial. Das folgende Beispiel zeigt dies anhand der relational verknüpften Tabellen Authors und Books. Das so erzeugte DataGrid-Control zeigt alle zu einem Autoren gehörenden Bücher an.

Listing 6.86 DetailColumn1.aspx

```
<% @Register TagPrefix="PAL" Namespace="PAL.Projects.AspNetKochbuch"
    Assembly="DetailColumn" %>

...

<form runat="server">

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID">

    <Columns>

        <ASP:BoundColumn
            HeaderText="Vorname"
            DataField="Firstname"/>

        <ASP:BoundColumn
            HeaderText="Nachname"
            DataField="Lastname"/>

        <PAL:DetailColumn
            HeaderText="Verfügbare Bücher"
            DataRelation="Authors2Books">

            <ItemTemplate>
                &bull; <%# DataBinder.Eval(Container, "DataItem[Title]") %>
```

```

        </ItemTemplate>

    </PAL:DetailColumn>

</Columns>

</ASP:DataGrid>

</form>

```

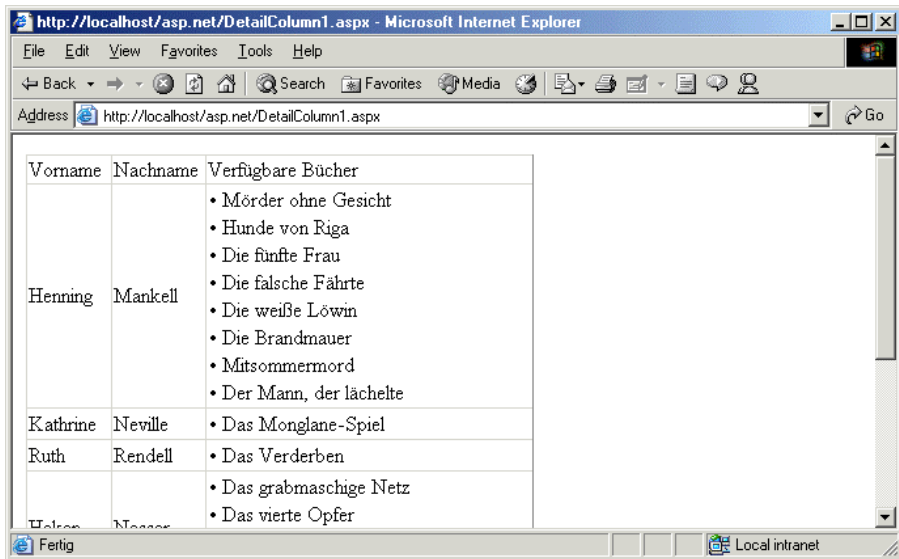


Abbildung 6.69 Die Spalte zeigt die Unterdatensätze einer übergebenen Relation an.

6.43 ... ein DataGrid sortieren?

Das DataGrid-Control unterstützt von Haus aus die individuelle Sortierung von Datensätzen durch den Benutzer. Um dieses Feature zu aktivieren, muss die Option `AllowSorting` explizit aktiviert werden. Anschließend werden für alle automatisch angelegten Spalten `LinkButton`-Controls in der Kopfzeile angelegt. Ein Klick auf den Button resultiert im Ereignis `SortCommand`, wo die Daten entsprechend der gewählten Spalte sortiert und wieder angezeigt werden können.

Das folgende Beispiel zeigt die Verwendung der Technik. Das Listing stammt aus dem Rezept „... Datensätze sortieren?“, in dem die konkurrierenden Möglichkeiten zur Sortierung per SQL und der `DataSet`-Klasse beschrieben werden.

Listing 6.87 Sort2.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        BindData();
}

void BindData()
{
    this.BindData(null);
}

void BindData(string sortExpression)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors";

    if (sortExpression != null)
        SQL += (" ORDER BY " + sortExpression);

    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();

    conn.Close();
}

void dg_Sort(object sender, DataGridSortCommandEventArgs e)
{
    BindData(e.SortExpression);
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        DataKeyField="ID"
        AllowSorting="true"
        OnSortCommand="dg_Sort"/>

</form>
```

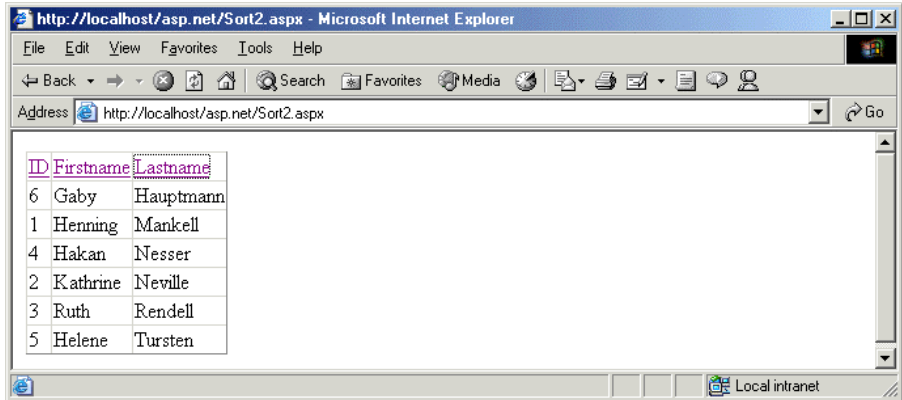



Abbildung 6.70 Mit Hilfe des DataGrid-Controls lassen sich Daten sehr einfach sortieren.

Sofern Sie die Option `AutoGenerateColumns` deaktivieren und die gewünschten Spalten manuell anlegen, werden die zur Sortierung benötigten `LinkButton`-Controls im Kopfbereich nicht mehr angelegt. Sie müssen in diesem Fall der Eigenschaft `SortExpression` explizit den gewünschten Sortierausdruck zuweisen, in der Regel den Namen des angezeigten Feldes. Der geänderte HTML-Bereich der Seite sieht so aus:

Listing 6.88 Sort4.aspx

```
<ASP:DataGrid id="dg" runat="server"
  DataKeyField="ID"
  AllowSorting="true"
  AutoGenerateColumns="false"
  OnSortCommand="dg_Sort">

  <Columns>

    <ASP:BoundColumn
      HeaderText="Vorname"
      SortExpression="Firstname"
      DataField="Firstname"/>

    <ASP:BoundColumn
      HeaderText="Nachname"
      SortExpression="Lastname"
      DataField="Lastname"/>

  </Columns>

</ASP:DataGrid>
```

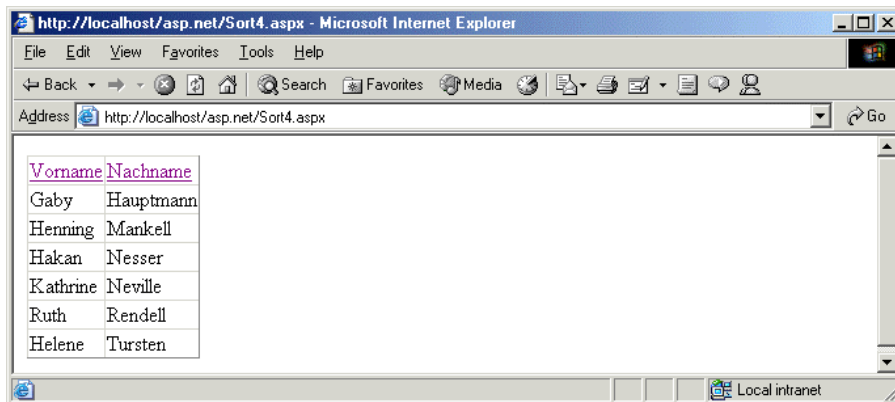


Abbildung 6.71 Auch manuell angelegte Spalten lassen sich sortieren.

6.44 ... ein DataGrid automatisch seitenweise darstellen?

Sollen große Datenmengen in einem DataGrid-Control angezeigt werden, so verbieten die Gebote der Übersichtlichkeit die vollständige Anzeige. Wie beispielsweise bei Suchmaschinen absolut üblich, sollten die Daten über mehrere Seiten verteilt dargestellt werden. Man nennt diese Aufteilung Paging. Das DataGrid-Control unterstützt von Haus aus Paging.

Um die automatische seitenweise Darstellung des Controls zu aktivieren, sind mindestens zwei Schritte notwendig. Zunächst muss die Eigenschaft `AllowPaging` aktiviert werden. Anschließend muss eine Behandlung für das Ereignis `PageIndexChanged` implementiert werden. Über die Ereignisargumente wird der Index der neuen Seite übergeben, der der Eigenschaft `CurrentPageIndex` zugewiesen werden muss.

Listing 6.89 Paging1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack) BindDataGrid();
}

void BindDataGrid()
{
    ...

    DataTable authors = dataset.Tables["Authors"];
```

```

DataColumn id = authors.Columns["ID"];

DataTable books = dataset.Tables["Books"];
DataColumn authorid = books.Columns["AuthorID"];

DataRelation relation = new DataRelation("Authors2Books",
    id, authorid);
dataset.Relations.Add(relation);

books.Columns.Add(new DataColumn("Author", typeof(string),
    "Parent.Firstname + ' ' + Parent.Lastname"));

books.DefaultView.Sort = "Author, Title";
dg.DataSource = books;
DataBind();
}

void dg_PageIndexChanged(object sender,
    DataGridPageChangedEventArgs e)
{
    dg.CurrentPageIndex = e.NewPageIndex;
    BindDataGrid();
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        DataKeyField="ID"
        AutoGenerateColumns="false"
        AllowPaging="true"
        OnPageIndexChanged="dg_PageIndexChanged">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Autor"
                DataField="Author"/>

            <ASP:BoundColumn
                HeaderText="Buchtitel"
                DataField="Title"/>

        </Columns>

    </ASP:DataGrid>

</form>

```

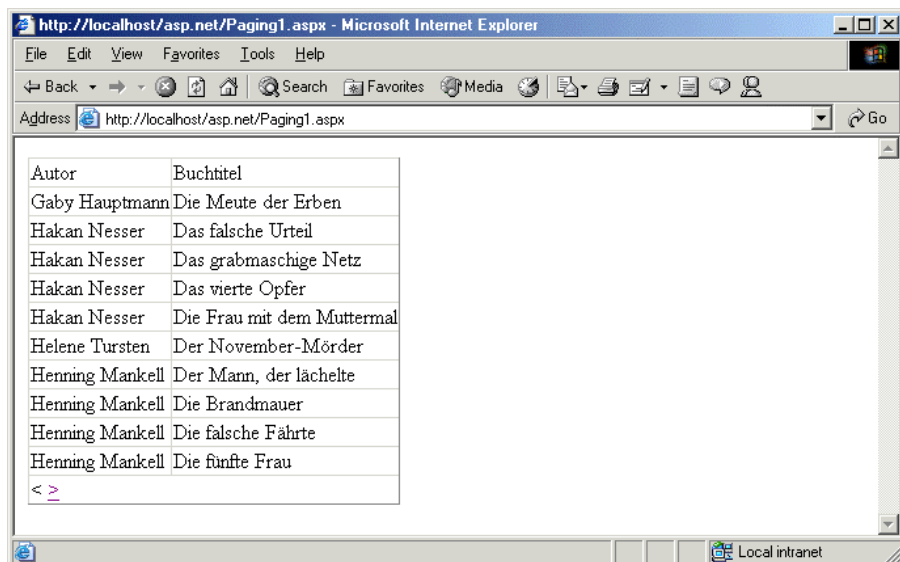


Abbildung 6.72 Die ersten zehn Bücher aus der Datenbank

Das Listing zeigt eine beispielhafte Implementierung der automatischen Paging-Möglichkeiten. Über die beiden Links am Ende lässt sich zwischen den Seiten navigieren. Es wird der Inhalt der Tabelle „Books“ angezeigt. Der Name des Autors wird über eine dynamisch angelegte Spalte auf Basis der relationalen Verknüpfung der Autorentabelle ermittelt.

Standardmäßig ist die Anzahl der pro Seite angezeigten Einträge auf zehn eingestellt. Sie können diese Anzahl über die Eigenschaft `PageSize` beliebig ändern. Das Erscheinungsbild der Navigationselemente lässt sich zudem über die Eigenschaft `PagerStyle` verändern. Das zweite Listing ist eine optisch angepasste Version des Beispiels.

Listing 6.90 Paging2.aspx

```
...  
  
<form runat="server">  
  
<ASP:DataGrid id="dg" runat="server"  
    DataKeyField="ID"  
    AutoGenerateColumns="false"  
    AllowPaging="true"  
    PageSize="5"  
    OnPageIndexChanged="dg_PageIndexChanged">
```

```
<PagerStyle
  Mode="NumericPages"
  Position="TopAndBottom"
  HorizontalAlign="Right"
  Font-Bold="true" />
```

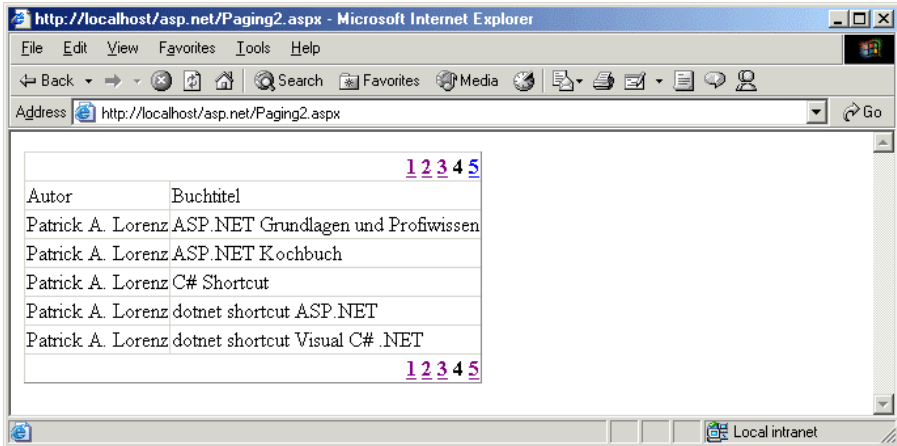


Abbildung 6.73 Das Erscheinungsbild der Navigation lässt sich individuell anpassen.



Wie die beiden Beispiele gezeigt haben, funktioniert das automatische Paging sehr problemlos. Dennoch hat dieses System einen Haken. Obwohl nur ein kleiner Teil der Daten angezeigt werden soll, werden alle aus der Datenbank ausgelesen und im Arbeitsspeicher abgelegt. Muss das sein? Nein, denn beim manuellen Paging haben Sie mehr Möglichkeiten ...

6.45 ... ein DataGrid manuell seitenweise darstellen?

Das automatische Paging des DataGrid-Controls hat den entscheidenden Nachteil, dass alle angezeigten und nicht angezeigten Daten aus der Datenbank gelesen werden. Insbesondere bei sehr umfangreichen Mengen sollte das manuelle Paging bevorzugt werden. Hier müssen nur die Daten ausgelesen werden, die auch wirklich an den Browser gesendet werden.

Das manuelle Paging wird über die Eigenschaft `AllowCustomPaging` aktiviert. Außerdem muss der Eigenschaft `VirtualItemCount` die Gesamtanzahl der Elemente zugewiesen werden. Beim automatischen Paging wird dieser Wert aus der Anzahl der übergebenen Datensätze ermittelt werden, bei der manuellen Variante ist dies natürlich nicht möglich.

Das Beispiel basiert auf dem vorherigen Rezept und ergänzt dies um das manuelle Paging. Beim ersten Aufruf wird die tatsächliche Gesamtanzahl der Datensätze ermittelt. Wie dies geht, erfahren Sie im Rezept „... die Anzahl von Datensätzen in einer Tabelle ermitteln?“ im Kapitel „Datenbanken“. Anschließend werden mit Hilfe des SQL-Befehls `TOP` die zur Anzeige benötigten Datensätze abgefragt.

Listing 6.91 Paging3.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        BindDataGrid(-1);
}

void BindDataGrid(int pageIndex)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    if(pageIndex == -1)
    {
        string SQLcount = "SELECT Count(*) FROM Books;";
        OleDbCommand cmdCount = new OleDbCommand(SQLcount, conn);
        dg.VirtualItemCount = (int) cmdCount.ExecuteScalar();
    }

    int pageCount = 5;
    int pageStart = (pageIndex == -1 ? 1 : pageIndex * pageCount + 1);

    string SQL = string.Format("SELECT TOP {0} * FROM Books WHERE ID>={1}
        ORDER BY ID", pageCount, pageStart);
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();

    DataBind();

    conn.Close();
}
```

```

void dg_PageIndexChanged(object sender,
    DataGridPageChangedEventArgs e)
{
    dg.CurrentPageIndex = e.NewPageIndex;
    BindDataGrid(e.NewPageIndex);
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        DataKeyField="ID"
        AutoGenerateColumns="false"
        AllowPaging="true"
        AllowCustomPaging="true"
        PageSize="5"
        OnPageIndexChanged="dg_PageIndexChanged">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Buchtitel"
                DataField="Title"/>

        </Columns>

    </ASP:DataGrid>

</form>

```

Das Beispiel wurde auf die Anzeige der Tabelle „Books“ begrenzt. Die relational verknüpften Autorennamen wurden weggelassen. Dies hat mehrere Gründe. Einerseits ist das Listing so kompakter, und andererseits würde eine Abfrage *aller* Autoren dem Gedanken des Pagings entgegenstehen. Hier sollte zuvor auf Basis der abgefragten Buchtitel sichergestellt werden, dass nur die benötigten Autoren ausgelesen werden. Hierzu könnte beispielsweise folgende Query benutzt werden:

Listing 6.92 Paging4.aspx

```

...

string authorsSQL = string.Empty;
foreach(DataRow row in dataset.Tables["books"].Rows)
{
    if(authorsSQL.Length > 0) authorsSQL += ", ";
    authorsSQL += row["AuthorID"];
}

```

```
}

authorsSQL = string.Format("SELECT * FROM Authors WHERE ID IN ({0})",
    authorsSQL);
OleDbDataAdapter adapter1 =
    new OleDbDataAdapter(new OleDbCommand(authorsSQL, conn));
adapter1.Fill(dataset, "Authors");

...
```

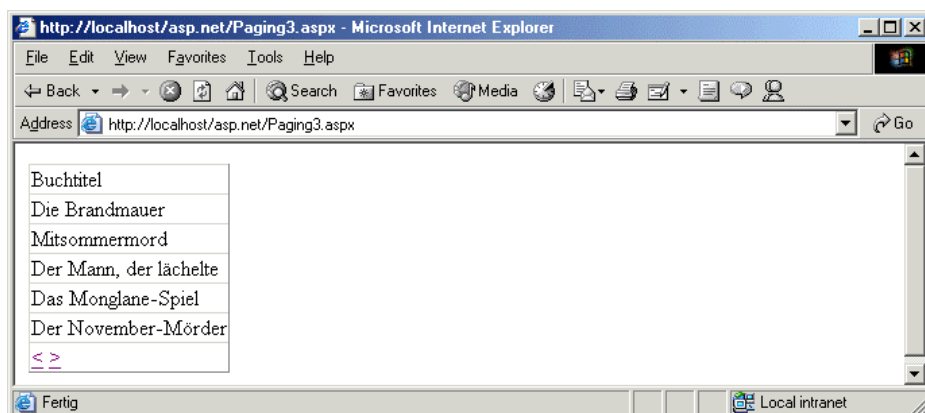


Abbildung 6.74 Auch das manuelle Paging zeigt die korrekten Datensätze an.

Das vorliegende Beispiel funktioniert nur so lange, wie der Wert des Feldes ID fortlaufend ist. Ansonsten kommt es zu Überschneidungen bei der seitenweisen Anzeige der Daten. Beachten Sie dies unbedingt bei der Verwendung des manuellen Pagings.

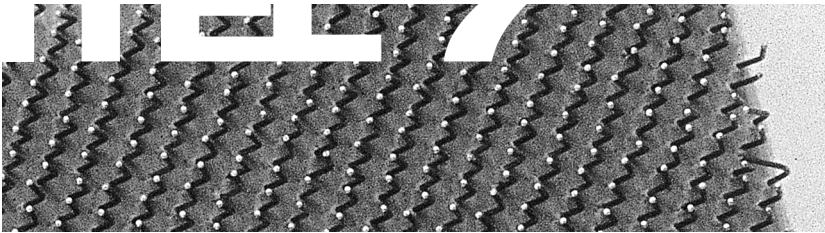
Alternativ können Sie die Arbeit auch der Methode `Fill` der Klasse `OleDbDataAdapter` überlassen. Einer Überladung können der Startwert sowie die Anzahl der Datensätze übergeben werden, die in die `DataSet`-Instanz kopiert werden. Hierbei werden zwar alle Daten über die Datenbank-Engine abgefragt, jedoch nicht in den Arbeitsspeicher kopiert. Ein mitunter gangbarer Weg.

Listing 6.93 Paging5.aspx

```
...  
  
int pageCount = 5;  
int pageStart = (pageIndex == -1 ? 1 : pageIndex * pageCount + 1);  
  
string SQL = "SELECT * FROM Books ORDER BY ID";  
OleDbDataAdapter adapter2 =  
    new OleDbDataAdapter(new OleDbCommand(SQL, conn));  
adapter2.Fill(dataset, pageStart, pageCount, "Books");  
  
...
```

7

User Controls



Wie kann ich ...

7 User Controls

Auch User Controls und Custom Controls sind für den Einsatz auf Web-Formularen gedacht. Hier nehmen allerdings Sie das Ruder in die Hand. In diesem Kapitel erfahren Sie nützliche Praxisratschläge im Umgang mit den benutzerdefinierten Objekten. Alles im Lot auf'm Boot.

7.1 ... Werte an ein User Control übergeben?

User Controls sind ein enormer Fortschritt im Hinblick auf die Wiederverwendbarkeit von Layout und Quellcode. Mit einfachen Mitteln lassen sich wiederkehrende Elemente auf mehreren Unterseiten darstellen. Soll der Inhalt geändert werden, reicht eine zentrale Aktualisierung des User Controls aus, und alle Seiten sind sofort auf dem neuesten Stand.

Ihre besonderen Fähigkeiten können User Controls erst ausspielen, wenn sie statt statischen Layouts dynamische Inhalte zur Verfügung stellen. Es muss eine Interaktion zwischen dem Control und der aufrufenden Seite stattfinden. Einige Ansätze sehen die Übergabe von Daten im Query-String vor, auf den das Control über das reguläre Objektmodell zugreifen kann. Dies entspricht jedoch weder zeitgemäßer ASP.NET-Entwicklung noch ist dieser Ansatz besonders flexibel.

Aus dem Blickwinkel der objektorientierten Programmierung betrachtet, handelt es sich bei einem User Control um eine Ableitung der gleichnamigen Klasse. Das dahinter stehende System ist mit den regulären ASP.NET-Seiten zu vergleichen mit dem Unterschied, dass die Ableitung dort auf der Klasse `Page` basiert. In beiden Fällen lässt sich die Klasse im Script-Bereich mit neuen Methoden und auch Eigenschaften erweitern. Diese Möglichkeiten können Sie sich zunutze machen, um eine Schnittstelle zwischen der Seite und dem Control zu realisieren.

Das folgende Listing zeigt ein sehr einfaches User Control. Dieses implementiert eine einzige Eigenschaft `Text`. Diese wird auf ein im Layout hinterlegtes Label-Control gespiegelt. Über die Eigenschaft lässt sich somit indirekt der Inhalt des Labels setzen und abfragen. Durch die Einrahmung in ein `h1`-Tag wird der Text als Überschrift dargestellt.

Listing 7.1 UserControlProperties1.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

public string Text
{
    get { return(lbl.Text); }
    set { lbl.Text = value; }
}

</script>

<h1> <ASP:Label id="lbl" runat="server"/> </h1>
```

Das zweite Listing zeigt eine ASP.NET-Seite, in der das Control mit dem Tag-Namen „Header“ registriert und im Layout hinterlegt wurde. Über die oben beschriebene Eigenschaft wird direkt bei der Definition des Controls die anzuzeigende Überschrift zugewiesen.

Listing 7.2 UserControlProperties1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlProperties1.ascx" %>

<PAL:Header runat="server" text="Hallo Welt!"/>
```

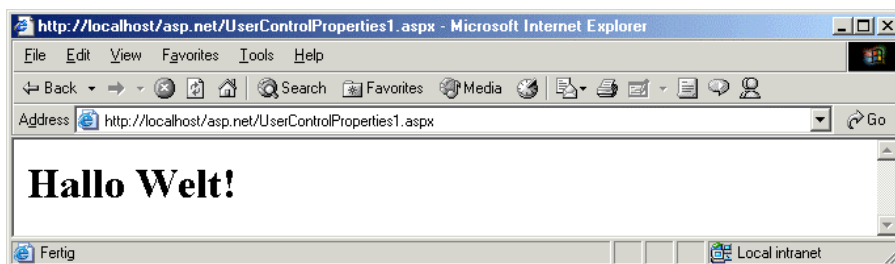


Abbildung 7.1 Das User Control stellt den zugewiesenen Text als Überschrift dar.

Selbstverständlich können Sie den Inhalt der Eigenschaft nicht nur bei der Definition, sondern auch zur Laufzeit festlegen. Das User Control unterscheidet sich in diesem Punkt nicht von anderen Klassen. Im folgenden Listing kann der anzuzeigende Text beispielsweise vom Benutzer eingegeben werden.

Listing 7.3 UserControlProperties1a.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlProperties1.ascx" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    myheader.Text = tb.Text;
}

</script>

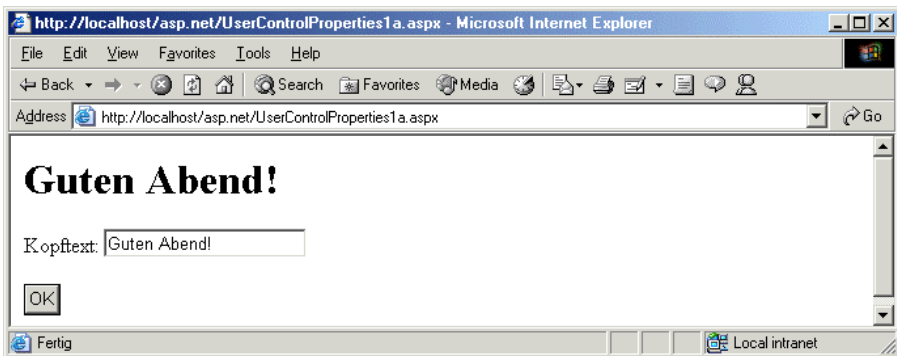
<form runat="server">

    <PAL:Header id="myheader" runat="server" text="Hallo Welt!"/>

    <p> Kopftext: <ASP:TextBox id="tb" runat="server"
        text="Hallo Welt!"/> </p>

    <p> <ASP:Button runat="server" text="OK" onclick="bt_click"/> </p>

</form>
```

**Abbildung 7.2** Der Text wurde dem User Control zur Laufzeit zugewiesen.

Statt im User Control die Eigenschaft eines anderen Controls zu spiegeln, können Sie auch unabhängige Eigenschaften implementieren. Wie bei der Klassenprogrammierung üblich, verwenden Sie dazu ein global als `protected` definiertes Mitgliedsfeld, in dem der Inhalt abgespeichert wird.

Das zweite Beispiel zeigt eine Erweiterung des ersten. Es werden nun zwei Eigenschaften `Title` und `Subtitle` angeboten. Der Inhalt der Letzteren wird innerhalb eines `h2`-Tags ausgegeben, allerdings nur wenn ein Inhalt übergeben wurde. Die Ausgabe der beiden Texte erfolgt mit Hilfe von `Response.Write` innerhalb einer Überschreibung der als `protected` markierten Methode `OnPreRender`.

Listing 7.4 UserControlProperties2.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

protected string m_Title;
protected string m_Subtitle;

public string Title
{
    get { return(m_Title); }
    set { m_Title = value; }
}

public string Subtitle
{
    get { return(m_Subtitle); }
    set { m_Subtitle = value; }
}

protected override void OnPreRender(EventArgs e)
{
    Response.Write("<h1>" + this.Title + "</h1>");

    if((this.Subtitle != null) && (this.Subtitle != string.Empty))
        Response.Write("<h2>Untertitel: " + this.Subtitle + "</h2>");

    base.OnPreRender(e);
}

</script>
```

Beachten Sie, dass innerhalb der überschriebenen Methode *unbedingt* die Basis aufgerufen werden sollte, da ansonsten eine eventuell zugewiesene Ereignisbehandlung für `PreRender` nie ausgelöst werden würde.

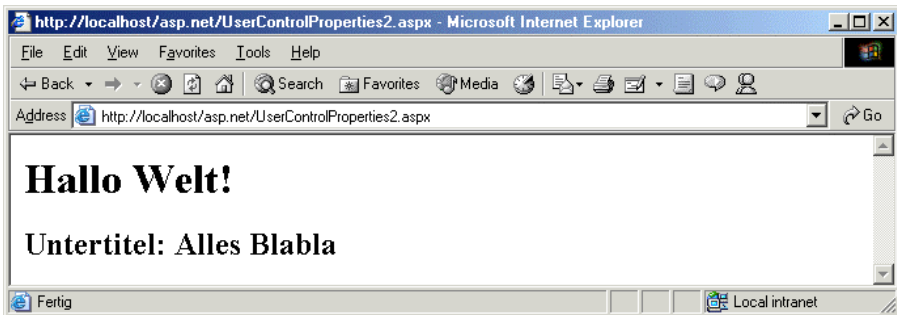
Listing 7.5 UserControlProperties2.aspx

```

<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlProperties2.ascx" %>

<PAL:Header runat="server"
    title="Hallo Welt!"
    subtitle="Alles Blabla"/>

```

**Abbildung 7.3** Der Text wurde wie gewünscht ausgegeben.

Wie Sie den Inhalt von Mitgliedsvariablen auch über PostBacks hinweg persistent halten können, erfahren Sie im folgenden Rezept.

7.2 ... den Status eines User Controls speichern?

Eine wichtige Möglichkeit von ASP.NET Server Controls ist die Tatsache, dass diese ihren Status über mehrere Seitenaufrufe/PostBacks hinweg automatisch erhalten können. Die zur späteren Darstellung benötigten Daten werden serialisiert und in einem versteckten Formularfeld an den Client übertragen. Bei einem PostBack werden die Daten vom Client zurück an den Server gesendet, deserialisiert und zur erneuten Darstellung des Controls genutzt. Ein gutes Beispiel hierfür ist das Label-Control. Wird diesem ein Text zugewiesen, wird er auch nach einem PostBack unverändert angezeigt.

Ausgehend vom Label-Control möchte ich Ihnen ein kleines Experiment vorstellen. Das nachfolgende User Control ist letztlich nichts anderes als ein nachgebautes Label. Der über die Eigenschaft `Text` zugewiesene Inhalt wird unverändert im Browser ausgegeben.

Listing 7.6 UserControlViewState1.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

protected string m_Text;

public string Text
{
    get { return(m_Text); }
    set { m_Text = value; }
}

</script>

<%=this.Text%>
```

Die folgende Seite verwendet das gezeigte User Control. Es existiert ein Eingabefeld, dessen Inhalt beim Klick auf den ersten Button parallel einem Label sowie dem User Control zugewiesen wird. Zudem wurde ein zweiter Button ohne Ereignisbehandlung hinterlegt. Ein Klick auf diesen löst also lediglich einen unbehandeltenPostBack aus.

Listing 7.7 UserControlViewState1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlViewState1.ascx" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    uc.Text = tb.Text;
    lb.Text = tb.Text;
}

</script>

<form runat="server">

<p>
    Eingabe: <ASP:TextBox id="tb" runat="server" text="Hallo Welt!"/>
</p>

<p> User Control: <PAL:Header id="uc" runat="server"/> </p>
<p> Label: <ASP:Label id="lb" runat="server"/> </p>
```



```

<p>
  <ASP:Button runat="server" text="Button1" onclick="bt_click"/>
</p>

<p> <ASP:Button runat="server" text="Button2"/> </p>

</form>

```

Was ist nun der Sinn und Zweck dieses kleinen Experiments? Probieren Sie einfach einmal aus, einen individuellen Text einzugeben. Nun klicken Sie auf den *ersten* Button. Sowohl das User Control als auch das Label zeigen Text an. Nun klicken Sie bitte auf den *zweiten* Button. Experiment geglückt, Patient tot – das User Control ist leer, das Label hingegen nicht. Der Grund für diesen Unterschied ist die Tatsache, dass ein Control selbst für die Unterstützung des ViewStates sorgen muss. Alle zu erhaltenen Daten müssen manuell im ViewState-StateBag abgelegt werden. Dies hat durchaus Sinn, denn würden alle verfügbaren Daten abgespeichert, würde die Bandbreite gerade bei komplexen Formularen deutlich über Gebühr in Anspruch genommen werden.

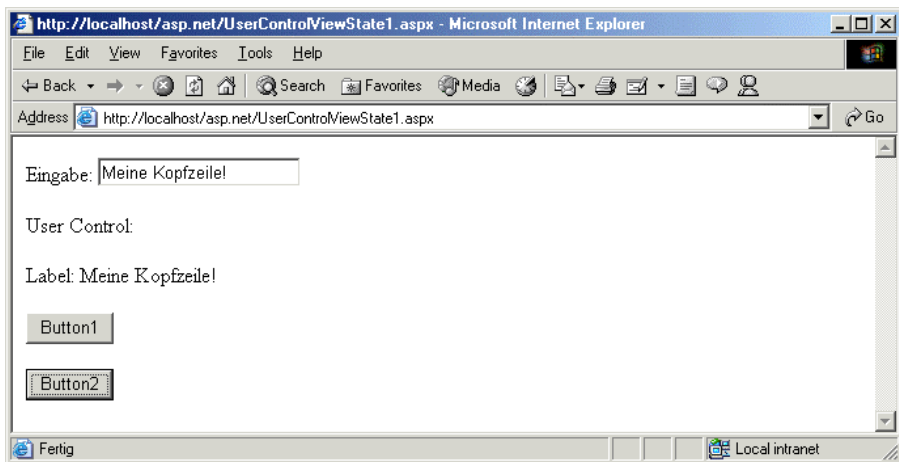


Abbildung 7.4 Der Inhalt des User Controls wurde nicht im ViewState gespeichert.

Hinter der Technik ViewState steht die Klasse `StateBag`. Diese unterstützt unter anderem die Schnittstelle `IDictionary` und agiert somit als ein eben solches Dictionary. Über den Indexer können neue Objekte mit einem Schlüssel versehen zugewiesen und auch wieder abgefragt werden. Der Schlüssel muss eine Zeichenkette sein. Der Wert selbst kann ein beliebiges, serialisierbares Objekt sein.

Das folgende Beispiel erweitert das erste und verwendet zur Speicherung des Eigenschaftsinhalts statt einer Mitgliedsvariablen den ViewState-StateBag.

Listing 7.8 UserControlViewState2.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

protected string m_Text;

public string Text
{
    get { return((string) ViewState["text"]); }
    set { ViewState["text"] = value; }
}

</script>

<%=this.Text%>
```

Wiederholt man das oben geschilderte Experiment mit dem geänderten User Control, wird man feststellen, dass der Inhalt nun auch nach dem zweiten PostBack unverändert zur Verfügung steht. Die Wert der Eigenschaft wurde erfolgreich im ViewState-StateBag abgespeichert.

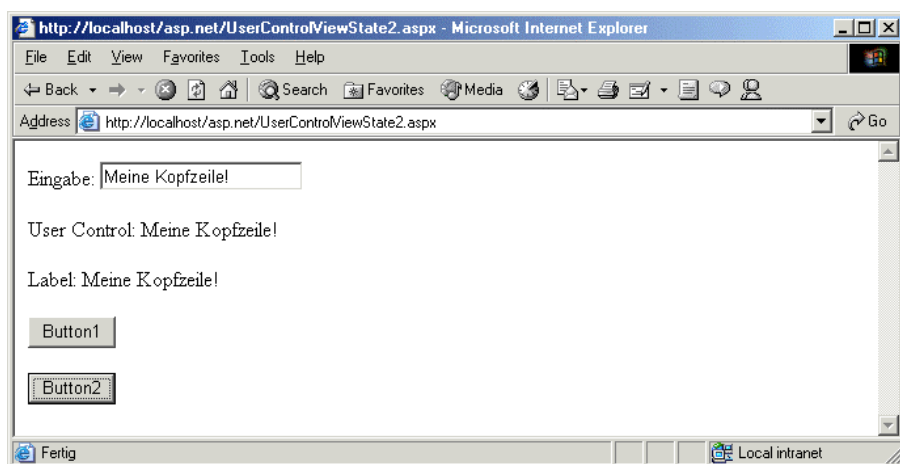


Abbildung 7.5 Der Inhalt des User Controls wurde im ViewState gespeichert.



Wie beschrieben können Sie im Prinzip ein beliebiges Objekt im ViewState ablegen, wenn dies nur serialisierbar ist (Serializable-Attribut). Je umfangreicher ein Objekt, desto größer ist jedoch der Speicherbedarf nach der Serialisierung. Da diese Daten bei jedem PostBack zwischen Client und Server hin und wieder zurück übertragen werden müssen, empfiehlt es sich, möglichst nur Standarddatentypen im ViewState abzulegen.

7.3 ... ein User Control mit Code Behind verwenden?

Code Behind wird verwendet, um Quellcode und Layout einer Seite wirkungsvoll zu trennen. In Form einer regulären Quellcode-Datei wird eine Basisklasse mit den notwendigen Eigenschaften und Methoden implementiert. Die aspx-Datei enthält weiterhin das Layout und wird statt direkt von Page von der Code Behind-Klasse abgeleitet. Die dort hinterlegten Controls werden überschrieben und können so sowohl aus der Basisklasse als auch der Layoutdatei heraus angesprochen werden.

Doch was passiert, wenn ein User Control verwendet und aus der Code Behind-Datei heraus genutzt werden soll? Schauen Sie sich hierzu bitte die drei folgenden Listings an. Sie zeigen nacheinander das User Control, die Code Behind-Datei und die Layoutdatei.

Listing 7.9 UserControlCB1.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

public string Text
{
    get { return(lbl.Text); }
    set { lbl.Text = value; }
}

</script>

<h1> <ASP:Label id="lbl" runat="server"/> </h1>
```

Listing 7.10 UserControlCB1.aspx.cs

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public class CB : Page
{
    protected UserControl myheader;
    protected TextBox tb;
    protected Button bt;

    protected void bt_click(object sender, EventArgs e)
    {
        myheader.Text = tb.Text;
    }
}
```

Listing 7.11 UserControlCB1.aspx

```
<% @Page Language="C#" Debug="true" Inherits="CB"
    Src="UserControlCB1.aspx.cs" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlCB1.ascx" %>

<form runat="server">

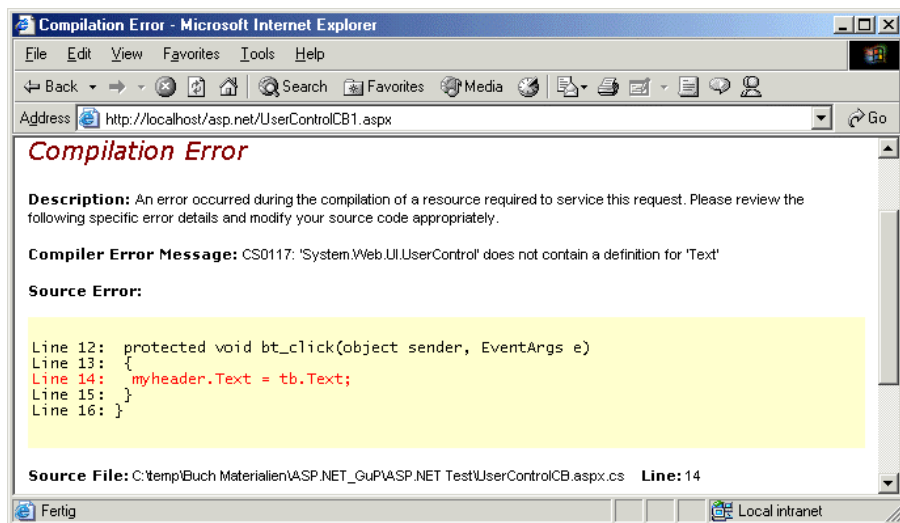
    <PAL:Header id="myheader" runat="server" text="Hallo Welt!"/>

    <p>
        Kopftext: <ASP:TextBox id="tb" runat="server" text="Hallo Welt!"/>
    </p>

    <p>
        <ASP:Button id="bt" runat="server" text="OK" onclick="bt_click"/>
    </p>

</form>
```

Prinzipiell ist der Quellcode in der vorliegenden Form korrekt, verwenden lässt er sich dennoch nicht. Das Problem ist die Definition des User Controls in der Code Behind-Klasse. Als Datentyp ist hier `UserControl` angegeben. In der Ereignisbehandlung des Buttons wird nun versucht, auf die Eigenschaft `Text` zuzugreifen. Dies misslingt jedoch, da die Klasse `UserControl` nicht über ein derartiges Mitglied verfügt. Aus Sicht der objektorientierten Programmierung ist der in der Abbildung gezeigte Kompilierungsfehler daher vollkommen rechters.

**Abbildung 7.6** Die Eigenschaft `Text` wird nicht gefunden.

Was also tun, um Zugriff auf die benötigte Eigenschaft zu erhalten? Es gibt zwei konkurrierende Ansätze, die ich Ihnen nachfolgend vorstelle.

Reflections

Die neue .NET-Technik Reflections ist ein wahrer Segen, wenn es um den dynamischen Zugriff auf unbekannte Objekte geht. So können Sie beispielsweise auf Eigenschaften zugreifen, die Sie nur vom Namen her kennen. Genau dies können Sie sich zur Verwendung des User Controls mit Code Behind zunutze machen. Das Listing zeigt die entsprechend geänderte Code Behind-Klasse; alles andere bleibt wie gehabt.

Listing 7.12 UserControlCB2.aspx.cs

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Reflection;

public class CB : Page
{
    protected UserControl myheader;
    protected TextBox tb;
    protected Button bt;

    protected void bt_click(object sender, EventArgs e)
    {
        Type t = myheader.GetType();
        PropertyInfo p = t.GetProperty("Text");
        p.SetValue(myheader, tb.Text, null);
    }
}
```

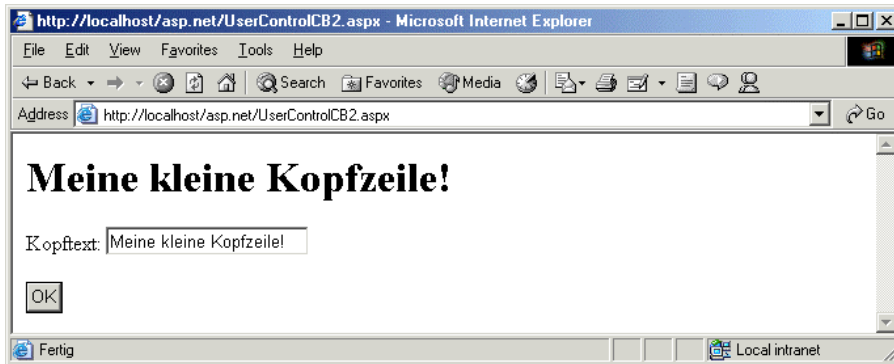


Abbildung 7.7 Der Wert der Eigenschaft wurde per Reflections zugewiesen.

Die Abbildung zeigt deutlich, dass die Zuweisung nun problemlos funktioniert. Im Listing wurde zunächst eine Instanz der Klasse `Type` mit Hilfe der von `object` geerbten Methode `GetType` angelegt. Mit `GetProperty` kann eine namentlich bekannte Eigenschaft abgerufen werden. Über die zurückgelieferte `PropertyInfo`-Instanz kann nun der Wert der Eigenschaft ausgelesen oder auch geändert werden. Hierzu wird wie im Beispiel gezeigt die Methode `SetValue` verwendet.

Zur Vereinfachung können Sie sich selbstverständlich eine kleine Hilfsfunktion schreiben, die Ihnen insbesondere bei vielen Eigenschaften einiges an Tipparbeit sparen kann. Nachfolgend sehen Sie eine entsprechende Implementierung mit zwei statischen Methoden `GetValue` und `SetValue` zum Abfragen und Setzen von Eigenschaften.

Listing 7.13 `UserControlCB3.aspx.cs`

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Reflection;

public class CB : Page
{
    protected UserControl myheader;
    protected TextBox tb;
    protected Button bt;

    protected void bt_click(object sender, EventArgs e)
    {
        ReflHelp.SetValue(myheader, "Text", tb.Text);
    }
}
```

```

public class ReflHelp
{
    public static object GetValue(object obj, string property)
    {
        Type t = obj.GetType();
        PropertyInfo p = t.GetProperty(property);
        return(p.GetValue(obj, null));
    }

    public static void SetValue(object obj, string property,
        object value)
    {
        Type t = obj.GetType();
        PropertyInfo p = t.GetProperty(property);
        p.SetValue(obj, value, null);
    }
}

```

Falls Sie sich wundern, welchen Zweck der jeweils letzte Parameter der Originalmethoden `GetValue` und `SetValue` der Klasse `PropertyInfo` hat, so denken Sie an indizierte Eigenschaften. Sofern eine Eigenschaft (beziehungsweise eine Klasse) über einen Indexer verfügt, kann hier statt `null` der zu verwendende Indexwert übergeben werden.

Code Behind x 2

Eine Alternative zu der Verwendung von Reflections ist die doppelte Verwendung von Code Behind. Gemeint ist die konsequente Nutzung der Technik auch für das User Control. Dies ist auch der empfohlene beziehungsweise eigentlich dafür gedachte Ansatz.

Im Folgenden sehen Sie die Code Behind-Datei. Neben der eigentlichen Seite ist nun auch eine Ableitung der Klasse `UserControl` vorhanden. Diese implementiert die benötigte Eigenschaft `Text` und enthält auch das zur Zuweisung benötigte Label-Control. Innerhalb der Seite wird diese Klasse genutzt, um auf die Eigenschaft zuzugreifen.

Listing 7.14 UserControlCB4.aspx.cs

```

using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Reflection;

```

```
public class CB : Page
{
    protected HeaderCB myheader;
    protected TextBox tb;
    protected Button bt;

    protected void bt_click(object sender, EventArgs e)
    {
        myheader.Text = tb.Text;
    }
}

public class HeaderCB : UserControl
{
    protected Label lbl;

    public string Text
    {
        get { return(lbl.Text); }
        set { lbl.Text = value; }
    }
}
```

Die Nutzung der Code Behind-Klasse innerhalb des User Controls funktioniert analog wie bei der Seite über die Attribute `Inherits` und `Src` der `@Control`-Direktive. Der Umfang des Controls verringert sich dadurch um den Script-Bereich.

Listing 7.15 UserControlCB4.ascx

```
<% @Control Language="C#" Debug="true" Inherits="HeaderCB"
    Src="UserControlCB4.aspx.cs" %>

<h1> <ASP:Label id="lbl" runat="server"/> </h1>
```

Gerade bei umfangreichen Projekten kann die Ablage der Klassen in einer einzelnen Quellcode-Datei unübersichtlich werden. In diesem Fall können Sie die Code Behind-Klassen für Seite und User Control auf zwei Dateien verteilen. Damit diese jedoch untereinander referenziert werden können, sollten Sie beide Dateien auf einen Schlag mittels des Kommandozeilenprogramms `csc.exe` zu einer DLL kompilieren. Wie dies geht, erfahren Sie im Rezept „... eine Code Behind-Datei kompilieren?“ im Kapitel „Basics“.

7.4 ... Eingabefelder in User Controls verwenden?

Innerhalb eines User Controls können Sie analog zu einer regulären Seite beliebige Controls ablegen. Genau dies ist ja auch Sinn und Zweck eines solches Objektes. Selbstverständlich können Sie auch ohne besondere Einschränkungen Eingabeelemente wie das TextBox-Control verwenden. Die beiden folgenden Listings zeigen dies. Das User Control enthält ein Eingabefeld, über das der Benutzer den Überschrifttext ändern kann. Die anschließend gezeigte Seite bindet das Control inklusive dem Eingabeformular ein.

Listing 7.16 UserControlInput1.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

public string Text
{
    get { return(lbl.Text); }
    set { lbl.Text = value; }
}

void bt_click(object sender, EventArgs e)
{
    this.Text = tb.Text;
}

</script>

<h1> <ASP:Label id="lbl" runat="server"/> </h1>

<p> Kopftext: <ASP:TextBox id="tb" runat="server" text="Hallo Welt!"/>
</p>

<p> <ASP:Button runat="server" text="OK" onclick="bt_click"/> </p>
```

Listing 7.17 UserControlInput1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlInput1.ascx" %>

<form runat="server">

    <PAL:Header runat="server" text="Hallo Welt!"/>

</form>
```



Abbildung 7.8 Die Eingabeelemente sind im User Control abgelegt.



Wenn die Einbindung so einfach ist, warum dann ein ganzes Rezept zum Thema schreiben? Der Grund hierfür ist mehr ein Tipp denn ein Lösungsvorschlag. Eingabeelemente wie TextBox- und Button-Controls benötigen ein serverseitiges form-Tag. Sie sollten dieses *unbedingt* innerhalb der Seite und *niemals* innerhalb des User Controls ablegen. Ansonsten könnten Sie innerhalb der Seite selbst keine anderen Eingabeelemente mehr verwenden. So ist es richtig:

```
<form runat="server">

  <PAL:Header runat="server" text="Hallo Welt!"/>

</form>
```

7.5 ... ein User Control mit Ereignissen ausrüsten?

Ereignisse bieten eine attraktive Form zur bedarfsgerechten Kommunikation zwischen zwei Objekten wie beispielsweise User Control und Seite an. Was Sie bereits von den regulären Web Controls kennen, können Sie daher problemlos auch in eigenen Objekten einsetzen. In diesem Rezept erfahren Sie, wie dies geht.

Einfache Ereignisse ohne Ereignisargumente

Ein einfaches Ereignis ist schnell implementiert. Dies sollte jedoch in jedem Fall den üblichen Designanforderungen von .NET genügen. Diese sehen vor, dass ein Ereignis mit einem Delegate definiert wird. Als Parameter werden das auslösende

Objekt sowie die Ereignisargumente übergeben. Statt das Ereignis direkt auszulösen, wird hierzu eine spezielle, als `protected virtual` markierte `On...`-Methode implementiert.

Das folgende Listing zeigt die einfache Realisierung eines Ereignisses. Das User Control enthält ein Eingabeformular. Ändert der Benutzer den, so wird dieser auf ein Label-Control abgebildet und ein Ereignis `TextChanged` ausgelöst. Es kommt der eben kurz beschriebene Design-Guide zum Einsatz. Da keine Ereignisargumente vorhanden sind, wird stattdessen `EventArgs.Empty` übergeben.

Listing 7.18 UserControlEvent1.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

public event EventHandler TextChanged;

public string Text
{
    get { return(lbl.Text); }
    set { lbl.Text = value; }
}

void bt_click(object sender, EventArgs e)
{
    if(this.Text != tb.Text)
    {
        this.Text = tb.Text;
        OnTextChanged(EventArgs.Empty);
    }
}

protected virtual void OnTextChanged(EventArgs e)
{
    if(TextChanged != null)
        TextChanged(this, e);
}

</script>

<h1> <ASP:Label id="lbl" runat="server"/> </h1>

<p> Kopfertext: <ASP:TextBox id="tb" runat="server" text="Hallo Welt!"/>
</p>

<p> <ASP:Button runat="server" text="OK" onclick="bt_click"/> </p>
```

Das zweite Listing zeigt die Verwendung des User Controls samt des Ereignisses in einer Seite. Wie Sie sehen, unterscheidet sich die Zuweisung und die Behandlung nicht von den Ereignissen anderer Controls.

Listing 7.19 UserControlEvent1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlEvent1.ascx" %>

<script runat="server">

void Header_TextChanged(object sender, EventArgs e)
{
    Response.Write("Text wurde geändert");
}

</script>

<form runat="server">

<PAL:Header runat="server"
    text="Hallo Welt!"
    OnTextChanged="Header_TextChanged"/>

</form>
```

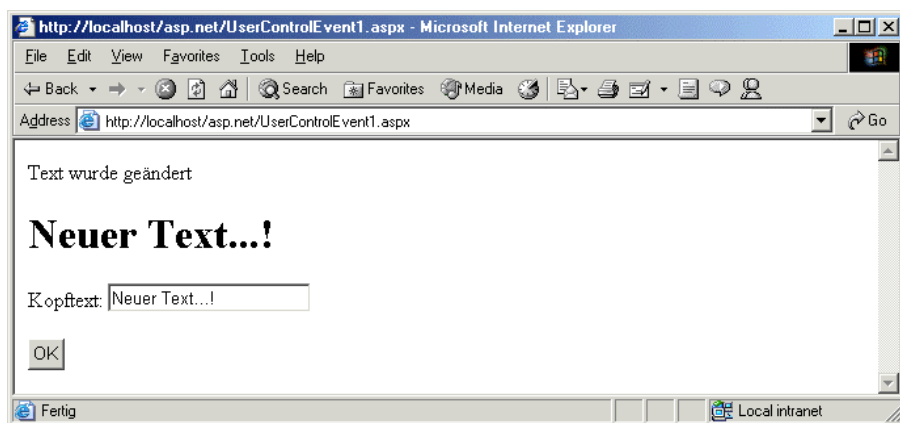


Abbildung 7.9 Die Benutzeränderung resultiert in einem Ereignis.

Sobald der Text geändert und das Ereignis ausgelöst wird, erfolgt die Ausgabe eines kurzen Hinweistextes mittels `Response.Write`. Die Abbildung zeigt dies deutlich.

Ereignisse mit Ereignisargumenten

Ereignisargumente sind dafür da, genutzt zu werden. Sie können über dieses System Daten an das Ereignis weiterleiten. Angewandt auf das gezeigte TextChanged-Ereignis könnten Sie beispielsweise den alten und den neuen Text übergeben. Zudem könnten Sie es der Ereignisbehandlung überlassen, den neuen Text vor der Zuweisung noch zu ändern.

Bitte schrecken Sie nicht vor der Textlänge des folgenden Listings zurück. Die Definition von Ereignisargumenten sieht auf den ersten Blick wirklich etwas erschreckend aus. Da jedoch immer wieder das gleiche Schema zum Einsatz kommt, lässt sich der einmal geschriebene Quellcode wiederverwenden.

Das Listing zeigt zunächst eine geschachtelte Klasse `TextChangedEventArgs`, die von der Basis `EventArgs` abgeleitet ist. Über den Konstruktor können die initiellen Werte der beiden Eigenschaften `OldText` und `NewText` zugewiesen werden. Das erste Mitglied verfügt über einen `get`-Accessor, das zweite zusätzlich über einen `set`-Accessor.

Im Anschluss an die Klasse folgt die Definition eines Delegates mit dem Namen `TextChangedEventHandler`. Hier werden die gezeigten Ereignisargumente verwendet. Es folgt das Ereignis selbst. Dieses wird weiter unten ausgelöst. Dabei wird zunächst eine neue Instanz der Ereignisargumente angelegt. Nach der Behandlung des Ereignisses wird der Wert der Eigenschaft `NewText` dem `TextBox`-Control zugewiesen. Auf diese Weise kann die aufrufende Seite den Inhalt noch modifizieren.

Listing 7.20 UserControlEvent2.ascx

```
<% @Control Language="C#" Debug="true" ClassName="HeaderControl" %>

<script runat="server">

public class TextChangedEventArgs : EventArgs
{
    protected string m_OldText;
    protected string m_NewText;

    public TextChangedEventArgs(string oldText, string newText)
    {
        m_OldText = oldText;
        m_NewText = newText;
    }

    public string OldText
    {
        get { return(m_OldText); }
    }
}
```

```
public string NewText
{
    get { return(m_NewText); }
    set { m_NewText = value; }
}

public delegate void TextChangedEventHandler(object sender,
    TextChangedEventArgs e);
public event TextChangedEventHandler TextChanged;

public string Text
{
    get { return(lbl.Text); }
    set { lbl.Text = value; }
}

void bt_click(object sender, EventArgs e)
{
    if(this.Text != tb.Text)
    {
        TextChangedEventArgs args = new
            TextChangedEventArgs(this.Text, tb.Text);
        OnTextChanged(args);
        this.Text = args.NewText;
    }
}

protected virtual void OnTextChanged(TextChangedEventArgs e)
{
    if(TextChanged != null)
        TextChanged(this, e);
}

</script>

<h1> <ASP:Label id="lbl" runat="server"/> </h1>

<p>
    Kopftext: <ASP:TextBox id="tb" runat="server" text="Hallo Welt!"/>
</p>

<p> <ASP:Button runat="server" text="OK" onclick="bt_click"/> </p>
```

Das zweite Listing zeigt die benötigte ASP.NET-Seite, die das User Control einsetzt. Hier wird das Ereignis behandelt. Dabei wird der alte sowie der neue Text im Browserfenster ausgegeben. Anschließend wird der neue Text noch durch einen Smiley ergänzt.

Listing 7.21 UserControlEvent2.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register Tagprefix="PAL" Tagname="Header"
    Src="UserControlEvent2.ascx" %>

<script runat="server">

void Header_TextChanged(object sender,
    HeaderControl.TextChangedEventArgs e)
{
    Response.Write("<p>Text wurde geändert<br>Alter Text: ");
    Response.Write(e.OldText);
    Response.Write("<br>Neuer Text: ");
    Response.Write(e.NewText);

    e.NewText += " :-))";
}

</script>

<form runat="server">

    <PAL:Header runat="server"
        text="Hallo Welt!"
        OnTextChanged="Header_TextChanged"/>

</form>
```

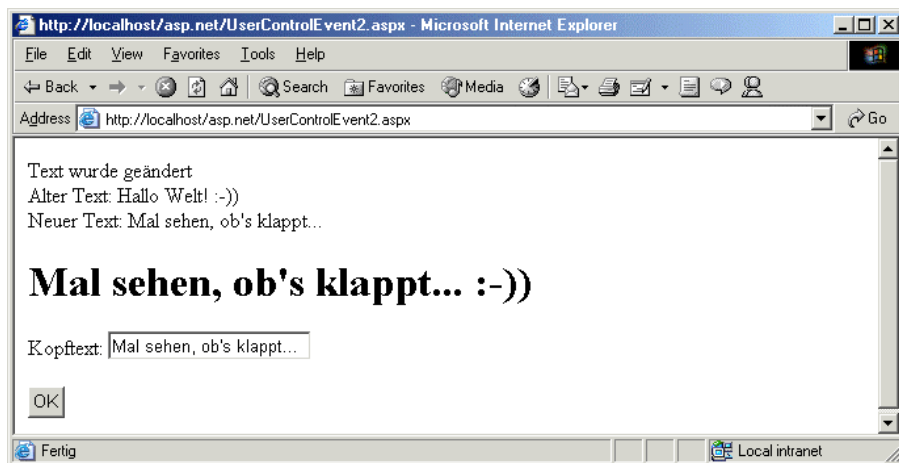


Abbildung 7.10 Die Ereignisargumente liefern die korrekten Daten.

Wie in der Abbildung zu sehen, wird über die Ereignisargumente der alte und neue Text geliefert. Die an diesem vorgenommenen Änderungen werden ganz offensichtlich auch vom User Control übernommen. Auf diese Weise hat der Entwickler den größtmöglichen Einfluss und kann notfalls die Benutzeränderung sogar zurücknehmen:

```
e.NewText = e.OldText;
```



Damit das vorgestellte System korrekt funktioniert, bedarf es eines eher unscheinbaren Tricks. Das Problem liegt in der verschachtelten Definition der Klasse `TextChangedEventArgs`. Damit innerhalb der Seite auf diese zugegriffen werden kann, muss die Verschachtelung berücksichtigt werden. Sie erkennen dies in der Deklaration an der zusätzlichen Angabe der übergeordneten Klasse:

```
void Header_TextChanged(object sender,
    HeaderControl.TextChangedEventArgs e)
...
```

Normalerweise wird der Klassenname des User Controls (wie auch der Seite) dynamisch durch die ASP.NET-Engine vorgegeben. In diesem Fall wäre die Angabe quasi unmöglich. Sie können der Engine jedoch über das `ClassName`-Attribut der `@Control`-Direktive explizit einen Namen vorgeben. Dies wurde bei der Erstellung des User Controls berücksichtigt:

```
<% @Control Language="C#" Debug="true" ClassName="HeaderControl" %>
```


Alternativ können Sie die benötigten Ereignisargumente auch ganz einfach in einer Assembly ablegen. Dazu erstellen Sie eine Quellcode-Datei mit der entsprechenden Klasse, kompilieren diese und legen sie als DLL im bin-Verzeichnis der Web-Applikation ab.

7.6 ... ein komplexes Custom Control erstellen?

Als Alternative zu User Controls können Sie komplett eigene Controls erstellen. Hier haben Sie alles in der Hand und können zudem entscheiden, von welcher Basis sie sich ableiten. Ich habe ein kleines Beispiel erarbeitet, das verschiedene Möglichkeiten aufzeigt. Zwei Eigenschaften verwenden den ViewState-StateBag, ein zusätzliches Ereignis ermöglicht die Interaktion des Entwicklers.

Das mit dem Namen `ScrapeWebpage` versehene Control basiert auf dem Rezept „... Inhalte anderer Seiten scrapen?“ aus dem Kapitel „Content-Management“. Sinn und Zweck des Controls ist die Integration von fremden Website-Inhalten in die eigene Seite. Die zu übernehmende URL wird der gleichnamigen Eigenschaft zugewiesen. Alles andere geschieht automatisch. Optional kann das Cachen der angeforderten Seite deaktiviert werden oder über ein Ereignis in den Datenstrom eingegriffen werden.

Das Control wurde – wie alle Custom Controls – in Form einer Quellcode-Datei erstellt. Ich habe dazu die Visual C# .NET gewählt, Visual Basic .NET oder andere Programmiersprachen sind eine gleichwertige Alternative.

Listing 7.22 ScrapeWebpage.cs

```
using System;
using System.IO;
using System.Net;
using System.Web;
using System.Web.Caching;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace PAL.WebControls
{
    public class ScrapeWebpage : WebControl
    {
        public delegate void ParsePageEventHandler(object sender,
            ParsePageEventArgs e);
        public event ParsePageEventHandler ParsePage;

        protected string _PageContents;
```

```
public string Url
{
    get
    {
        if (ViewState["Url"] == null) return(null);
        return((string) ViewState["Url"]);
    }
    set { ViewState["Url"] = value; }
}

public bool EnableCaching
{
    get
    {
        if (ViewState["EnableCaching"] == null) return(true);
        return((bool) ViewState["EnableCaching"]);
    }
    set { ViewState["EnableCaching"] = value; }
}

protected override void CreateChildControls()
{
    _PageContents = this.LoadFromCache();
    if(_PageContents == null)
    {
        _PageContents = this.Fetch(this.Url);
        ParsePageEventArgs e = new ParsePageEventArgs(_PageContents);
        OnParsePage(e);
        _PageContents = e.PageContents;
        SaveInCache(_PageContents);
    }
}

protected virtual string Fetch(string url)
{
    if(url != null)
    {
        WebRequest request = WebRequest.Create(url);
        WebResponse response = request.GetResponse();
        StreamReader reader =
            new StreamReader(response.GetResponseStream());
        return(reader.ReadToEnd());
    }
    return(null);
}

protected virtual void SaveInCache(string data)
{
    if(this.EnableCaching && data != null)
```

```
{
    Cache cache = HttpContext.Current.Cache;
    cache["scrapewebpage_" + this.Url] = data;
}
}

protected virtual string LoadFromCache()
{
    if(this.EnableCaching)
    {
        Cache cache = HttpContext.Current.Cache;
        object obj = cache["scrapewebpage_" + this.Url];
        return((obj == null ? null : (string) obj));
    }
    return(null);
}

protected virtual void OnParsePage(ParsePageEventArgs e)
{
    if(ParsePage != null)
        ParsePage(this, e);
}

protected override void RenderContents(HtmlTextWriter writer)
{
    writer.Write(_PageContents);
}
}

public class ParsePageEventArgs : EventArgs
{
    private string _PageContents;

    private ParsePageEventArgs() {}

    public ParsePageEventArgs(string pageContents)
    {
        _PageContents = pageContents;
    }

    public string PageContents
    {
        get { return(_PageContents); }
        set { _PageContents = value; }
    }
}
}
```

Im Prinzip ist das Control recht einfach aufgebaut. Die zwei beschriebenen Eigenschaften werden direkt auf das ViewState-StateBag abgebildet.

Die eigentliche Aufgabe des Controls erledigt dieses in der überschriebenen Methode `CreateChildControls`. Hier wird das Control aufgefordert, die Unterobjekte und somit den Inhalt aufzubereiten. Zunächst wird über die Methode `LoadFromCache` abgefragt, ob die abzufragende Seite möglicherweise bereits im Cache vorliegt. Diese Funktionalität wurde ausgegliedert, um bei einer eventuellen Wiederverwendung des Controls eine spezifische Überschreibung dieses Bereichs zu ermöglichen. Das Gleiche gilt für die Methode `Fetch`, die die fremde Seite abruft, sofern diese nicht im Cache vorhanden ist. Eine genaue Beschreibung dieser Technik findet Sie im bereits erwähnten Rezept.

Um dem Entwickler einen Eingriff in die abgefragten Daten zu ermöglichen, wurde ein Ereignis `ParsePage` integriert. Für dieses ist ein Delegate `ParsePageEventHandler`, das Ereignis selbst sowie eine aufrufende Methode `OnParsePage` implementiert. Die verwendeten Ereignisargumente vom Typ `ParsePageEventArgs` erlauben über die Eigenschaft `PageContents` den lesenden und schreibenden Zugriff auf den Inhalt.

Die abgefragten und gegebenenfalls noch modifizierten Daten werden mittels `SaveInCache` im Cache abgelegt, sofern dies nicht über die Eigenschaft `EnableCaching` verhindert wird. Die letzte Ausgabe der Daten an den Browser erfolgt in der überschriebenen Methode `RenderContents`.

Um das Control einzusetzen, müssen Sie es mit Hilfe des Kompilers `csc.exe` kompilieren und die erzeugte DLL im bin-Verzeichnis Ihrer Web-Applikation ablegen:

```
csc.exe /t:library ScrapeWebpage.cs
```

Was nun noch fehlt, ist ein kleines Beispiel, das die Verwendung des Controls demonstriert. Und hier kommt es auch schon ...

Listing 7.23 scrape1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register TagPrefix="PAL" Namespace="PAL.WebControls"
    Assembly="ScrapeWebpage"%>
<% @Import Namespace="System.Text.RegularExpressions" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    scrape.Url = tb.Text;
}

</script>
```

```

<h1>ScrapeWebpage Sample</h1>

<form runat="server">

    <p> Bitte geben Sie die gewünschte Adresse ein:
        <ASP:TextBox id="tb" runat="server" text="http://www.asp-buch.de"/>
        <ASP:Button runat="server" text="ok" onclick="bt_click"/>
    </p>

</form>

<hr/>

<PAL:ScrapeWebpage id="scrape" runat="server"
    EnableCaching="false"/>

<hr/>

```

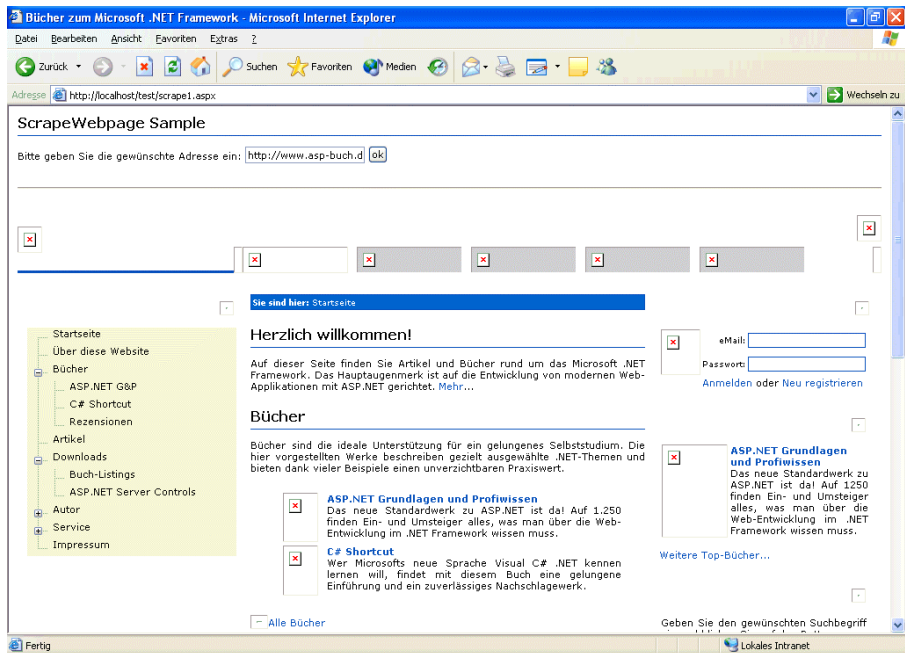


Abbildung 7.11 Die Seite wurde übernommen, die Bilder fehlen jedoch.

Die Seite ist recht einfach gehalten. Sie enthält ein Eingabeformular mit TextBox- und Button-Control. Ein Klick auf den Button weist die eingegebene URL der gleichnamigen Eigenschaft des ScrapeWebpage-Controls zu. Die Inhalte werden

dabei an genau der Stelle dargestellt, an der das Control im Layout der Seite platziert wurde. Dies ergibt sich durch die Tatsache, dass die Methode `RenderContents` überschrieben wurde, die automatisch zur richtigen Zeit aufgerufen wird.

Die Abbildung zeigt das Ergebnis im Browserfenster. Aufgrund der relativen Links fehlen die Bilder. Dies kann durch eine Behandlung des Ereignisses `ParsePage` behoben werden. Hier werden alle Grafiken mit absoluten Pfaden versehen.

Listing 7.24 `scrape2.aspx`

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    scrape.Url = tb.Text;
}

void scrape_ParsePage(object sender, ParsePageEventArgs e)
{
    if(e.PageContents != null)
    {
        Regex regex = new Regex(@"\<img.+src=\"");
        e.PageContents = regex.Replace(e.PageContents,
            string.Format("$0{0}/", tb.Text));
    }
}

</script>

<h1>ScrapeWebpage Sample</h1>

<form runat="server">

    <p> Bitte geben Sie die gewünschte Adresse ein:
        <ASP:TextBox id="tb" runat="server" text="http://www.asp-buch.de"/>
        <ASP:Button runat="server" text="ok" onclick="bt_click"/>
    </p>

</form>

<hr/>

<PAL:ScrapeWebpage id="scrape" runat="server"
    EnableCaching="false"
    OnParsePage="scrape_ParsePage"/>

<hr/>
```

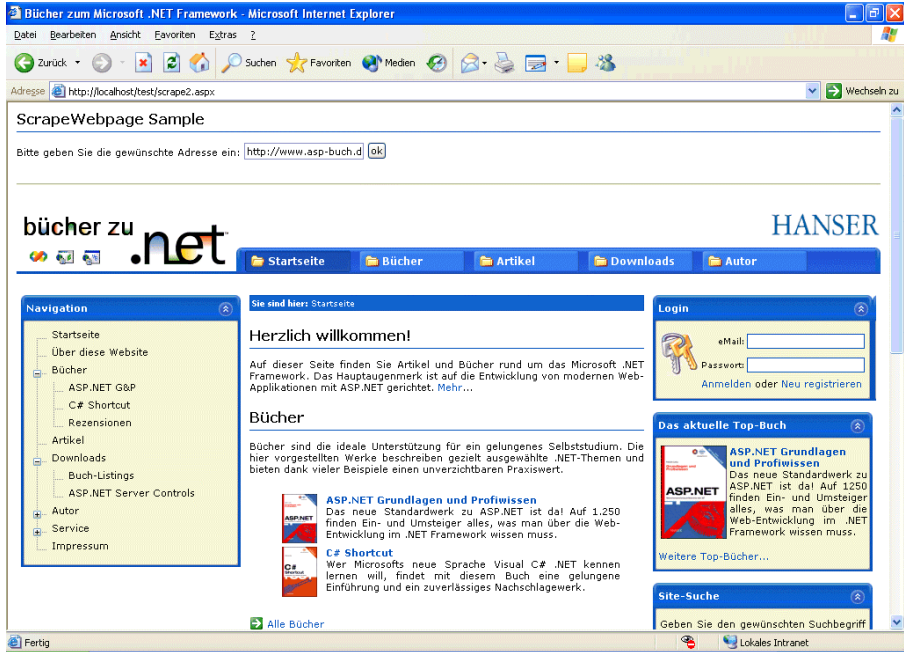


Abbildung 7.12 Nun werden auch die Bilder korrekt angezeigt.

7.7 ... ein User Control dynamisch platzieren?

In aller Regel werden User Controls statisch innerhalb einer Seite hinterlegt. Eine schöne Möglichkeit von Server Controls im Allgemeinen ist die dynamische Anlage und Platzierung über die Controls-Collection. Lässt sich diese Möglichkeit auch mit User Controls nutzen? Natürlich, allerdings ist das ein klein wenig trickreich.

Zunächst möchte ich Ihnen das einzubindende User Control zeigen. Dieses enthält ein Label-Control zur Anzeige einer h1-Überschrift. Der Inhalt dieser kann über die öffentliche Eigenschaft `Text` festgelegt werden. Das Control sieht auf den ersten Blick harmlos aus, enthält aber bereits einen Trick. Über das `ClassName`-Attribut der `@Control`-Direktive wurde explizit ein Klassenname festgelegt. Dieser wird ansonsten dynamisch durch die ASP.NET-Engine vergeben.

Listing 7.25 Header1.ascx

```
<% @Control Language="C#" Debug="true" ClassName="Header" %>

<script runat="server">

public string Text
{
    get { return(lb.Text); }
    set { lb.Text = value; }
}

</script>

<h1> <ASP:Label id="lb" runat="server"/> </h1>
```

Der zweite Trick ist die Referenzierung des Controls in der einbindenden Seite. Der Dateiname wird über die `@Reference`-Direktive angegeben. Diese stellt sicher, dass das Control zusammen mit der Seite kompiliert wird. Auf diese Weise steht das Control über den weiter oben angegebenen Klassennamen zur Verfügung und kann wie gewohnt genutzt werden.

Listing 7.26 UserControl1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Reference Control="Header1.ascx" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Header header = new Header();
    header.Text = "Hallo Welt";
    this.Controls.Add(header);
}

</script>
```

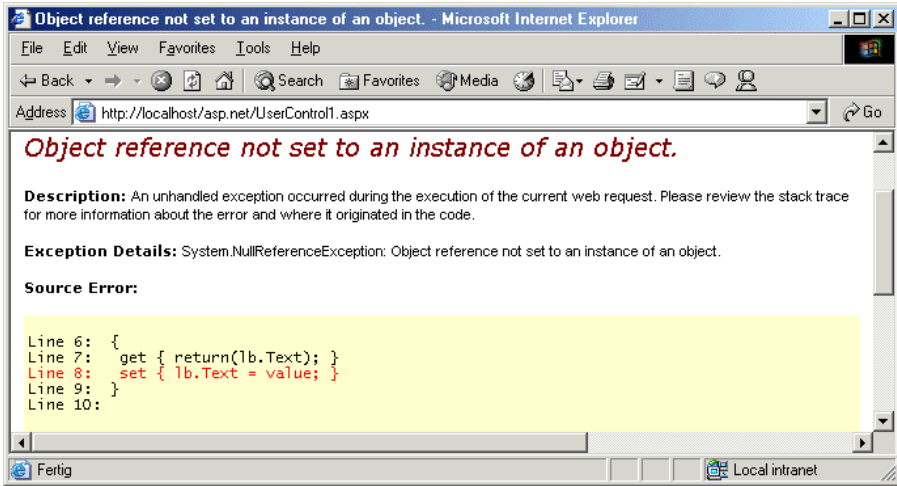



Abbildung 7.13 Das User Control lässt sich nicht korrekt verwenden.

So viel zur Theorie, doch funktioniert dieser Ansatz auch in der Praxis? Noch nicht ganz. Die Abbildung lässt erkennen, dass das Label-Control innerhalb des User Controls offensichtlich nicht instanziiert wurde. Aus diesem Grund kann die Eigenschaft `Text` nicht weitergereicht werden, und es kommt zu einer Ausnahme.

Der Grund für das Problem liegt in der Architektur von User Controls. Diese müssen an eine Seite gebunden werden. Um eine deklarative Einbindung, wie in diesem Rezept vorgestellt, möglich zu machen, haben die Entwickler der Basisklasse `UserControl` eine zusätzliche Methode `InitializeAsUserControl` spendiert – Trick Nummer 3. Dieser Methode sollte direkt im Anschluss an die Instanziierung der Klasse die aktuelle Seite übergeben werden. Das geänderte Listing zeigt die Verwendung.

Listing 7.27 `UserControl2.aspx`

```
<% @Page Language="C#" Debug="true" %>
<% @Reference Control="Header1.ascx" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Header header = new Header();
    header.InitializeAsUserControl(this);
    header.Text = "Hallo Welt";
    this.Controls.Add(header);
}

</script>
```

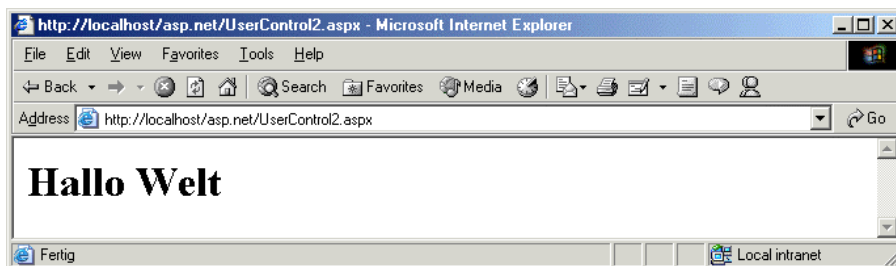


Abbildung 7.14 Die Überschrift wird korrekt ausgegeben.

Die Abbildung zeigt, dass das Control nun auch deklarativ korrekt verwendet werden kann. Der Aufruf der Methode `InitializeAsUserControl` ist jedoch obligatorisch, damit das Control korrekt initialisiert wird.

7.8 ... ein User Control dynamisch laden?

Meistens sind zu benutzende User Controls bekannt und werden statisch platziert. Eine Möglichkeit, die Objekte dynamisch zu instanziiieren, haben Sie im Rezept „... ein User Control dynamisch platzieren?“ erfahren. Insbesondere bei modular aufgebauten Portalen sind die einzelnen zu verwendenden Module vorher nicht bekannt. Der Dateiname könnte beispielsweise in einer Datenbank hinterlegt sein.

Statisches Control

In einem derartigen Fall können Sie das User Control dynamisch einladen. Die Klasse `Page` bietet hierzu eine Methode `LoadControl` an, der der Dateiname übergeben wird und die das instanziierte Control vom gleichnamigen Datentyp zurückliefert.

Das erste Listing zeigt das einzubindende User Control. Es beinhaltet ausschließlich statischen Text in Form einer Überschriftenzeile. Die im zweiten Listing gezeigte Seite lädt das Control dynamisch mit Hilfe der `LoadControl`-Methode ein und fügt es der `Controls-Collection` an.

Listing 7.28 StaticHeader1.ascx

```
<% @Control Language="C#" Debug="true" %>

<h1> Dies ist eine statische Überschrift </h1>
```

Listing 7.29 UserControl1.aspx

```
<% @Page Language="C#" Debug="true" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Control ctl = this.LoadControl("StaticHeader1.ascx");
    this.Controls.Add(ctl);
}

</script>
```

**Abbildung 7.15** Das Control wurde dynamisch eingeladen.**Dynamisches Control per Reflection**

Die Abbildung zeigt, dass das Control samt dem enthaltenen statischen Text korrekt eingeladen wurde. Die Verwendung des Datentyps `Control` zeigt aber ein Problem dieses Ansatzes auf. Eventuelle Eigenschaften des User Controls lassen sich nicht ohne weiteres setzen, da kein passender Datentyp zur Verfügung steht. Um dennoch darauf zuzugreifen, können Sie Reflection verwenden. Die beiden nachfolgenden Listings demonstrieren dies anhand eines dynamischen Überschriftenobjekts. Der Text wird über die gleichnamige Eigenschaft zugewiesen.

Listing 7.30 Header1.ascx

```
<% @Control Language="C#" Debug="true" ClassName="Header" %>

<script runat="server">

public string Text
{
    get { return(lb.Text); }
    set { lb.Text = value; }
}

</script>

<h1> <ASP:Label id="lb" runat="server"/> </h1>
```

Listing 7.31 UserControl2.aspx

```
<% @Import Namespace="System.Reflection" %>

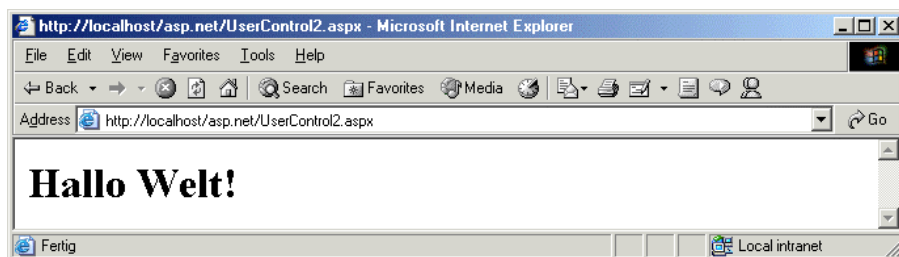
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Control ctl = this.LoadControl("Header1.ascx");

    Type t = ctl.GetType();
    PropertyInfo p = t.GetProperty("Text");
    p.SetValue(ctl, "Hallo Welt!", null);

    this.Controls.Add(ctl);
}

</script>
```

**Abbildung 7.16** Die Eigenschaft wurde per Reflection zugewiesen.

Dynamisches Control per Referenz

Der Einsatz von Reflection ist ein wenig umständlich und nicht jedermanns Sache. Eine andere Möglichkeit ist die explizite Angabe eines Klassennamens für das User Control. Dies geschieht über das `ClassName`-Attribut der `@Control`-Direktive und ist im Listing weiter oben bereits zu sehen. Sie referenziert den Datentyp über die `@Reference`-Direktive und kann daher auf die Eigenschaften des Controls zugreifen.

Listing 7.32 UserControl3.aspx

```
<% @Reference Control="Header1.ascx" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Header ctl = (Header) this.LoadControl("Header1.ascx");
    ctl.Text = "Hallo Welt!";
    this.Controls.Add(ctl);
}

</script>
```

Dynamisches Control per Code Behind

Als Beispielanwendung für das dynamische Einladen von Controls habe ich eine modulare Portalseite genannt. Gerade bei diesem Einsatz ist zu erwarten, dass die einzelnen Controls immer wieder über ähnliche oder gleiche Eigenschaften verfügen. Denkbar wäre daher, eine zentrale Code Behind-Klasse zu erstellen, von der sich alle User Controls ableiten. Die Zuweisung der Eigenschaften erfolgt nur noch über diesen Datentyp, und es muss keine direkte Referenz zwischen Seite und User Control existieren.

Die nachfolgenden Listings demonstrieren dieses Vorgehen anhand mehrerer Überschriften-Controls. Zur Demonstration der Technik unterscheiden sich diese im verwendeten HTML-Tag. Kommt im ersten `h1` zum Einsatz, wird im zweiten eine kleinere Überschrift mittels `h2` ausgegeben. Zunächst sehen Sie jedoch die identische Code Behind-Klasse.

Listing 7.33 CommonHeader.cs

```
using System;
using System.Web;
using System.Web.UI;

public class CommonHeader : UserControl
{
    public string Text
    {
        get { return((string) ViewState["Text"]); }
        set { ViewState["Text"] = value; }
    }
}
```

Die beiden nachfolgenden User Controls leiten sich von der Klasse `CommonHeader` ab. Sie geben den dort zugewiesenen Text über die Inline-Anweisung innerhalb eines `h1`- beziehungsweise `h2`-HTML-Tags aus.

Listing 7.34 h1.ascx

```
<% @Control Language="C#" Debug="true" Inherits="CommonHeader" %>

<h1> <%= this.Text %> </h1>
```

Listing 7.35 h2.ascx

```
<% @Control Language="C#" Debug="true" Inherits="CommonHeader" %>

<h2> <%= this.Text %> </h2>
```

Der Einsatz der beiden Controls ist im letzten Listing zu sehen. Es kommt wie auch zuvor die Methode `LoadControl` zum Einsatz. Als Datentyp ist die Klasse `CommonHeader` angegeben, die sowohl dem einen als auch dem anderen Control als Basis dient.

Listing 7.36 UserControl4.aspx

```
<% @Page Language="C#" Debug="true" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    CommonHeader h1 = (CommonHeader) this.LoadControl("h1.ascx");
    h1.Text = "Überschrift erster Ordnung";
    this.Controls.Add(h1);
}
```

```
CommonHeader h2 = (CommonHeader) this.LoadControl("h2.ascx");  
h2.Text = "Überschrift zweiter Ordnung";  
this.Controls.Add(h2);  
}  
  
</script>
```



Abbildung 7.17 Die Zuweisung erfolgt über die gemeinsame Basisklasse.

Dynamisches Control per Schnittstelle

Alternativ zu der Verwendung von Code Behind besteht die Möglichkeit, eine zentral definierte Schnittstelle zu unterstützen. Diese gibt vor, welche Eigenschaften implementiert werden müssen, und wird als Datentyp zum Zugriff auf das Control angegeben.

Auch diesen Ansatz möchte ich Ihnen anhand des vorherigen Beispiels demonstrieren. Die Code Behind-Datei enthält nun nur noch die Schnittstelle `IHeader` mit der Definition der zu unterstützenden Mitglieder.

Listing 7.37 IHeader.cs

```
public interface IHeader  
{  
    string Text  
    {  
        get;  
        set;  
    }  
}
```

Die Schnittstelle wird von den beiden User Controls unterstützt, indem die entsprechenden Mitglieder implementiert werden und der Name der Schnittstelle über die `@Implements`-Direktive zugewiesen wird.

Listing 7.38 `IHeader_h1.ascx`

```
<% @Control Language="C#" Debug="true" %>
<%@ Implements Interface="IHeader" %>

<script runat="server">

public string Text
{
    get { return((string) ViewState["Text"]); }
    set { ViewState["Text"] = value; }
}

</script>

<h1> <%= this.Text %> </h1>
```

Listing 7.39 `IHeader_h2.ascx^`

```
<% @Control Language="C#" Debug="true" %>
<%@ Implements Interface="IHeader" %>

<script runat="server">

public string Text
{
    get { return((string) ViewState["Text"]); }
    set { ViewState["Text"] = value; }
}

</script>

<h2> <%= this.Text %> </h2>
```

Der Zugriff auf die Eigenschaft `Text` erfolgt nun über die von beiden Controls unterstützte Schnittstelle `IHeader`. Bevor das Objekt allerdings an die Controls-Collection angefügt werden kann, muss eine explizite Typenkonvertierung stattfinden.

Listing 7.40 UserControl5.aspx

```
<% @Page Language="C#" Debug="true" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    IHeader h1 = (IHeader) this.LoadControl("IHeader_h1.ascx");
    h1.Text = "Überschrift erster Ordnung";
    this.Controls.Add((Control) h1);

    IHeader h2 = (IHeader) this.LoadControl("IHeader_h2.ascx");
    h2.Text = "Überschrift zweiter Ordnung";
    this.Controls.Add((Control) h2);
}

</script>
```

Code Behind versus Schnittstelle

Die Verwendung einer zentralen Code Behind-Klasse unterscheidet sich im Prinzip nicht wesentlich von der Vorgabe einer Schnittstelle. Letztere Variante erfordert die manuelle Implementierung der einzelnen Mitglieder innerhalb jedes Controls. Der Vorteil ist die mögliche Unterstützung mehrerer Schnittstellen durch mehrfache Verwendung der `@Implements`-Direktive. Für welchen Weg Sie sich entscheiden sollten, hängt primär vom Einsatzgebiet ab.

7.9 ... ein Template dynamisch laden?

Datengebundene Controls wie `Repeater`, `DataList` und `DataGrid` erlauben das Hinterlegen von Vorlagen. Bei der Bindung der Controls werden diese Vorlagen für jedes verfügbare Datenelement ausgewertet und ausgegeben.

Template aus Datei laden

In aller Regel sind die Vorlagen innerhalb des Layout-Bereichs statisch hinterlegt. Bei der ausschließlichen Verwendung von Code Behind oder der dynamischen Anlage der Controls ist dies nicht möglich. In diesem Fall kann die Vorlage aus einer externen Datei eingeladen werden. Die Methode `LoadTemplate` der Klasse `Page` übernimmt diese Aufgabe.

Der Methode muss der Dateiname einer Vorlage übergeben werden. Die Endung muss aus zunächst unerklärlichen Gründen `ascx` lauten, obwohl diese augenscheinliche Funktionalität mit User Controls an sich eigentlich nichts zu tun. Intern verwendet die ASP.NET-Engine jedoch ein User Control.

Nachfolgend sehen Sie die Einbindung einer Vorlage und deren Zuweisung an ein `DataList`-Control. Als Datenquelle dient die Autorentabelle der Beispieldatenbank.

Listing 7.41 Template1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    ITemplate template = this.LoadTemplate("template.ascx");
    dl.ItemTemplate = template;

    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dl.DataSource = cmd.ExecuteReader();
    DataBind();

    conn.Close();
}

</script>

<ASP:DataList id="dl" runat="server"/>
```

Die eingeladene Vorlage in der Datei `template.ascx` enthält die Vorlage, die für jedes Datenelement individuell ausgewertet wird. Im Unterschied zu der regulären Notation innerhalb eines datengebundenen Controls steht die Eigenschaft `Container.DataItem` nicht direkt zur Verfügung. Die Abfrage der Datenquelle sieht daher minimal anders aus.

Listing 7.42 template.ascx

```
&bull;
<%# DataBinder.Eval(Container, "DataItem['Firstname']") %>
<%# DataBinder.Eval(Container, "DataItem['Lastname']") %>
```

Im Ergebnis unterscheidet sich die Einbindung auf diese Weise jedoch nicht von der regulären Notation innerhalb des Controls. Die Abbildung beweist dies.

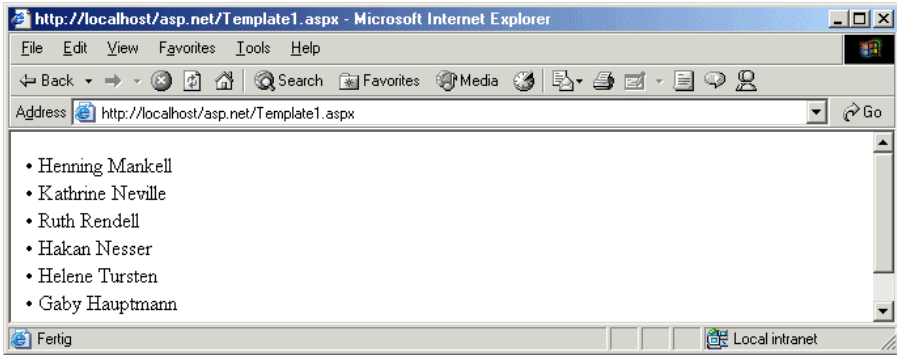


Abbildung 7.18 Die Vorlage wurde aus einer Datei eingelesen.

Template programmatisch anlegen

Mal anders gefragt: Was ist eine Vorlage eigentlich? Es handelt sich um ein Objekt, das die Schnittstelle `ITemplate` unterstützt, mehr nicht. Die Schnittstelle implementiert eine einzige Methode `InstantiateIn`, der ein `Control` übergeben wird. Das Template wird aufgefordert, diesem Container sämtliche Objekte zu übergeben. Es spricht also nichts dagegen, eine Vorlage programmatisch anzulegen.

Das Beispiel zeigt die Verwendung in Form einer Klasse, die die Schnittstelle `ITemplate` unterstützt. Innerhalb der Methode `InstantiateIn` wird ein Literal-Control angelegt und der Controls-Collection des übergebenen Containers angefügt. Um an den datengebundenen Inhalt zu gelangen, wird innerhalb eine Ereignisbehandlung für das eingefügte Literal-Control implementiert. Über die Eigenschaft `NamingContainer` kann darüber der Container und somit das entsprechende Daten-element abgefragt werden. Das Ergebnis entspricht optisch dem vorherigen.

Listing 7.43 Template2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Template template = new Template();
    dl.ItemTemplate = template;

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors;";
```

```
OleDbCommand cmd = new OleDbCommand(SQL, conn);
dl.DataSource = cmd.ExecuteReader();
DataBind();

conn.Close();
}

public class Template : ITemplate
{
    public void InstantiateIn(Control container)
    {
        DataListItem dli = (DataListItem) container;

        Literal literal = new Literal();
        literal.DataBinding += new EventHandler(this.literal_DataBinding);
        container.Controls.Add(literal);
    }

    protected void literal_DataBinding(object sender, EventArgs e)
    {
        Literal literal = (Literal) sender;
        DataListItem dli = (DataListItem) literal.NamingContainer;

        literal.Text += "&bull; ";
        literal.Text += DataBinder.Eval(dli.DataItem, "[Firstname]");
        literal.Text += " ";
        literal.Text += DataBinder.Eval(dli.DataItem, "[Lastname]");
    }
}

</script>

<ASP:DataList id="dl" runat="server"/>
```

7.10 ... ein Control aus einer Zeichenkette parsen?

Es ist kaum bekannt, aber es ist durchaus möglich, ein Control auf Basis einer Zeichenkette on the fly zu parsen und auf der Seite einzusetzen. Die Klasse `Page` bietet hierzu eine Methode `ParseControl` an, der die Zeichenkette übergeben wird. Das Ergebnis ist ein Control. Sie glauben das nicht? Ich beweise es Ihnen.

Listing 7.44 ParseControl1.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Control ctl = this.ParseControl("<ASP:Label runat='server'
    ForeColor='Red' Font-Bold='true' Text='Hallo Welt!'/>");
    this.Controls.Add(ctl);
}

</script>

```

Im Listing ist die Verwendung der Methode `ParseControl` zu sehen. Als Zeichenkette wird die Deklaration eines serverseitigen Label-Controls übergeben. Neben der Zuweisung des anzuzeigenden Textes wird auch eine Reihe weiterer Einstellungen vorgenommen. Die Abbildung zeigt, dass diese Angaben tatsächlich interpretiert und in ein Server Control umgewandelt wurden.

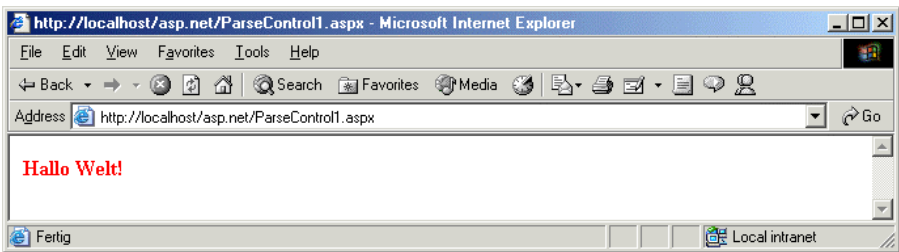


Abbildung 7.19 Das Label wurde on the fly angelegt.

Da als Rückgabewert der Methode `Control` und kein Array angegeben ist, könnte man auf die Idee kommen, dass nur ein Objekt pro Aufruf geparkt werden könnte. Das ist jedoch nicht ganz richtig, denn dieses Control kann durchaus auch Unterobjekte beinhalten, die über die Controls-Collection angeboten werden. Das zweite Beispiel zeigt ein serverseitiges Formular, in dem sich ein TextBox- sowie ein Button-Control befinden.

Listing 7.45 ParseControl2.aspx

```

void Page_Load(object sender, EventArgs e)
{
    Control ctl = this.ParseControl("<form runat='server'><ASP:TextBox
    id='tb' runat='server' /><ASP:Button id='bt' runat='server'
    Text='OK' /></form>");
    this.Controls.Add(ctl);
}

```

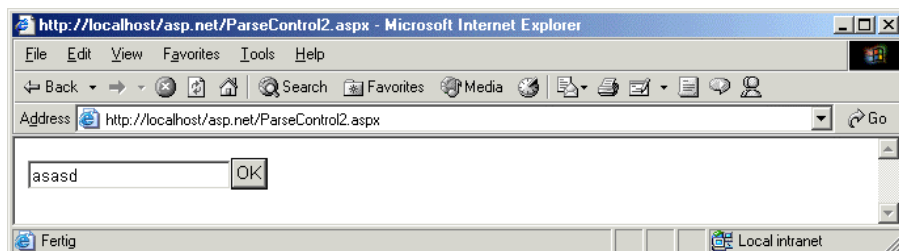


Abbildung 7.20 Auch mehrere verschachtelte Controls lassen sich parsen.

Eine derartige Möglichkeit ist insbesondere zum dynamischen Aufbau von Seiten interessant. Die darzustellenden Objekte können beispielsweise einer Datenbank entnommen werden. Dabei anzumerken ist der Performance-Verlust, der durch das Interpretieren der Objekte zur Laufzeit entsteht. Zudem sind nicht alle Möglichkeiten einer regulären Seite vorhanden. Beispielsweise wird die Behandlung von Ereignissen durch die dynamische Anlage sehr erschwert. Ist innerhalb der Seite Zugriff auf ein Control notwendig, muss dieses zunächst per `FindControl` gesucht werden:

```
TextBox tb = (TextBox) this.FindControl("tb");
```

Was aber, wenn Sie das dynamisch erzeugte Control durch die angebotenen Eigenschaften direkt verändern wollen? Auch das ist möglich, allerdings müssen Sie dazu erst einmal wissen, dass um das eigentliche Control ein weiteres gelegt wird. Es handelt sich um eine quasi unsichtbare Instanz der gleichnamigen Klasse. Dies hat den Zweck, dass Sie innerhalb der Zeichenkette direkt mehrere Controls notieren können. Das folgende Beispiel zeigt das Problem.

Listing 7.46 ParseControl3.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Control ctl = this.ParseControl("<ASP:Label runat='server'
    ForeColor='Red' Font-Bold='true' Text='Hallo Welt!'/>");
    Label lb = (Label) ctl;
    lb.Text = "Hallo Universum!";
    this.Controls.Add(ctl);
}

</script>
```

Man sollte annehmen, dass nun der geänderte Text in einem Label ausgegeben wird. Mitnichten, denn durch das beschriebene zusätzliche Control ist die Typenkonvertierung gar nicht erst möglich, und Sie erhalten eine `InvalidCastException`. Des Problems Lösung ist die Controls-Collection. Sie müssen lediglich das erste Unterobjekt abfragen und erhalten das gewünschte Label. Auf einen zweiten Versuch ...

Listing 7.47 ParseControl4.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Control ctl = this.ParseControl("<ASP:Label runat='server'
    ForeColor='Red' Font-Bold='true' Text='Hallo Welt!'/>");
    Label lb = (Label) ctl.Controls[0];
    lb.Text = "Hallo Universum!";
    this.Controls.Add(ctl);
}

</script>
```

Die Abbildung zeigt, dass der zweite Versuch tatsächlich von Erfolg gekrönt ist. Die Struktur lässt sich übrigens recht einfach nachvollziehen, wenn der Trace-Modus eingeschaltet ist. Informationen dazu finden Sie auch im Rezept „... Debugging und Tracing nutzen?“ im Kapitel „Basics“. Nur am Rande als kleine Gedankenankregung: Haben Sie schon einmal eine ganze HTML-Tabelle parsen müssen? Was halten Sie davon, diese Aufgabe zukünftig ASP.NET zu überlassen und über das Objektmodell der Klasse `HtmlTable` auf die Tabelle zuzugreifen? Weitere Informationen und ein Beispiel finden Sie im Rezept „... eine Tabelle parsen?“ im Kapitel „Content-Management“.

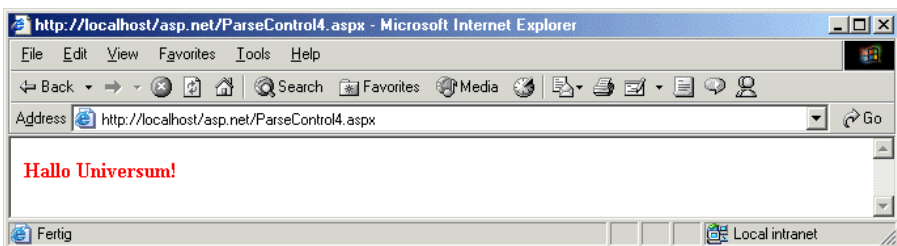


Abbildung 7.21 Auch der direkte Zugriff auf das Control ist möglich.

7.11 ... ein User Control direkt im Browser anzeigen?

Wenn Sie versuchen, ein User Control direkt im Browser aufzurufen, erhalten Sie die in der Abbildung gezeigte Fehlermeldung.

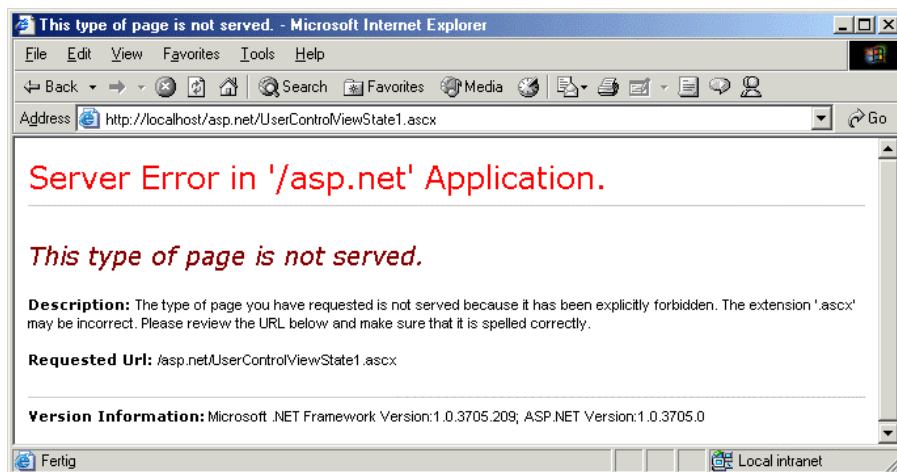
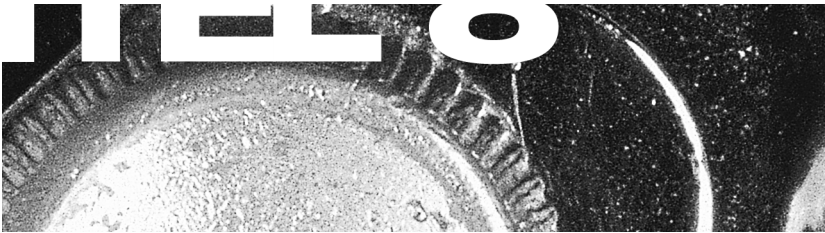


Abbildung 7.22 Ein User Control kann nicht direkt im Browser angezeigt werden.

Die Ursache für den Fehler ist klar, denn ein User Control kann in keinem Fall direkt angezeigt werden. Das Control kann ausschließlich als Bestandteil einer ASP.NET-Seite genutzt werden. Hierzu wird dieses über die `@Register`-Direktive registriert und mit einem Tag-Präfix sowie einem -Namen versehen. Anschließend kann das Control wie andere Web Controls im Layout-Bereich der Seite eingebunden werden.

8

Datenbanken



Wie kann ich ...

8 Datenbanken

Keine dynamische Website kommt ohne zumindest eine Datenbank im Hintergrund aus. Mit ASP.NET und ADO.NET ist die Einbindung eigentlich kein Problem. Doch da gibt es ein paar Fragen, auf die Sie schon immer eine Antwort wissen wollten. Hier finden Sie sie!

Einbindung

Die Klassen zum Datenbankzugriff werden bei .NET unterhalb des Namespaces `System.Data` angesammelt. Bevor Sie die Klassen aus den folgenden Rezepten nutzen können, müssen Sie in aller Regel diesen Namespace zunächst einbinden:

```
<% @Import Namespace="System.Data" %>
```

Sofern Sie auf die Datenbanken per OLE DB zugreifen, müssen Sie zusätzlich folgenden Namespace einbinden:

```
<% @Import Namespace="System.Data.OleDb" %>
```

Greifen Sie hingegen auf einen SQL-Server zu, verwenden Sie folgende Zeile:

```
<% @Import Namespace="System.Data.SqlClient" %>
```

8.1 ... ein strongly typed DataSet anlegen?

Ein strongly typed DataSet ist eine tolle Sache, aber was ist das eigentlich genau? Ein offensichtlicher Nachteil bisheriger Datenbanksysteme ist der typenunsichere Zugriff. Bei ADO und auch beim neuen ADO.NET notieren Sie beispielsweise die Tabellen und Spalten, auf die Sie zugreifen wollen, als Zeichenkette:

```
DataTable = dataset.Tables["Authors"];
```

Ob die Tabelle tatsächlich unter diesem Namen existiert oder nicht, erfahren Sie erst zur Laufzeit. Tippfehler können auf diese Weise unter Umständen selbst bei einer produktiven Website zunächst unerkannt bleiben.

Dieses Problem zu lösen, ist die Aufgabe von strongly typed DataSets. Zusammengefasst wird lediglich eine Ableitung der Klasse DataSet vorgenommen, die die Tabellen und darunter Zeilen und Spalten als typensichere Eigenschaften implementiert. Sie greifen anschließend nicht mehr über einen Zeichenketten-Indexer, sondern über korrespondierende Eigenschaften auf die einzelnen Daten zu. Tippfehler sind dadurch ausgeschlossen.

Anlage eines strongly typed DataSets

Zur Erstellung eines strongly typed DataSet-Objekts sind mehrere Schritte notwendig. Zunächst muss die Struktur der später aufzunehmenden Daten ausgelesen und abgespeichert werden. Als Format kommt die XML Schema Definition Language (XSD) zum Einsatz.

Die Erstellung einer XSD-Datei kann die Methode WriteXmlSchema der DataSet-Klasse übernehmen. Die Methode schreibt sämtliche im DataSet vorhandenen Meta-Daten in die übergebene Datei. Das folgende Listing zeigt, wie es geht.

Listing 8.1 XmlSchema1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter();
    adapter.SelectCommand = cmd;

    DataSet dataset = new DataSet("AuthorsDataSet");
    adapter.FillSchema(dataset, SchemaType.Mapped, "Authors");
    dataset.WriteXmlSchema("c:\\authors.xsd");

    conn.Close();
}

</script>
```

Im Listing werden die Schemainformationen der Tabelle `Authors` der Datenbank `books` ausgelesen und in ein `DataSet` geschrieben. Dem `DataSet` wurde über den Konstruktor ein Name zugewiesen. Das Gleiche gilt für die Tabelle. Beides ist unbedingt erforderlich, damit man die jeweiligen Elemente später ansprechen kann.

Die erzeugte XSD-Datei muss nun in eine Klasse, genauer eine Ableitung der Klasse `DataSet`, umgewandelt werden. Das .NET Framework bietet hierzu das Kommandozeilenprogramm `xsd.exe` an. Der Aufruf zur Erstellung einer C#-Klasse sieht wie folgt aus:

```
xsd.exe /d /l:C# authors.xsd
```

Das Ergebnis ist eine Datei mit dem Namen `authors.cs`. Diese kann entweder kompiliert und im `bin`-Verzeichnis abgelegt oder über die `@Assembly`-Direktive direkt in eine Seite eingebunden werden. Damit sind die Vorbereitungen abgeschlossen.

Verwendung eines strongly typed DataSets

Das folgende Beispiel zeigt die Verwendung des soeben erstellten strongly typed DataSets. Auf den ersten Blick sieht alles ganz normal aus, doch auf den zweiten erkennen Sie sicher die Raffinesse der neuen Technik.

Listing 8.2 TypesDataSet1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter();
    adapter.SelectCommand = cmd;

    AuthorsDataSet dataset = new AuthorsDataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    AuthorsDataSet.AuthorsDataTable authors = dataset.Authors;
```

```
foreach(AuthorsDataSet.AuthorsRow row in authors)
{
    Response.Write(string.Format("{0} {1}<br>",
        row.Firstname, row.Lastname));
}
}
</script>
```

Anstatt eine Instanz der Klasse `DataSet` zu erstellen, kommt die abgeleitete Variante `AuthorsDataSet` zum Einsatz. Der Name entspricht der oben im Konstruktor der Klasse `DataSet` zugewiesenen Zeichenkette.

Das abgeleitete `DataSet` verfügt über eine Eigenschaft `Authors`. Diese liefert eine Instanz der verschachtelten Klasse `AuthorsDataSet.AuthorsDataTable`. Diese verfügt wiederum über einen Enumerator, mit Hilfe dessen sich die einzelnen Zeilen vom Typ `AuthorsDataSet.AuthorsRow` durchlaufen lassen. Jede Zeile verfügt über die Eigenschaften `Firstname` und `Lastname`, über die sich der Inhalt der jeweiligen Zeile abfragen lässt. That's it!

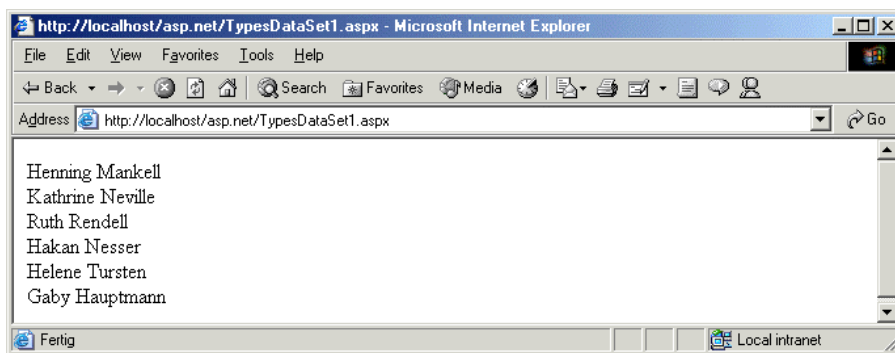


Abbildung 8.1 Die ausgelesenen Daten

Fazit

Mit geringem Aufwand können Sie manuell ein strongly typed `DataSet` anlegen. Tippfehler werden dank Typensicherheit bereits beim Kompilieren erkannt. Ein wenig komfortabler geht es übrigens noch mit der Visual Studio .NET Entwicklungsumgebung. Hier werden die verschiedenen Schritte von der Entwicklungsumgebung übernommen.

8.2 ... auf eine System-DSN zugreifen?

Über den ODBC-Administrationsdialog in der Systemsteuerung können Sie so genannte System-DSN anlegen. Dabei wird für eine ausgewählte Datenbank unter einem Aliasnamen eine Konfiguration vorgehalten. Beispielsweise ist hier auch hinterlegt, mit welchem ODBC-Treiber die Datenbank angesprochen werden soll.

Dieses sehr einfache System wird von vielen ASP-Providern angeboten. Über ein Web-Interface lässt sich eine feste Anzahl von unterschiedlichen System-DSN konfigurieren. Der Zugriff auf die Datenbanken erfolgt dann ausschließlich über den Aliasnamen, nicht aber auf Dateiebene. Auf diese Weise kann die Datenbank außerhalb des im Internet freigegebenen Verzeichnisses liegen, und ein direkter Zugriff von außen per Download ist ausgeschlossen.

Bei den bisherigen ASP-Versionen 2.0 und 3.0 konnte eine Connection sehr einfach auf Basis einer System-DSN aufgebaut werden. Dies sah beispielsweise so aus:

```
Dim conn
Set conn = CreateObject("ADODB.Connection")
conn.Open "<aliasname>"
```

Bei ASP.NET funktioniert dieses System nicht mehr. Versucht man, der Klasse `OleDbConnection` im Konstruktor den Aliasnamen zu übergeben, kommt es zu einem Laufzeitfehler.

Listing 8.3 dsn1.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

public void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new OleDbConnection("test");
    // ...
}

</script>
```

In diesem Fall ist der Aufbau des übergebenen Connection-Strings nicht korrekt. Dieser muss bei OLE DB dem Format `Paramter1=Wert1; Parameter2=Wert2` genügen. Doch auch wenn man das Format ändert und beispielsweise `DSN=test` angibt, ändert sich das Ergebnis nicht merklich. Weiteres Ausprobieren ist zwecklos, es geht nicht ...

Die Ursache für die scheinbare Fehlfunktion ist relativ einfach zu beschreiben. Eine System-DSN basiert auf einem ODBC-Treiber. OLE DB verwendet – wie der Name ja schon unmissverständlich nahe legt – die neueren OLE DB-Treiber. Ein Zugriff auf eine ODBC-Connection mittels OLE DB ist schlichtweg nicht möglich. Um dennoch auf die Daten zugreifen zu können, gibt es zwei unterschiedliche Ansätze.

Zugriff über OLE DB ohne System-DSN

Statt über eine System-DSN zuzugreifen, können Sie in den meisten Fällen auch direkt die entsprechende Datei angeben. In diesem Fall können Sie die bestehenden OLE DB-Treiber verwenden, um Ihre Datenbank zu erreichen.

Im Konstruktor der Klasse `OleDbConnection` wird ein OLE DB-Connection-String übergeben. Über zwei Parameter wird hier der zu verwendende Treiber (`Provider`) sowie der vollständige Dateiname der Datenbank (`Data Source`) übergeben. Relative Namen wie beispielsweise die Datenbank im aktuellen Verzeichnis können mittels `Server.MapPath` umgewandelt werden. Der Zugriff über OLE DB funktioniert nun reibungslos.

Listing 8.4 dsn2.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

public void Page_Load(object sender, EventArgs e)
{
    string connstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
        Server.MapPath("test.mdb");
    OleDbConnection conn = new OleDbConnection(connstr);
    conn.Open();
    // ...
}

</script>
```

Dieser Aufruf hat einen entscheidenden Nachteil. Sie müssen den vollständigen Connection-String in jeder Seite wiederholen, die auf die Datenbank zugreifen soll. Insbesondere bei Änderungen an der Verzeichnisstruktur oder dem Dateinamen der Datenbank können hier lästige Wartungsarbeiten nach sich ziehen.

Ich empfehle, den Connection-String als individuelle Konfiguration innerhalb der zentralen `web.config` abzulegen. Spätere Änderungen müssen nur einmal durchgeführt werden und wirken sich automatisch auf alle relevanten Seiten aus.

Die Konfiguration wird im Abschnitt `<appSettings>` hinterlegt. Eine entsprechende Konfigurationsdatei könnte beispielsweise wie folgt aussehen:

Listing 8.5 `web.config`

```
<configuration>
  <appSettings>
    <add key="dsn" value="Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=c:\inetpub\asp.net\test.mdb" />
  </appSettings>
  <system.web>
    <!-- ... -->
  </system.web>
</configuration>
```

Der Zugriff auf die hinterlegten Daten innerhalb einer ASP.NET-Seite erfolgt über die `NameValueCollection`, die von der statischen Methode `ConfigurationSettings.AppSettings` geliefert wird. Der Aufruf sieht beispielsweise so aus:

Listing 8.6 `dsn3.aspx`

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

public void Page_Load(object sender, EventArgs e)
{
  string connstr = ConfigurationSettings.AppSettings["dsn"];
  OleDbConnection conn = new OleDbConnection(connstr);
  conn.Open();
  // ...
}

</script>
```

Auf diese Weise können Sie den Quellcode und dessen Konfiguration trennen und müssen Änderungen nur noch einmalig zentral vornehmen.

Zugriff über ODBC mit System-DSN

Mit dem .NET Framework werden standardmäßig Data Provider für OLE DB und den direkten Zugriff auf den SQL Server ab Version 2000 ausgeliefert. Microsoft bastelt jedoch auch an der Unterstützung anderer Formate wie beispielsweise Oracle. Ein bereits fertiger dritter Data Provider ist für den Zugriff auf Datenbanken mittels entsprechender ODBC-Treiber möglich.

Um den neuen Data Provider nutzen zu können, müssen Sie diesen zunächst vom Microsoft-Server herunterladen und auf dem lokalen System beziehungsweise Web-Server installieren. Das Paket ist knapp 800 Kilobyte groß und problemlos eingerichtet:

```
http://www.microsoft.com/downloads/release.asp?ReleaseID=35715
```

Anschließend wurde eine neue Assembly `Microsoft.Data.Odbc` im Global Assembly Cache registriert. Damit Sie diese innerhalb einer ASP.NET-Seite nutzen können, müssen Sie zunächst einen Verweis auf die Assembly einfügen. Dies geschieht, wie nicht anders zu erwarten war, über die Konfigurationsdatei `web.config` und sieht wie folgt aus:

Listing 8.7 web.config

```
<configuration>
  <system.web>
    <compilation debug="true">
      <assemblies>
        <add assembly="Microsoft.Data.Odbc,
          Version=1.0.3300.0, Culture=neutral,
          PublicKeyToken=b77a5c561934e089"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Nun können Sie innerhalb Ihrer Seiten den neuen Namespace einbinden. Da die Assembly nicht zum regulären Umfang des Frameworks gehört und somit quasi von einem Drittanbieter stammt, hört der Namespace auf den Namen `Microsoft.Data.Odbc`.

Die Objekte wurden analog zu den Implementierungen für OLE DB und den SQL-Server realisiert. Zunächst öffnen Sie die Datenbankverbindung mittels einer `OdbcConnection`. Im Konstruktor der Klasse können Sie auch hier den gewünschten Connection-String angeben. In Verbindung mit dem Parameternamen DSN können Sie hier auch endlich die gewünschte System-DSN verwenden.

Das Listing zeigt den Zugriff auf eine eingerichtete System-DSN. Der weitere Zugriff verläuft analog zu den bisherigen Data Providern – statt `OleDb...` beziehungsweise `Sql...` sind die Klassen hier jedoch mit dem Präfix `Odbc...` versehen.

Listing 8.8 dsn4.aspx

```
<% @Import Namespace="Microsoft.Data.Odbc" %>

<script runat="server">

public void Page_Load(object sender, EventArgs e)
{
    OdbcConnection conn = new OdbcConnection("DSN=test");
    conn.Open();
    // ...
}

</script>
```

Fazit

Die OLE DB-Treiber sind schneller, und das .NET Framework ist dafür ausgelegt. Jede Framework-Installation bringt immer auch die entsprechenden Treiber mit. Ob die Provider hingegen den optionalen ODBC Data Provider installieren, ist fraglich. Wann immer möglich, sollten Sie sich daher von dem Zugriff per System-DSN trennen und direkt über die Datei zugreifen.

8.3 ... eine Connection lokal und beim Provider verwenden?

Die überwiegende Anzahl der Entwickler arbeitet mit lokalen Entwicklungskopien ihrer Web-Applikation. Erst wenn die einzelnen Module fertig sind, werden diese auf den eigenen oder zumeist den Server des Providers übertragen. In vielen Fällen verfügen die beiden Systeme über eine unterschiedliche Konfiguration. Betroffen sind insbesondere die Verzeichnisse, so dass sich eine Datenbank beim Provider in einem anderen Verzeichnis befindet, als dies lokal der Fall ist.

Um diesem offensichtlichen Problem vorzubeugen, wurde bei früheren Versionen von ASP eine systemweit gültige System-DSN verwendet. Diese erlaubte einen einheitlichen Zugriff auf die Datenbank über den vergebenen Alias. Warum dieser Ansatz bei ASP.NET in dieser Form meist nicht mehr verwendbar ist, lesen Sie im Rezept „... auf eine System-DSN zugreifen?“.

Bei ASP.NET erfolgt der Zugriff in aller Regel über einen festen Dateinamen. Damit Sie diesen Pfad nach der Übertragung der Seiten auf den Real-Server nicht innerhalb jeder Seite ändern müssen, empfiehlt sich die Ablage des gesamten Connection-Strings innerhalb der Konfigurationsdatei `web.config`. Derart benutzer-spezifische Einträge werden im Zweig `appSettings` direkt unterhalb von `configuration` abgelegt. Es handelt sich um ein Schlüssel-Wert-System, das in der folgenden Variante der Konfigurationsdatei zu erkennen ist.

Listing 8.9 `web.config`

```
<configuration>
  <appSettings>
    <add key="ConnectionString"
      value="Provider=Microsoft.Jet.OLEDB.4.0;
        Data Source=c:\inetpub\wwwroot\aspnet\countries.mdb"/>
  </appSettings>
</configuration>
```

Auf die so hinterlegte Konfiguration kann innerhalb einer ASP.NET-Seite über eine `NameValueCollection` zugegriffen werden, die die statische Eigenschaft `ConfigurationSettings.AppSettings` liefert.

Listing 8.10 `Connection1.aspx`

```
void Page_Load(object sender, EventArgs e)
{
    string ConnectionString =
        ConfigurationSettings.AppSettings["ConnectionString"];
    OleDbConnection conn = new OleDbConnection(ConnectionString);
    conn.Open();
    ...
}
```

Diese Art der Ablage bedeutet, dass Sie zwei Versionen der Konfigurationsdatei pflegen müssen, eine für die lokale Installation und eine für den echten Server. In manchen Fällen können Sie dies umgehen, wenn die Datenbank im gleichen relativen Pfad zu finden ist. Hier hilft die Methode `Server.MapPath`, um den jeweiligen lokalen und absoluten Pfad zu ermitteln. Der eignet sich ebenfalls in Verbindung mit der Konfigurationsdatei, der Eintrag muss jedoch aufgeteilt werden.

Listing 8.11 web.config

```

<configuration>
  <appSettings>

    <add key="ConnectionString"
      value="Provider=Microsoft.Jet.OLEDB.4.0;Data Source="/>

    <add key="ConnectionPath"
      value="/~db/countries.mdb"/>

  </appSettings>
</configuration>

```

Der Grundbestandteil des Connection-Strings sowie der relative Pfad zur Datenbank können nun getrennt abgefragt, evaluiert und zusammengefügt werden. Das Beispiel zeigt dies. Dabei wird auch ein Caching berücksichtigt, damit die Umwandlung nicht bei jedem Aufruf neu erfolgen muss.

Listing 8.12 Connection2.aspx

```

void Page_Load(object sender, EventArgs e)
{
  if(Cache["ConnectionString"] == null)
  {
    string ConnectionString =
      ConfigurationSettings.AppSettings["ConnectionString"];
    string ConnectionPath =
      ConfigurationSettings.AppSettings["ConnectionPath"];
    ConnectionPath = Server.MapPath(ConnectionPath);
    ConnectionString += ConnectionPath;
    Cache["ConnectionString"] = ConnectionString;
  }

  OleDbConnection conn = new OleDbConnection((string)
    Cache["ConnectionString"]);
  conn.Open();

  ...

```

Damit Sie eine derartige Funktionalität nicht in jeder Seite getrennt implementieren, empfiehlt sich die Anlage einer globalen Methode. Wie dies geht, erfahren Sie im Rezept „...Hilfsfunktionen global hinterlegen?“ im Kapitel „Sprachelemente“.

8.4 ... das Schema einer Tabelle abfragen?

Es gibt eine Reihe unterschiedlicher Möglichkeiten, die Schemainformationen einer Datenbanktabelle abzufragen. Die im Rezept „... einen strongly typed DataSet anlegen?“ beschriebene Methode `WriteXmlSchema` ist eine davon. Sofern Sie bereits über ein DataSet mit den entsprechenden Daten verfügen, können Sie programmatisch auf die Columns-Collection der jeweiligen Tabelle zugreifen. Das Listing zeigt, wie diese Collection als Datenquelle für ein DataGrid-Control dienen kann.

Listing 8.13 Columns1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    OleDbDataAdapter adapter = new OleDbDataAdapter();
    adapter.SelectCommand = cmd;

    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Authors");
    conn.Close();

    datagrid.DataSource = dataset.Tables["Authors"].Columns;
    DataBind();
}

</script>

<ASP:DataGrid id="datagrid" runat="server"/>
```

Prefix	DesignMode	ReadOnly	ColumnName	Expression	Ordinal	Caption	AllowDBNull	AutoIncrement	See
	False	False	ID		0	ID	True	0	
	False	False	Firstname		1	Firstname	True	0	
	False	False	Lastname		2	Lastname	True	0	

Abbildung 8.2 Die Spalten einer DataTable

Die Abbildung zeigt die gewonnenen Informationen. Ausgegeben werden alle Eigenschaften der Klasse `DataColumn`, also die Meta-Daten der unterliegenden DataTable.

Eine weitere Möglichkeit, derartige Meta-Informationen auszulesen, bietet der `DataReader`. Hier existiert eine Methode `GetSchemaTable`. Zurückgeliefert wird eine Instanz der Klasse `DataTable` mit den beschreibenden Daten.

Listing 8.14 `GetSchemaTable1.aspx`

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM AUTHORS";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    OleDbDataReader reader = cmd.ExecuteReader();

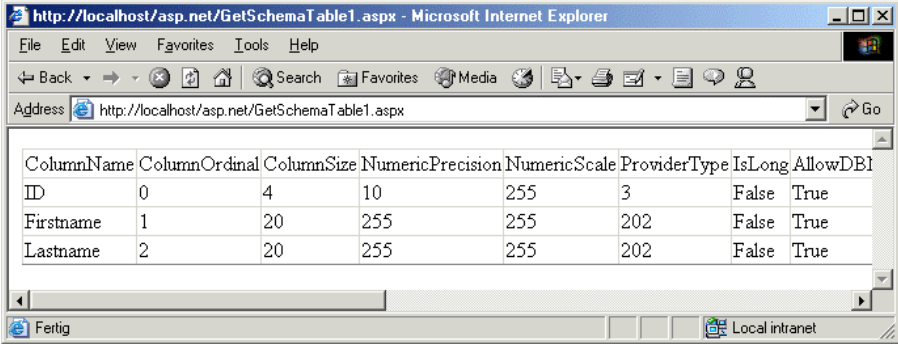
    DataTable schema = reader.GetSchemaTable();
    datagrid.DataSource = schema;
    DataBind();

    reader.Close();
    conn.Close();
}

</script>

<ASP:DataGrid id="datagrid" runat="server"/>
```

Das Listing zeigt die Abfrage der Schema-Informationen über die Klasse `OleDbDataReader`. Die zurückgelieferte `DataTable` wird auch hier an ein `DataGrid`-Control gebunden. Die Abbildung zeigt die Ausgabe im Browser. Die gewonnenen Daten unterscheiden sich deutlich von den vorherigen. Die Ursache hierfür liegt in der Quelle der Daten. Im vorherigen Listing handelte es sich um die Meta-Informationen des DataSets. Hier sind es nun die „offiziellen“ Daten des unterliegenden OLE DB-Datenbanktreibers.



ColumnName	ColumnOrdinal	ColumnSize	NumericPrecision	NumericScale	ProviderType	IsLong	AllowDBNull
ID	0	4	10	255	3	False	True
Firstname	1	20	255	255	202	False	True
Lastname	2	20	255	255	202	False	True

Abbildung 8.3 Die Daten stammen direkt vom OLE DB-Treiber.

8.5 ... per SQL eine neue Tabelle anlegen?

Eine Tabelle mit Access oder den Tools des SQL-Servers anzulegen ist ein Kinderspiel. Doch was passiert im Hintergrund? Hier werden SQL-Befehle gegen die Engine abgesetzt. Sofern Sie nicht direkt auf eine Datenbank zugreifen können, können Sie mit Hilfe dieser SQL-Befehle dennoch tiefgreifende Änderungen vornehmen. Mit Hilfe des Befehls `CREATE TABLE` können Sie etwa eine neue Tabelle anlegen.

Der Aufruf des Befehls erfolgt über die Methode `ExecuteNonQuery` der Klassen `OleDbCommand` beziehungsweise `SqlCommand`. Neben dem Tabellennamen können auch die initiiell anzulegenden Felder samt deren Datentyp übergeben werden. Es ergibt sich dabei folgendes Schema:

```
CREATE TABLE <Tabellename> (<Feldname1> <Feldtyp1>,  
<Feldname2> <Feldtyp2>, ...);
```

Während die Namen für Tabelle und Felder weitgehend frei gewählt werden können, muss der angegebene Datentyp einer Liste von vordefinierten Werten entnommen werden. Die Tabelle zeigt diese.

Tabelle 8.1 Wichtige Datentypen bei der Anlage eines Feldes

Datentyp	Beschreibung/C# Datentyp
Varchar(n)	Textfeld mit der Länge n, maximal möglich sind 255 Zeichen
Text	Memo-Feld mit maximal 32.768 Zeichen (2 ¹⁵)
Integer	4 Byte, int
Smallint	2 Byte, short
Float	float
DateTime	DateTime
Bit	bool

Das Listing zeigt die Anlage einer neuen Tabelle mit einigen Feldern unterschiedlichen Datentyps. Die Abbildung zeigt das Ergebnis im Entwurfsfenster von Access.

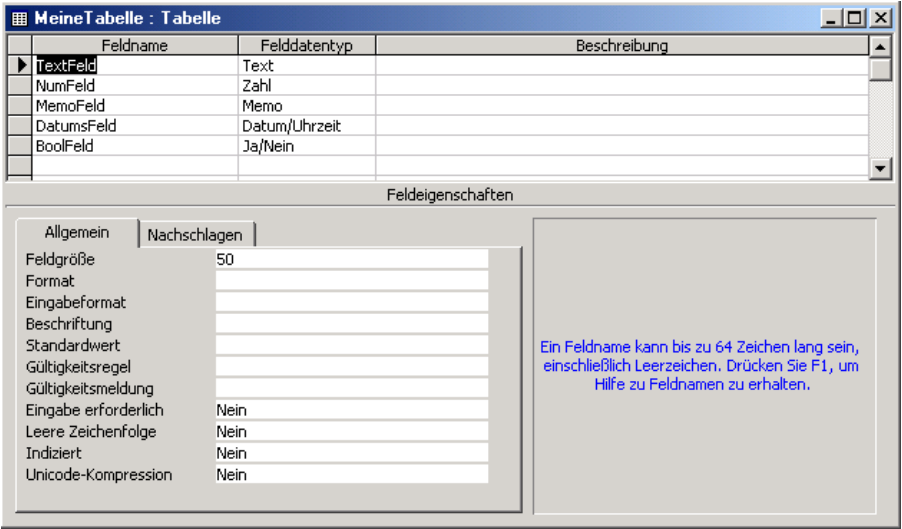


Abbildung 8.4 Die Tabelle wurde erfolgreich angelegt.

Listing 8.15 CreateTable1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\empty.mdb");
    conn.Open();
}
```

```
string SQL = "CREATE TABLE MeineTabelle (TextFeld Varchar(50),
    NumFeld Integer, MemoFeld Text, DatumsFeld Datetime,
    BoolFeld Bit);";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
cmd.ExecuteNonQuery();

conn.Close();
}

</script>
```

Neben dieser Erstellung einer Tabelle haben Sie auch die Möglichkeit, erweiterte Angaben zu den einzelnen Feldern zu übergeben. So können Sie beispielsweise null als Inhalt erlauben, einen Standardwert vorgeben oder einen Primärindex anlegen. Die Tabelle zeigt einige der dazu notwendigen Schlüsselwörter.

Tabelle 8.2 Erweiterte Schlüsselwörter zur Feldanlage

Schlüsselwort	Beschreibung
NULL	null als Wert ist erlaubt
NOT NULL	null als Wert ist nicht erlaubt
DEFAULT	Ermöglicht die Angabe eines Standardwerts, beispielsweise Now() für Datumsfelder mit dem aktuellen Datum bei Datensatzanlage
PRIMARY KEY	Legt einen Primärindex an
REFERENCES	Legt eine Referenz auf ein anderes Feld einer anderen Tabelle in der Datenbank an

Eine ausführliche Beschreibung der einzelnen Befehle entnehmen Sie bitte der individuellen Dokumentation Ihrer Datenbank-Engine oder der MSDN.

8.6 ... per SQL eine neue Spalte anlegen oder ändern?

Mit Hilfe von SQL ist es möglich, nachträgliche neue Spalten anzulegen, bestehende zu modifizieren oder diese gar zu löschen. Zum Einsatz kommt hierbei der SQL-Befehl ALTER TABLE (englisch: ändere Tabelle).

Neue Spalte anlegen

Eine neue Spalte wird bei einer bestehenden Tabelle ähnlich angelegt wie bei einer neuen. Die möglichen Datentypen und Zusatzoptionen finden Sie im Rezept „... per SQL eine neue Tabelle anlegen?“ weiter oben. Das zu verwendende Schema sieht wie folgt aus:

```
ALTER TABLE <Tabellenname> ADD <Feldname> <Feldtyp>;
```

Das Listing zeigt die Ergänzung einer Tabelle um ein Textfeld mit einem Umfang von maximal zehn Zeichen.

Listing 8.16 AlterTable1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\empty.mdb");
    conn.Open();

    string SQL = "ALTER TABLE MeineTabelle ADD NeuesFeld Varchar(10);";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.ExecuteNonQuery();

    conn.Close();
}

</script>
```

Bestehende Spalte ändern

Die Änderung eines bestehenden Feldes erfolgt ganz ähnlich der Neuanlage. Lediglich das Wort `add` wird durch `ALTER` ersetzt. Das Listing zeigt die Erweiterung des zuvor angelegten Feldes von zehn auf 20 Zeichen.

Listing 8.17 AlterTable2.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
```

```
@ "Data Source=c:\inetpub\wwwroot\aspnet\empty.mdb");
conn.Open();

string SQL = "ALTER TABLE MeineTabelle ALTER NeuesFeld Varchar(20);";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
cmd.ExecuteNonQuery();

conn.Close();
}

</script>
```

Bestehende Spalten löschen

Auch das Löschen von angelegten Feldern ist mit Hilfe von `ALTER TABLE` möglich. Als Unterbefehl wird hier `DROP` gefolgt von dem zu löschenden Feld angegeben.

Listing 8.18 AlterTable3.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\empty.mdb");
    conn.Open();

    string SQL = "ALTER TABLE MeineTabelle DROP NeuesFeld;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.ExecuteNonQuery();

    conn.Close();
}

</script>
```

8.7 ... per SQL einen neuen Datensatz anlegen?

Bei ADO.NET haben Sie die Möglichkeit, neue Datensätze über die `DataSet`-Klasse quasi automatisch neu erstellen zu lassen. Insbesondere wenn jedoch nur kurz ein einfacher Datensatz angelegt werden soll, erscheint die Verwendung der `DataSet`-Klasse ein wenig übertrieben. In diesem Fall können Sie auf SQL zurückgreifen und die notwendigen Daten manuell an die Datenbank-Engine übergeben.

Die Anlage neuer Datensätze mittels SQL geschieht über den Befehl `INSERT`. Das Schema des Befehls sieht wie folgt aus:

```
INSERT INTO <Tabellenname> (<Feldname1>, <Feldname2>, ...)
VALUES (<Wert1>, <Wert2>, ...);
```

Ein wenig ungewöhnlich erscheint die Trennung von Feldname und dem entsprechenden Wert. Bei vielen Feldern kann dies mitunter ein wenig unübersichtlich werden. Nicht so allerdings bei dem folgenden Beispiel. Über zwei Eingabefelder kann einer neuer Eintrag zu der Liste der Buchautoren hinzugefügt werden. Per Buttonklick wird eine SQL-Query gegen die Datenbank abgesetzt.

Listing 8.19 Insert1.aspx

```
<script language="C#" runat=server>

OleDbConnection conn;

void Page_Load(object sender, EventArgs e)
{
    conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
        @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    if(!IsPostBack)
        BindData();
}

void Page_Unload(object sender, EventArgs e)
{
    conn.Close();
}

void BindData()
{
    string SQL = "SELECT * FROM Authors";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();
}
```

```
void bt_click(object sender, EventArgs e)
{
    string SQL = "INSERT INTO Authors (Firstname, Lastname) VALUES ('"
        + tb_fn.Text + "', '" + tb_ln.Text + "')";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.ExecuteNonQuery();
    BindData();
}

</script>

<ASP:DataGrid id="dg" runat="server"/>

<h2>Neuer Autor</h2>

<form runat="server">

    <p> Vorname:
        <ASP:TextBox id="tb_fn" runat="server"/>
    </p>

    <p> Nachname:
        <ASP:TextBox id="tb_ln" runat="server"/>
    </p>

    <ASP:Button runat="server" text="Hinzufügen" onclick="bt_click"/>

</form>
```

Die Connection ist als globales Feld definiert und wird im Page_Unload-Ereignis geschlossen. Dies ist sehr wichtig, denn ansonsten würde die Verbindung zur Datenbank bestehen bleiben. Davon abgesehen spricht das Beispiel jedoch für sich, und die Abbildung tut ihr Übriges. Hier ist der neu hinzugefügte Datensatz deutlich im DataGrid-Control erkennbar.

Das System funktioniert so lange einwandfrei, bis ein Benutzer ein ungültiges Zeichen eingibt. Ich spiele insbesondere auf das einfache wie doppelte Anführungszeichen an, die bei SQL zur Eingrenzung der Textdaten dienen. Die Eingabe eines dieser Zeichen resultiert in einer `OleDbException`.

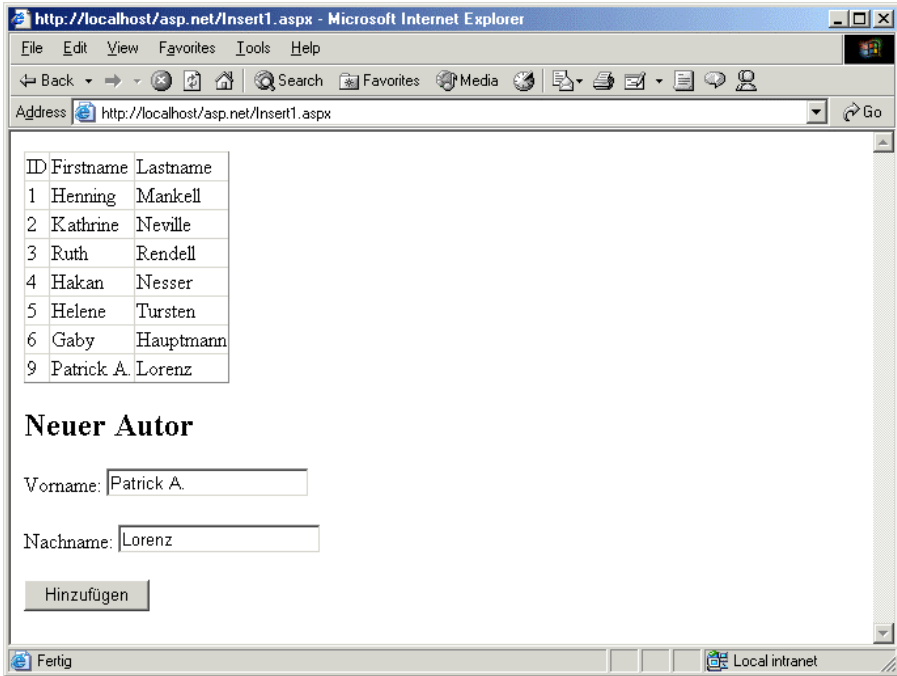


Abbildung 8.5 Der Autor wurde der Tabelle als neuer Datensatz hinzugefügt.

Was also tun? Das Filtern nicht erlaubter Zeichen ist eine – wenngleich unschöne – Möglichkeit. Viel eleganter ist hingegen die Verwendung der Parameters-Collection. Hier können Platzhalter innerhalb der SQL-Query definiert werden, die programmatisch über eine Collection mit Inhalt gefüllt werden. Nachfolgend sehen Sie das entsprechend angepasste Beispiel.

Listing 8.20 Insert2.aspx

```
...
void bt_click(object sender, EventArgs e)
{
    string SQL = "INSERT INTO Authors (Firstname, Lastname)
        VALUES (@Firstname, @Lastname)";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Firstname", tb_fn.Text);
    cmd.Parameters.Add("@Lastname", tb_ln.Text);
    cmd.ExecuteNonQuery();
    BindData();
}
...
```

Im Listing werden der SQL-Query zwei Platzhalter `@Firstname` und `@Lastname` zugewiesen. Über die `Parameters-Collection` (einem Dictionary) werden anschließend die tatsächlichen Daten übergeben. Die Konvertierung in eine Zeichenkette und die Angabe von Anführungsstrichen ist in diesem Fall nicht notwendig. Finden Sie diesen Weg nicht auch *viel* eleganter?



Die mutwillige Ausnutzung derartiger Sicherheitslöcher nennt man übrigens SQL Injection. Mehr zu diesem Thema erfahren Sie innerhalb des Kapitels im Rezept „... SQL Injection verhindern?“.

8.8 ... die automatische ID des zuletzt eingefügten Datensatzes abfragen?

Viele Tabellen verwenden zur eindeutigen Identifizierung eines Datensatzes einen automatisch generierten Primärschlüssel. Hierbei handelt es sich um einen inkrementell oder zufällig vergebenen numerischen Wert. Fügt man einen neuen Datensatz hinzu, weist die Engine dem Datensatz eine neue ID zu. Doch was, wenn diese ID auch innerhalb der Seite zur weiteren Bearbeitung benötigt wird? Hier hilft ein kleiner Trick in Form der folgenden, allgemein gültigen SQL-Abfrage:

```
SELECT @@Identity;
```

Zurückgeliefert wird der letzte, automatisch vergebene ID-Wert. Kollisionen mit anderen Benutzern der Datenbank sind übrigens ausgeschlossen, da sich die Abfrage ausschließlich auf die aktuelle Connection bezieht.

Ausgehend von dem Beispiel des vorherigen Rezepts zum Einfügen von Datensätzen zeigt das folgende Listing die anschließende Abfrage der zuletzt vergebenen ID.

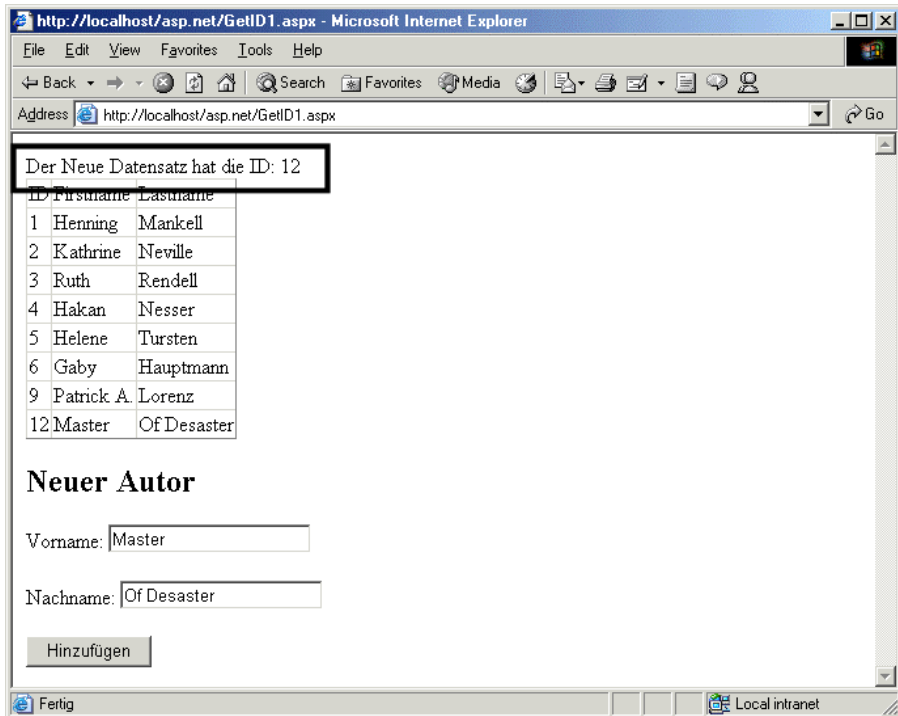


Abbildung 8.6 Der neu eingefügte Datensatz trägt die ID 12.

Listing 8.21 GetID1.aspx

```
...

void bt_click(object sender, EventArgs e)
{
    string SQL = "INSERT INTO Authors (Firstname, Lastname) VALUES
(@Firstname, @Lastname)";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Firstname", tb_fn.Text);
    cmd.Parameters.Add("@Lastname", tb_ln.Text);
    cmd.ExecuteNonQuery();

    cmd.CommandText = "SELECT @@Identity;";
    int id = (int) cmd.ExecuteScalar();
    Response.Write("Der neue Datensatz hat die ID: " + id.ToString());

    BindData();
}

...
```

8.9 ... per SQL einen bestehenden Datensatz aktualisieren?

Die Änderung von bestehenden Datensätzen ist mit Sicherheit eine Standardanforderung an eine Datenbank. Bei SQL erledigt dies der UPDATE-Befehl. Dieser arbeitet ähnlich wie der SELECT-Befehl und die Änderungen wirken sich auf alle Datensätze aus, die über den WHERE-Teil angegeben wurden. Folgendes Schema wird verwendet:

```
UPDATE <Tabellenname> SET <Feld1> = <Wert1>,  
<Feld2> = <Wert2> ... WHERE <BEDINGUNG>;
```

Das folgende Beispiel zeigt den Einsatz eines DataGrid-Controls. Über eine Edit-CommandColumn-Spalte können die Inhalte bearbeitet werden. Eine UPDATE-Query innerhalb der Behandlung des OnUpdateCommand-Ereignisses sorgt dafür, dass die Änderungen direkt in die Datenbank übernommen werden.

Listing 8.22 Update1.aspx

```
<script language="C#" runat=server>  
  
OleDbConnection conn;  
  
void Page_Load(object sender, EventArgs e)  
{  
    conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +  
        @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");  
    conn.Open();  
  
    if (!IsPostBack)  
        BindData();  
}  
  
void Page_Unload(object sender, EventArgs e)  
{  
    conn.Close();  
}  
  
void BindData()  
{  
    string SQL = "SELECT * FROM Authors";  
    OleDbCommand cmd = new OleDbCommand(SQL, conn);  
    dg.DataSource = cmd.ExecuteReader();  
    DataBind();  
}  
  
void dg_Edit(Object sender, DataGridCommandEventArgs e)
```

```

{
    dg.EditItemIndex = e.Item.ItemIndex;
    BindData();
}

void dg_Cancel(Object sender, DataGridCommandEventArgs e)
{
    dg.EditItemIndex = -1;
    BindData();
}

void dg_Update(Object sender, DataGridCommandEventArgs e)
{
    TextBox tb_fn = (TextBox) e.Item.Cells[0].Controls[0];
    TextBox tb_ln = (TextBox) e.Item.Cells[1].Controls[0];

    string SQL = "UPDATE Authors SET Firstname='"
        + tb_fn.Text + "', Lastname='" + tb_ln.Text
        + "' WHERE ID=" + dg.DataKeys[e.Item.ItemIndex] + " ";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.ExecuteNonQuery();

    dg.EditItemIndex = -1;
    BindData();
}

</script>

<form runat="server">

<ASP:DataGrid id="dg" runat="server"
    AutoGenerateColumns="false"
    DataKeyField="ID"
    OnEditCommand="dg_Edit"
    OnCancelCommand="dg_Cancel"
    OnUpdateCommand="dg_Update">

    <Columns>

        <ASP:BoundColumn
            HeaderText="Vorname"
            DataField="Firstname"/>

        <ASP:BoundColumn
            HeaderText="Nachname"
            DataField="Lastname"/>

        <ASP:EditCommandColumn
            EditText="Bearbeiten"

```

```

        CancelText="Abbrechen"
        UpdateText="Übernehmen"/>

</Columns>

</ASP:DataGrid>

</form>

```

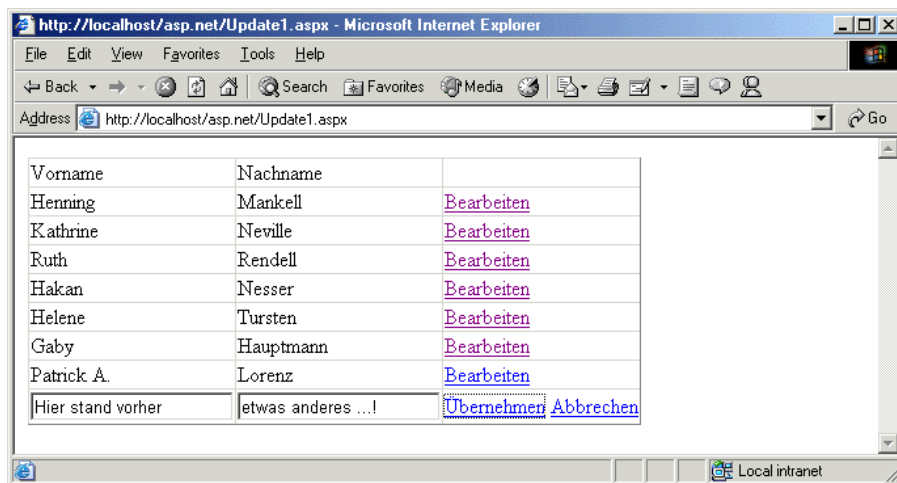


Abbildung 8.7 Mit einfachen Mitteln können die Inhalte der Tabelle geändert werden.

Was für das Einfügen eines Datensatzes gilt, wird bei der Aktualisierung nicht anders gehandhabt. Wie bereits im Rezept „... per SQL einen neuen Datensatz anlegen?“ sollte die Übergabe von Daten an eine SQL-Query aus Sicherheitsgründen ausschließlich über die Parameters-Collection erfolgen. Das zweite Listing zeigt ein entsprechend angepasstes Beispiel:

Listing 8.23 Update2.aspx

```

...

void dg_Update(Object sender, DataGridCommandEventArgs e)
{
    TextBox tb_fn = (TextBox) e.Item.Cells[0].Controls[0];
    TextBox tb_ln = (TextBox) e.Item.Cells[1].Controls[0];

    string SQL = "UPDATE Authors SET Firstname=@Firstname,
        Lastname=@Lastname WHERE ID=@ID;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Firstname", tb_fn.Text);

```

```

cmd.Parameters.Add("@Lastname", tb_ln.Text);
cmd.Parameters.Add("@ID", dg.DataKeys[e.Item.ItemIndex]);
cmd.ExecuteNonQuery();

dg.EditItemIndex = -1;
BindData();
}

...

```



Für den Fall, dass Sie mehrere Datensätze ändern wollen, sind Sie vielleicht an deren genauer Anzahl interessiert. Die Methode `ExecuteNonQuery` liefert diese als `int`-Wert zurück:

```
int changedrecords = cmd.ExecuteNonQuery();
```

8.10 ... per SQL einen Datensatz löschen?

Die bisherigen Rezepte haben bereits einige wichtige SQL-Befehle gezeigt. Was noch fehlt ist die Möglichkeit, bestehende Datensätze zu löschen. Hierzu wird der `DELETE`-Befehl verwendet. Ähnlich zu den `SELECT`- und `UPDATE`-Befehlen wird der oder werden die zu löschenden Datensätze über eine `WHERE`-Bedingung selektiert. Es ergibt sich folgendes Schema:

```
DELETE FROM <Tabellenname> WHERE <BEDINGUNG>;
```

Ich habe das Beispiel aus dem vorhergehenden Rezept ein wenig modifiziert. Es existiert nun eine zusätzliche `ButtonColumn`-Spalte, über die sich ein Datensatz löschen lässt.

Listing 8.24 Delete1.aspx

```

...

void dg_ItemCommand(object sender, DataGridCommandEventArgs e)
{
    if (e.CommandName == "delete")
    {
        string SQL = "DELETE FROM Authors WHERE ID=@ID;";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        cmd.Parameters.Add("@ID", dg.DataKeys[e.Item.ItemIndex]);
        cmd.ExecuteNonQuery();

        BindData();
    }
}

```

```
}  
}  
  
...  
  
<form runat="server">  
  
  <ASP:DataGrid id="dg" runat="server"  
    AutoGenerateColumns="false"  
    DataKeyField="ID"  
    OnEditCommand="dg_Edit"  
    OnCancelCommand="dg_Cancel"  
    OnUpdateCommand="dg_Update"  
    OnItemCommand="dg_ItemCommand">  
  
    <Columns>  
  
      <ASP:BoundColumn  
        HeaderText="Vorname"  
        DataField="Firstname"/>  
  
      <ASP:BoundColumn  
        HeaderText="Nachname"  
        DataField="Lastname"/>  
  
      <ASP:EditCommandColumn  
        EditText="Bearbeiten"  
        CancelText="Abbrechen"  
        UpdateText="Übernehmen"/>  
  
      <ASP:ButtonColumn  
        Text="Löschen"  
        CommandName="delete"/>  
  
    </Columns>  
  
  </ASP:DataGrid>  
  
</form>
```

Das ItemCommand-Ereignis wird zur Auswertung des Buttonklicks zum Löschen des Datensatzes verwendet. Hierbei müssen Sie unbedingt den festgelegten Wert der CommandName-Eigenschaft abfragen, da das Ereignis beispielsweise auch beim Bearbeiten und Ändern des Datensatzes von den anderen LinkButton-Controls ausgelöst wird.

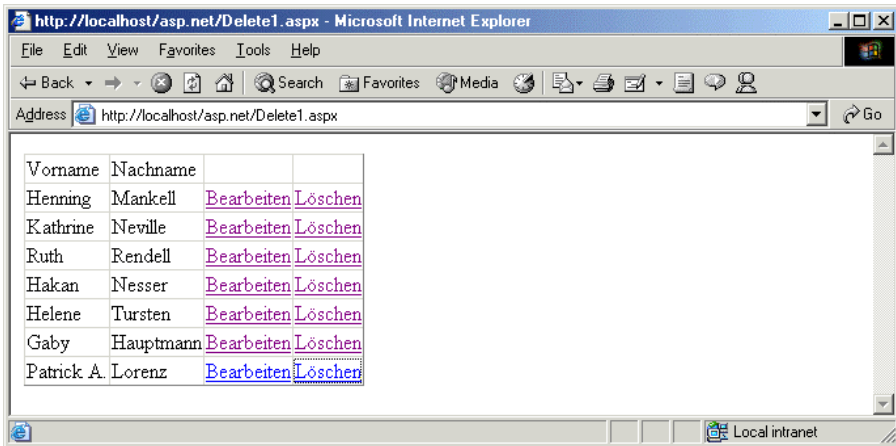


Abbildung 8.8 Über den LinkButton kann der Datensatz direkt gelöscht werden.

8.11 ... die Anzahl von Datensätzen in einer Tabelle ermitteln?

Es ist häufiger wichtig, die Anzahl der Datensätze in einer Datenbanktabelle zu ermitteln. Es gibt zwei recht unterschiedliche Ansätze, das gewünschte Ziel zu erreichen. Natürlich können Sie die gesamte Tabelle in ein DataSet-Objekt kopieren und über die Eigenschaft Count der Rows-Collection der entsprechenden DataTable die Anzahl ermitteln. Das Listing zeigt diesen Ansatz.

Listing 8.25 Count1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    DataSet dataset = new DataSet();

    string SQL = "SELECT * FROM Books;";
    OleDbDataAdapter adapter =
        new OleDbDataAdapter(new OleDbCommand(SQL, conn));
    adapter.Fill(dataset, "Authors");

    conn.Close();
}
```

```
DataTable authors = dataset.Tables["Authors"];
Response.Write("Anzahl Bücher: " + authors.Rows.Count.ToString());
}

</script>
```

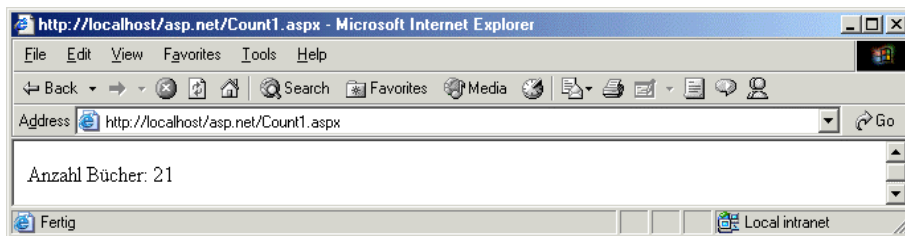


Abbildung 8.9 Die Tabelle „Books“ enthält 21 Datensätze.

Die Abbildung zeigt, dass das System durchaus funktioniert. Allerdings zu welchem Preis? Sämtliche Daten werden in das DataSet-Objekt und somit in den Arbeitsspeicher des Servers kopiert. Bei den gut 20 Datensätzen im Beispiel mag dies vertretbar sein, aber bei 200.000 Einträgen wird der Server deutlich in Bedrängnis kommen. Eine Verbesserung könnte beispielsweise die folgende Abfrage bringen:

```
SELECT ID FROM Books;
```

Nun wird statt der eigentlichen Daten nur noch das ID-Feld in den Speicher kopiert. Doch auch das ist immer noch zu viel. Geschickter ist der folgende Ansatz, der mit Hilfe des SQL-Befehls `COUNT` ausschließlich die Anzahl der Datensätze abfragt. Die Arbeit wird vollständig der Datenbank-Engine überlassen, die vorhandene Optimierungen nutzen kann. Dieser Ansatz spart nicht nur Quellcode, sondern auch in starkem Maße Ressourcen.

Listing 8.26 Count2.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT Count(*) FROM Books;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
```



```

Response.Write("Anzahl Bücher: " + cmd.ExecuteScalar().ToString());

conn.Close();
}

</script>

```

Eine dritte Möglichkeit bietet die Methode `Fill` der Klasse `OleDbDataAdapter`, die zum Füllen einer `DataSet`-Instanz verwendet wird. Diese liefert als Rückgabewert einen `int` mit der Anzahl der übertragenen Datensätze. Aber auch hier müssen alle Datensätze zunächst einmal intern von der Datenbank abgefragt werden.

Listing 8.27 Count3.aspx

```

...

DataSet dataset = new DataSet();

string SQL = "SELECT * FROM Books";
OleDbDataAdapter adapter =
    new OleDbDataAdapter(new OleDbCommand(SQL, conn));
int count = adapter.Fill(dataset, "Authors");

conn.Close();

Response.Write("Anzahl Bücher: " + count.ToString());
}

</script>

```

8.12 ... ermitteln, ob und, wenn ja, wie viele Datensätze geändert wurden?

Die Methode `ExecuteNonQuery` der Klasse `OleDbCommand` beziehungsweise `SqlCommand` wird insbesondere dazu verwendet, INSERT-, UPDATE- oder DELETE-Anforderungen gegen eine Datenbank zu fahren. Der Erfolg der beiden Letztgenannten hängt von dem WHERE-Teil der Query ab. Hier wird bestimmt, ob ein Datensatz, 200 oder vielleicht auch gar keiner modifiziert oder gelöscht wurde.

Um Informationen über das Ergebnis einer derartigen Operation zu erhalten, können Sie den Rückgabewert der Methode auswerten. Hier wird die Anzahl der geänderten Datensätze als int-Wert geliefert. Dadurch lässt sich indirekt beispielsweise auch die Existenz eines Datensatzes überprüfen:

```
// UPDATE ...
if(cmd.ExecuteNonQuery() == 1)
{
    ...
}
```

Die im Listing nachfolgenden Aktionen werden nur ausgeführt, wenn ein Datensatz geändert wurde und dieser folgerichtig auch existiert.

8.13 ... SQL Injection verhindern?

SQL Injection ist ein echtes Sicherheitsproblem bei der direkten Verarbeitung von Benutzereingaben in SQL-Abfragen. Problematisch ist dabei die Umwandlung der Eingaben als Teil der Query. Kennt sich der Benutzer ein wenig aus, kann er geschickt ungültige Daten eingeben und so möglicherweise sicherheitsrelevante Abfragen ausschalten.

Ich möchte Ihnen das Problem anhand eines kleinen Beispiels demonstrieren. Es handelt sich um eine einfache Benutzeranmeldung. Die eingegebene Kombination von Benutzername und Passwort wird in einer Datenbank gesucht. Wird ein passender Datensatz gefunden, ist die Eingabe korrekt, ansonsten nicht. Derartige Abfragen lassen sich bei einer Vielzahl von Web-Applikationen finden.

Listing 8.28 SqlInjection1.aspx

```
<script language="C#" runat=server>

void bt_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\users.mdb");
    conn.Open();

    string SQL = string.Format("SELECT ID FROM USERS WHERE Username='{0}'"
        AND Password='{1}'", tb_us.Text, tb_pw.Text);
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    if(cmd.ExecuteScalar() != null)
        lb_msg.Text = "Alles O.K., die Daten sind korrekt.";
}
```

```

else
    lb_msg.Text = "Ups, das sollten Sie noch einmal überprüfen!";

conn.Close();
}

</script>

<form runat="server">

<h1>Benutzeranmeldung</h1>

<p> Benutzername:
    <ASP:TextBox id="tb_us" runat="server"/>
</p>

<p> Benutzername:
    <ASP:TextBox id="tb_pw" runat="server"/>
</p>

<p> <ASP:Label id="lb_msg" runat="server"/> </p>

<ASP:Button runat="server"
    text="Anmelden"
    onclick="bt_click"/>

</form>

```

Das Beispiel funktioniert prima. Werden keine oder ungültige Daten eingegeben, wird der Benutzer abgewiesen. Nur mit den korrekten Daten aus der abgebildeten Tabelle „Users“ ist eine Anmeldung möglich.

Users : Tabelle			
	ID	Username	Password
▶	1	Patrick	hallowelt
	2	MasterOfDesaster	secret
*	(AutoWert)		

Datensatz: 1 von 2

Abbildung 8.10 Die Benutzerdaten werden zur Anmeldung benötigt.

Aber ist die Anmeldung tatsächlich so sicher? Natürlich nicht. Bitte löschen Sie einmal Ihre Eingaben im Feld „Benutzername“. Nun geben Sie bitte die nachfolgend gezeigte Zeichenkette eins zu eins in das Passwortfeld ein:

```
' OR ''='
```

Tja, und genau das ist SQL Injection. Mehr wird nicht benötigt, um die Benutzeranmeldung zu umgehen und sich mit dem ersten verfügbaren Account anzumelden. Die Abbildung zeigt die positive Bestätigung der Seite. Aber was ist hier genau schief gelaufen? Die Antwort auf diese Frage liegt in der SQL-Abfrage. Nach Eingabe des gezeigten Passworts sieht diese wie folgt aus:

```
SELECT ID FROM USERS WHERE Username='' AND Password='' OR ''=''
```

Die Bedingung trifft auf jeden Datensatz zu, bei dem der Benutzername und das Passwort leer sind oder für die eine leere Zeichenkette identisch ist mit einer leeren Zeichenkette – und genau, das sind eben alle Datensätze.



Abbildung 8.11 Mit einfachen Mittel gelingt ein Einbruch in die Seite.

Problem erkannt, Problem gebannt! Das Zusammensetzen von SQL-Abfragen über Zeichenkettenoperationen sollte unbedingt vermieden werden, wenn die Daten in irgendeiner Form durch den Benutzer beeinflussbar sind.

Stattdessen können Sie Platzhalter verwenden. Dabei notieren Sie innerhalb der Query einen beliebigen, jedoch nicht vergebenen Namen. Typischerweise beginnt dieser mit dem Klammeraffen. Die Übergabe der benötigten Daten erfolgt über die Parameters-Collection der Klasse `OleDbCommand` beziehungsweise `SqlCommand`. Hier übergeben Sie den genutzten Platzhalter sowie dessen Wert.

Das Listing zeigt die Verwendung von Parametern. Im Unterschied zu der direkten Übergabe der Werte als Zeichenkette ist diese Variante sicher. Dies unterstreicht auch die Tatsache, dass die zuvor gezeigte SQL Injection – wie in der Abbildung zu sehen – nicht mehr möglich ist.

Listing 8.29 `SqliInjection2.aspx`

```
<script language="C#" runat=server>

void bt_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\users.mdb");
    conn.Open();

    string SQL = "SELECT ID FROM USERS WHERE Username=@Username AND
        Password=@Password";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Username", tb_us.Text);
    cmd.Parameters.Add("@Password", tb_pw.Text);

    if(cmd.ExecuteScalar() != null)
        lb_msg.Text = "Alles O.K., die Daten sind korrekt.";
    else
        lb_msg.Text = "Ups, das sollten Sie noch einmal überprüfen!";

    conn.Close();
}

</script>

...
```

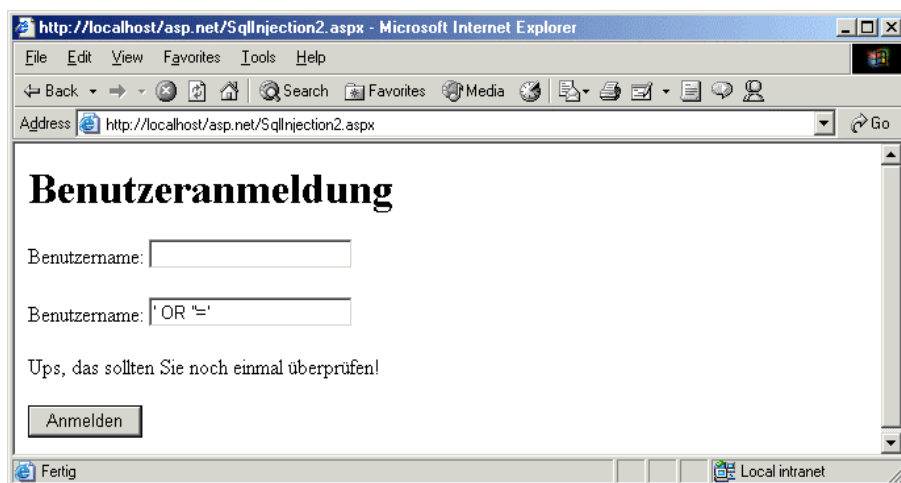


Abbildung 8.12 SQL Injection lässt sich ganz einfach verhindern.



WICHTIG: Beachten Sie bitte unbedingt, dass die Anlage der Parameter in genau der gleichen Reihenfolge erfolgen sollte, wie diese innerhalb der SQL-Query benutzt werden. Auch wenn dies aufgrund der eindeutigen Kennzeichnung über den Schlüssel nicht logisch erscheint, ließ sich in einigen Fällen eine Berücksichtigung der Reihenfolge und somit eine inkorrekte Verwendung der Parameter seitens ADO.NET beziehungsweise der Datenbank-Engine feststellen.

8.14 ... Datensätze sortieren?

Je mehr Datensätze vorhanden sind, desto wichtiger wird eine sinnvolle Sortierung dieser. Bei ADO.NET stehen Ihnen zwei unterschiedliche Ansätze zur Auswahl, die anzuzeigenden Datensätze zu sortieren. Sie können diese Aufgabe entweder der Datenbank-Engine oder dem DataSet überlassen.

In diesem Rezept stelle ich Ihnen beide Möglichkeiten anhand der Autorentabelle vor. Das Listing zeigt den Rumpf des Beispiels. Es handelt sich um ein DataGrid-Control mit automatisch erstellten Spalten. Zudem wurde die Option `AllowSorting` aktiviert und dem Ereignis `SortCommand` eine Behandlungsmethode zugewiesen. Über diese wird der Methode `BindData` das zu sortierende Feld übergeben. In der folgenden ersten Version des Beispiels wird dieses jedoch noch nicht berücksichtigt.

Listing 8.30 Sort1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        BindData();
}

void BindData()
{
    this.BindData(null);
}

void BindData(string sortExpression)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Authors";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    dg.DataSource = cmd.ExecuteReader();
    DataBind();

    conn.Close();
}

void dg_Sort(object sender, DataGridSortCommandEventArgs e)
{
    BindData(e.SortExpression);
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        DataKeyField="ID"
        AllowSorting="true"
        OnSortCommand="dg_Sort"/>

</form>
```

SQL

Die Abfragesprache SQL bietet mit dem Befehl `ORDER BY` eine eingebaute Möglichkeit, Datensätze während der Abfrage aus der Datenbank zu sortieren. Die Mechanismen sind in der Regel sehr performant und optimal auf die jeweilige Engine zugeschnitten. Im Normalfall empfiehlt sich daher die Verwendung von SQL zur Sortierung. Dies gilt insbesondere, wenn ein spezieller Datenbank-Server zur Verfügung steht und der Web Server so entlastet werden kann.

Das folgende Schema zeigt die einfache Verwendung des SQL-Befehls:

```
SELECT * FROM <Tabellenname> WHERE <Bedingung> ORDER BY <Sortierfeld>
```

Sofern die Sortierung nacheinander über mehrere Felder erfolgen soll, geben Sie diese mit Kommata getrennt an. Zu jedem Feld können Sie auch festlegen, ob die Sortierung aufsteigend (Standard) oder absteigend erfolgen soll. Hierzu notieren Sie hinter dem jeweiligen Feld eines der beiden Schlüsselwörter `ASC` oder `DESC`:

```
SELECT * FROM <Tabellenname> WHERE <Bedingung>  
ORDER BY <Sortierfeld> DESC
```

Die Erweiterung des gezeigten Beispiels um eine funktionierende Sortierung ist recht einfach. Innerhalb der Methode `BindData` wird der Parameter `sortExpression` abgefragt und gegebenenfalls mit Hilfe des Befehls `ORDER BY` an die Query angehängt.

Listing 8.31 Sort2.aspx

```
void BindData(string sortExpression)  
{  
    OleDbConnection conn = new  
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +  
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");  
    conn.Open();  
  
    string SQL = "SELECT * FROM Authors";  
  
    if(sortExpression != null)  
        SQL += (" ORDER BY " + sortExpression);  
  
    OleDbCommand cmd = new OleDbCommand(SQL, conn);  
    dg.DataSource = cmd.ExecuteReader();  
    DataBind();  
  
    conn.Close();  
}
```

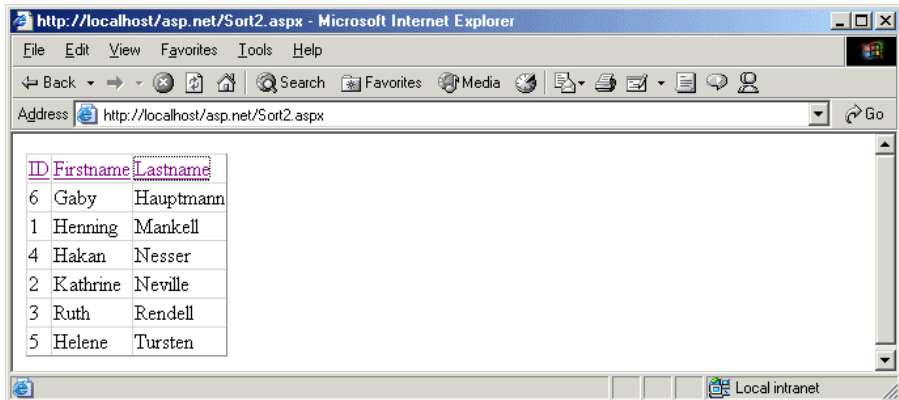



Abbildung 8.13 Die Daten wurde mittels Mausklick nach dem Nachnamen sortiert.

Die Abbildung zeigt den Einsatz der Sortierung. Das DataGrid-Control kümmert sich automatisch um die Bereitstellung entsprechender LinkButton-Controls. Ein Mausklick genügt, um das SortCommand-Ereignis und somit die gewünschte Sortierung auszulösen.

DataSet

Auch im Anschluss an die Abfrage von der Datenbank können Sie Ihre Daten individuell sortieren. Allerdings nur, wenn Sie die DataSet-Klasse und nicht den sequenziellen DataReader einsetzen. Wenn man es genau nimmt, ist für die Sortierung jedoch nicht die Klasse DataSet und auch nicht die herausgepickte DataTable verantwortlich. Vielmehr wird die gewünschte Sortierung der DataView-Klasse zugewiesen. Diese repräsentiert eine individuelle Sicht auf die angeforderten Daten.

Das Beispiel zeigt die Abfrage der Klasse über die Eigenschaft DataTable.DefaultView und die Zuweisung der Sortierung. Hier kommt die Eigenschaft Sort zum Einsatz, der lediglich das gewünschte Feld zugewiesen werden muss. Es können aber ganz analog zu SQL auch mehrere Felder sowie die Richtung der Sortierung angegeben werden.

Listing 8.32 Sort3.aspx

```
...

void BindData(string sortExpression)
{
    OleDbConnection conn = new
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
```

```
conn.Open();

string SQL = "SELECT * FROM Authors";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);

DataSet dataset = new DataSet();
adapter.Fill(dataset);
conn.Close();

DataTable table = dataset.Tables[0];
DataView view = table.DefaultView;

if(sortExpression != null)
    view.Sort = sortExpression;

dg.DataSource = view;
DataBind();
}

...
```

Wie weiter oben bereits beschrieben, sollte die Sortierung per SQL vorgezogen werden. Wann immer Sie jedoch über keinen Zugriff (mehr) auf die Datenquelle verfügen, können Sie die Sortierung auch nachträglich über eine DataView-Instanz vornehmen. Diese wird über ein DataSet abgefragt und kann analog dazu als Quelle zur Datenbindung genutzt werden.

8.15 ... Relationen zwischen Tabellen herstellen?

Ein integraler Bestandteil moderner Datenbanken sind Relationen. Zwei Tabellen lassen sich darüber verknüpfen und so in einen relationalen Kontext stellen. Im Regelfall existiert in einer Tabelle ein eindeutiger Primärindex, über den ein Datensatz dieser Tabelle einer beliebigen Anzahl n der zweiten zugeordnet ist. Diese Zuordnung erfolgt durch Ablage des identischen (Fremd-)Schlüssels in der zweiten Datenbank. Man nennt diesen Fall eine Eins-zu- n -Verknüpfung (1:n).

Die DataSet-Klasse kennt ebenfalls Relationen und erlaubt so, Verknüpfungen mehrerer Tabellen dynamisch abzubilden. Dabei werden *nicht* die ursprünglichen Relationen der Datenquelle verwendet. Sollten diese nachgebildet werden, müssen sie manuell angelegt werden. Das ist allerdings sehr einfach.

Verknüpfungen zwischen Tabellen werden über die Relations-Collection der DataSet-Klasse verwaltet. Eine einzelne Relation wird durch die Klasse `DataRelation` repräsentiert. Diese basiert auf zwei oder mehreren Spalten zweier unterschiedlicher Tabellen. Spalten werden dabei über `DataColumn` und die übergeordneten Tabellen über `DataTable` abgebildet.

Um eine neue Verknüpfung herzustellen, muss der Relations-Collection eine neue Instanz der Klasse `DataRelation` angefügt werden. Dies wird über einen eindeutigen Namen sowie zwei `DataColumn`-Instanzen angelegt. Das Beispiel zeigt dies anhand der Verknüpfung der Tabellen `Authors` und `Books`.

Listing 8.33 `Relations1.aspx`

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    DataSet dataset = new DataSet();

    string SQL = "SELECT * FROM Authors";
    OleDbDataAdapter adapter1 =
        new OleDbDataAdapter(new OleDbCommand(SQL, conn));
    adapter1.Fill(dataset, "Authors");

    SQL = "SELECT * FROM Books";
    OleDbDataAdapter adapter2 =
        new OleDbDataAdapter(new OleDbCommand(SQL, conn));
    adapter2.Fill(dataset, "Books");

    conn.Close();

    DataTable authors = dataset.Tables["Authors"];
    DataColumn id = authors.Columns["ID"];

    DataTable books = dataset.Tables["Books"];
    DataColumn authorid = books.Columns["AuthorID"];

    DataRelation relation = new DataRelation("Authors2Books",
        id, authorid);
    dataset.Relations.Add(relation);
}

</script>
```

Die Anlage einer Relation ist an sich schon recht interessant, um eine entsprechende Ausgabe führt jedoch nichts herum. Hier zeigt sich das wahre Potenzial der Relationen von ADO.NET. Im erweiterten Listing sehen Sie eine Schleife durch alle Datensätze der Autorentabelle. Über die Methode `GetChildRows` werden die relational verknüpften Datensätze eines jeden Autoren und somit die von ihm verfassten Bücher abgefragt und im Browser ausgegeben. Zurückgeliefert wird ein `DataRow`-Array, das seinerseits durchlaufen und ausgegeben wird.

Listing 8.34 Relations2.aspx

```
...

DataTable authors = dataset.Tables["Authors"];
DataColumn id = authors.Columns["ID"];

DataTable books = dataset.Tables["Books"];
DataColumn authorid = books.Columns["AuthorID"];

DataRelation relation = new DataRelation("Authors2Books",
    id, authorid);
dataset.Relations.Add(relation);

Response.Write("<ul>");
foreach(DataRow row in authors.Rows)
{
    string fn = (string) row["firstname"];
    string ln = (string) row["lastname"];
    Response.Write(string.Format("<li>{0} {1}</li>", fn, ln));

    Response.Write("<ul>");
    foreach(DataRow childrow in row.GetChildRows("Authors2Books"))
    {
        Response.Write(string.Format("<li>{0}</li>", childrow["title"]));
    }
    Response.Write("</ul>");
}
Response.Write("</ul>");
}

</script>
```

Die Abbildung zeigt die Ausgabe im Browser. Es sind zwei verschachtelte Auflistungen zu erkennen. Zunächst werden die Autoren ausgegeben und jeweils untergeordnet die entsprechenden Bücher. Die Zuordnung kann ADO.NET selbstständig auf Basis der angelegten Relation vornehmen.

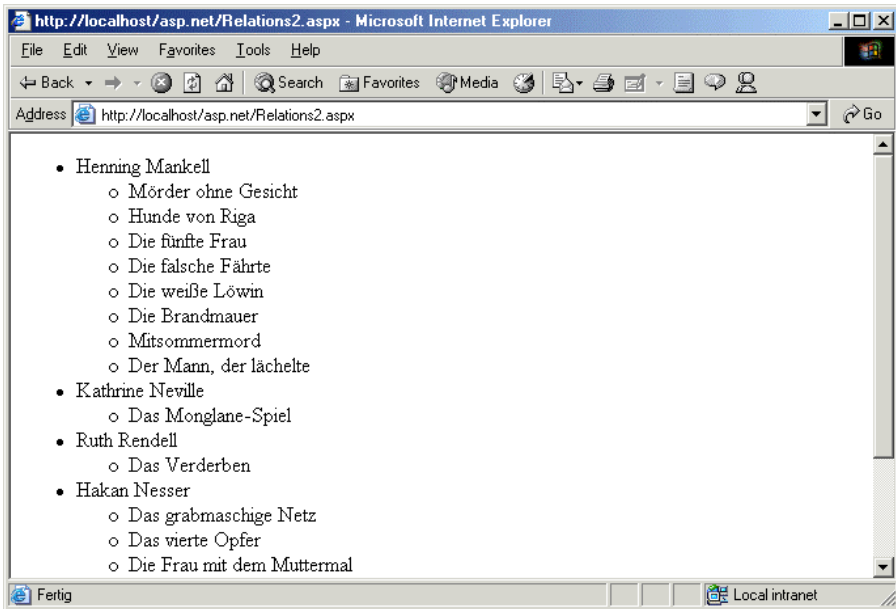


Abbildung 8.14 Die Daten der zwei Tabellen wurden relational verknüpft.

Die Möglichkeiten der von ADO.NET angebotenen Relationen gehen übrigens noch sehr viel weiter. Sie können selbstverständlich beliebig viele derartige Verknüpfungen anlegen und sowohl die Kind- als auch den oder die Elterndatensätze abfragen. Zudem sind auf diese Weise Auswertungen möglich. Hinweise hierzu finden Sie beispielsweise im Rezept „... Daten in einem DataSet filtern?“.

8.16 ... einen hierarchischen Datensatz über das DataGrid-Control anzeigen?

Im vorherigen Rezept „... Relationen zwischen Tabellen herstellen?“ haben Sie gelesen, wie Sie relational verknüpfte Datensätze mit ADO.NET und der DataSet-Klasse abbilden können. Das dort gezeigte Beispiel zur Visualisierung derartiger Master-Detail-Datensätze ist jedoch recht dürftig ausgefallen.

Bevor ich Ihnen den etwas umfangreicheren Quellcode des Beispiels präsentiere, möchte ich Ihnen zunächst das Ergebnis in Form einer Abbildung vorstellen. Diese zeigt ein DataGrid-Control mit den Namen der Autorentabelle. In einer dritten Spalte „Verfügbare Bücher“ wird ein LinkButton-Control angezeigt. Ein Klick befördert eine Liste aller verfügbaren und somit relational zugeordneten Bücher des Autors zur Stelle. Die Anzeige erfolgt dabei anstelle des Links und somit innerhalb des DataGrid-Controls.

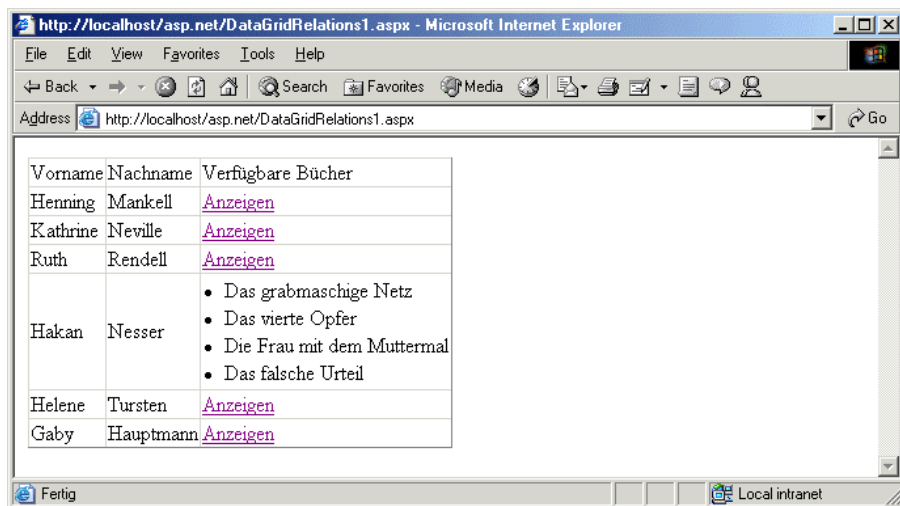


Abbildung 8.15 Die Master-Detail-Daten werden über ein DataGrid-Control dargestellt.

Die Frage der Fragen lautet jetzt verständlicherweise: Und wie geht das? Zusammengefasst handelt es sich um ein DataGrid-Control, in dem wiederum ein DataList-Control abgelegt wurde. Während das erste Objekt die Autorentabelle darstellt, zeigt die zweite die korrespondierenden Bücher an.

Listing 8.35 DataGridRelations1.aspx – Teil 1

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        BindDataGrid();
}

void BindDataGrid()
{
    ...

    DataTable authors = dataset.Tables["Authors"];
    DataColumn id = authors.Columns["ID"];

    DataTable books = dataset.Tables["Books"];
    DataColumn authorid = books.Columns["AuthorID"];

    DataRelation relation = new DataRelation("Authors2Books",
        id, authorid);
    dataset.Relations.Add(relation);
}
```

```

    dg.DataSource = authors;
    DataBind();
}

void dg_ItemCommand(object sender, DataGridCommandEventArgs e)
{
    dg.SelectedIndex = e.Item.ItemIndex;
    BindDataGrid();
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        DataKeyField="ID"
        AutoGenerateColumns="false"
        OnItemCommand="dg_ItemCommand">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Vorname"
                DataField="Firstname"/>

            <ASP:BoundColumn
                HeaderText="Nachname"
                DataField="Lastname"/>

```

Wichtiger Bestandteil des Beispiels ist die dritte Spalte, die mit dem Typ `TemplateColumn` angelegt wurde. Diese enthält zwei `Placeholder-Controls`, die gegensätzlich entsprechend dem aktuellen Zeilentyp umgeschaltet werden. Maßgeblich ist, ob die aktuelle Zeile selektiert ist oder nicht. Ist dies nicht der Fall, wird ein `LinkButton` angezeigt. Ein Klick auf diesen resultiert im `ItemCommand`-Ereignis des `DataGrid`-Controls. Hier wird die aktuelle Zeile markiert und die Datenbindung neu ausgeführt. Dadurch wird das zweite `Placeholder-Control` aktiv, das eine `DataList` anzeigt.

Listing 8.36 DataGridRelations1.aspx – Teil 2

```

<ASP:TemplateColumn
    HeaderText="Verfügbare Bücher">

    <ItemTemplate>

        <ASP:Placeholder runat="server" visible=
            "<%# !(Container.ItemType == ListItemType.SelectedItem) %>">

```

```
<ASP:LinkButton runat="server" text="Anzeigen"
  commandname="select"/>
</ASP:Placeholder>

<ASP:Placeholder runat="server" visible=
  "<%# Container.ItemType == ListItemType.SelectedItem %>">

  <ASP:DataList runat="server"
    datasource=<%# ((DataRowView)
      Container.DataItem).Row.GetChildRows("Authors2Books") %>>

    <HeaderTemplate>
      <ul>
    </HeaderTemplate>

    <ItemTemplate>
      <li>
        <%# DataBinder.Eval(Container.DataItem, "[Title]") %>
      </li>
    </ItemTemplate>

    <FooterTemplate>
      </ul>
    </FooterTemplate>

  </ASP:DataList>

</ASP:Placeholder>

</ItemTemplate>

</ASP:TemplateColumn>

</Columns>

</ASP:DataGrid>

</form>
```

Die Datenbindung des verschachtelten DataList-Controls ist nicht uninteressant, denn diese wird direkt bei der Definition des Controls mittels DataBinding-Syntax zugewiesen. Das sieht zunächst ungewöhnlich aus, ist aber durchaus machbar. Nach einer Typenumwandlung wird das von `GetChildRows` gelieferte DataRow-Array an das Control gebunden.

Innerhalb der `DataList` wird jedes einzelne Buch über die hinterlegte Vorlage ausgegeben. Unbedingt zu beachten ist die spezielle Syntax beim Aufruf von `DataBinder.Eval`. Der Name des gewünschten Feldes – hier „Title“ – muss in eckigen Klammern angegeben werden. Auf diese Weise wird explizit der Indexer und nicht etwa die nicht vorhandene Eigenschaft `Title` der Klasse `DataRow` angesprochen.



Analog zu diesem Beispiel können Sie natürlich auch komplexere Daten visualisieren und beispielsweise zwei oder auch mehrere `DataGrid`-Controls verschachteln. Hier ist allerdings zu beachten, dass die automatische Anlage der Spalten nicht verwendbar ist beziehungsweise innerhalb des `ItemCreated`-Ereignisses des übergeordneten Controls manuell nachgebildet werden muss.

Eine weitere Möglichkeit zur Anzeige von Master-Detail-Beziehungen zeigt der neue Spaltentyp `DetailColumn`, den ich Ihnen im Rezept „... neue Spaltentypen für das `DataGrid` entwickeln?“ im Kapitel „Eingabeformulare“ vorstelle.

8.17 ... zwei relational verknüpfte Datensätze neu anlegen?

Die beiden vorhergehenden Rezepte haben sich mit der relationalen Verknüpfung von Datensätzen mittels ADO.NET und der `DataSet`-Klasse beschäftigt. Konkret wurde der lesende Zugriff gezeigt. Änderungen an bestehenden Datensätzen lassen sich ebenfalls mit den bekannten Mitteln bewerkstelligen. Doch wie schaut es mit der Neuanlage von zwei verknüpften Datensätzen aus? Problematisch ist hierbei die der Verknüpfung zugrunde liegende ID, die ja von der Datenbank erst bei der Neuanlage vergeben wird.

Stellen Sie sich das `DataSet`-Objekt mit den zwei neuen Datensätzen vor. Zunächst wird der Hauptdatensatz in die Datenbank geschrieben. Hierbei wird die ID für diesen vergeben, allerdings nicht an das `DataSet` zurückgeliefert. Bei der Neuanlage des zweiten, untergeordneten Datensatzes wird genau diese ID jedoch benötigt, um die Verknüpfung auf die Datenbank abbilden zu können.

Die Lösung des Problems liegt in den Tiefen der `DataAdapter`-Klassen verborgen. Hier existiert ein Ereignis, das im Anschluss an die Datenbankaktualisierung ausgelöst wird. Mit einem kleinen Trick aus dem Rezept „... die automatische ID des zuletzt eingefügten Datensatzes abfragen?“ kann hier die ID des angelegten Datensatzes ausgelesen und das `DataSet` entsprechend aktualisiert werden.

Das Beispiel zeigt gleichermaßen Problem und Lösung. Es wird ein neues `DataSet`-Objekt angelegt und mit den Schemadaten der beiden Tabellen `Authors` und `Books` gefüllt. Zwischen beiden Tabellen wird eine relationale Verknüpfung angelegt.

Diese wird genutzt, um zwei neue Datensätze anzulegen, pro Tabelle einen. Anschließend werden die neuen Daten mittels der Methode `Update` der beide `OleDbDataAdapter`-Instanzen zurück in die Datenbank geschrieben. Hier werden die angelegten `OleDbCommandBuilder` genutzt.

Listing 8.37 Create2Records1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    DataSet dataset = new DataSet();

    OleDbDataAdapter adapter1 = new OleDbDataAdapter(
        new OleDbCommand("SELECT * FROM Authors;", conn));
    OleDbCommandBuilder cb1 = new OleDbCommandBuilder(adapter1);
    adapter1.FillSchema(dataset, SchemaType.Source);
    adapter1.RowUpdated +=
        new OleDbRowUpdatedEventHandler(adapter1_RowUpdated);
    DataTable authors = dataset.Tables["Table"];
    authors.TableName = "Authors";

    OleDbDataAdapter adapter2 = new OleDbDataAdapter(
        new OleDbCommand("SELECT * FROM Books;", conn));
    OleDbCommandBuilder cb2 = new OleDbCommandBuilder(adapter2);
    adapter2.FillSchema(dataset, SchemaType.Source);
    DataTable books = dataset.Tables["Table"];
    books.TableName = "Books";

    dataset.Relations.Add(new DataRelation("Authors2Books",
        authors.Columns["ID"], books.Columns["AuthorID"]));

    DataRow pRow = authors.NewRow();
    pRow["Firstname"] = "Patrick A.";
    pRow["Lastname"] = "Lorenz";
    authors.Rows.Add(pRow);

    DataRow cRow = books.NewRow();
    cRow["Title"] = "ASP.NET Kochbuch";
    cRow.SetParentRow(pRow);
    books.Rows.Add(cRow);

    adapter1.Update(dataset, "Authors");
    adapter2.Update(dataset, "Books");
}
```

```

    conn.Close();
}

void adapter1_RowUpdated(object sender, OleDbRowUpdatedEventArgs e)
{
    OleDbDataAdapter adapter = (OleDbDataAdapter) sender;
    OleDbConnection conn = adapter.SelectCommand.Connection;
    OleDbCommand cmd = new OleDbCommand("SELECT @@Identity", conn);
    e.Row["ID"] = (int) cmd.ExecuteScalar();
    e.Row.AcceptChanges();
}

</script>

```

Entscheidend für den Erfolg der Operation ist die Behandlung des RowUpdated-Ereignisses des für die Autorentabelle zuständigen OleDbDataAdapter. Hier wird die automatisch von der Datenbank vergebene ID abgefragt und damit das DataSet aktualisiert. Dieses sorgt automatisch dafür, dass der Kinddatensatz ebenfalls aktualisiert wird und die tatsächliche ID bei der anschließenden Übertragung an die Datenbank zur Verfügung steht. Dass dieses System wirklich klappt, zeigt der spätere Aufruf des Beispiels aus dem Rezept „... einen hierarchischen Datensatz über das DataGrid-Control anzeigen?“.

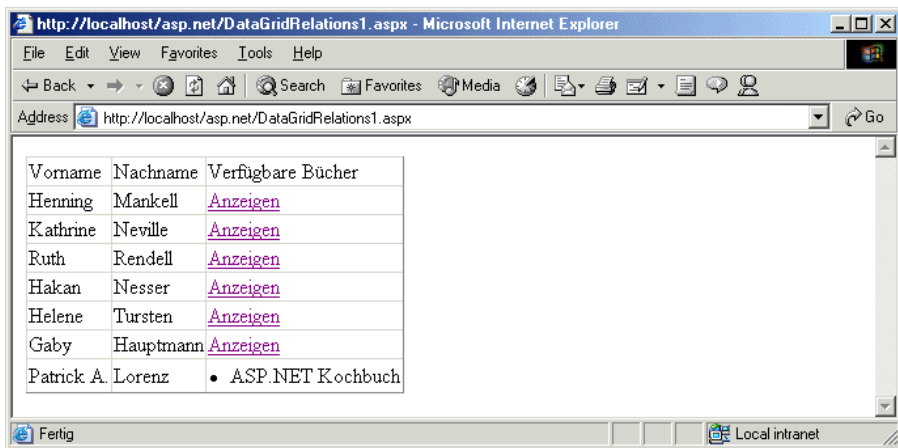


Abbildung 8.16 Die beiden Datensätze wurden erfolgreich gespeichert.

8.18 ... Daten in einem DataSet filtern?

Es ist mitunter hilfreich, die Daten in einem DataSet-Objekt vor der Anzeige zu filtern. Möglicherweise sind nicht alle Datensätze für den Benutzer relevant, oder es soll eine bessere Übersicht gewährleistet werden.

Die Klasse `DataGridView` erlaubt über die Eigenschaft `RowFilter` die Zuweisung einer Filterbedingung. Anschließend werden nur noch die Daten angezeigt, auf die die Bedingung zutrifft. Die erlaubte Syntax entspricht dabei weitgehend den Möglichkeiten von SQL.

Das Beispiel zeigt den Einsatz der Eigenschaft. Das Formular enthält neben einem `DataGridView`-Control für die Tabelle „Books“ eine `TextBox`. Hier kann der Benutzer eine individuelle Filterbedingung eingeben. Ein Klick auf den Aktualisieren-Button weist diese der Standard-`DataGridView`-Instanz der entsprechenden `DataTable` zu und aktualisiert die Ansicht.

Listing 8.38 RowFilter1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        BindDataGrid();
}

void BindDataGrid()
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    DataSet dataset = new DataSet();

    string SQL = "SELECT * FROM Books";
    OleDbDataAdapter adapter =
        new OleDbDataAdapter(new OleDbCommand(SQL, conn));
    adapter.Fill(dataset, "Books");

    conn.Close();

    DataTable books = dataset.Tables["Books"];
    books.DefaultView.RowFilter = tb_filter.Text;

    dg.DataSource = books;
    DataBind();
}
```

```

void bt_click(object sender, EventArgs e)
{
    BindDataGrid();
}

</script>

<form runat="server">

    <p> Filter:
      <ASP:TextBox id="tb_filter" runat="server" text="Title LIKE
'Die%'"/>
      <ASP:Button runat="server" text="Aktualisieren" onclick="bt_click"/>
    </p>

    <ASP:DataGrid id="dg" runat="server"/>

</form>

```

Die Abbildung zeigt das Beispiel im Einsatz. Die vorgegebene Filterbedingung sorgt dafür, dass ausschließlich Bücher angezeigt werden, deren Titel mit „Die“ beginnt. Hierzu wird der Name des Feldes gefolgt von dem Operator `LIKE` und der zu vergleichenden Zeichenkette notiert. Das Prozentzeichen dient als Wildcard. Alternativ kann auch das `*` genutzt werden. Beide Zeichen werden in dieser Form von gängigen SQL-Implementierungen genutzt.

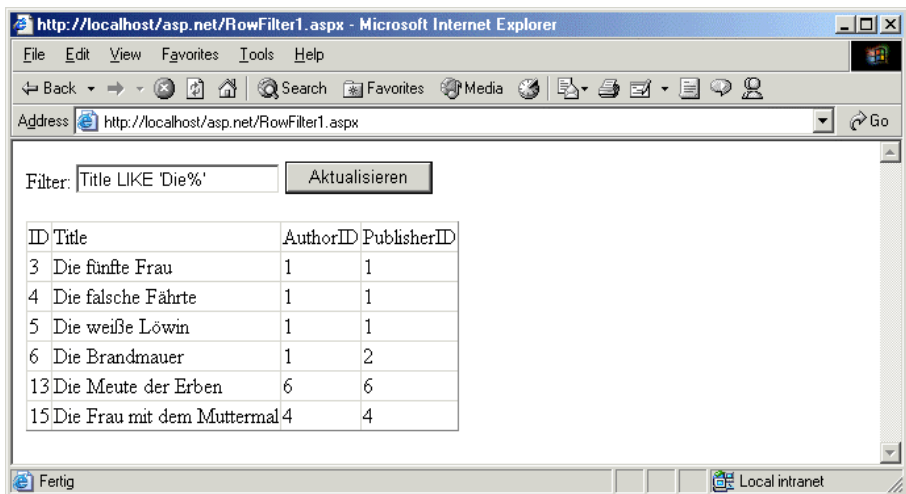


Abbildung 8.17 Es werden nur Bücher angezeigt, deren Titel mit „Die“ beginnt.

Auch weitergehende Möglichkeiten von SQL lassen sich hier nutzen. Im Grunde entspricht die Formel also dem WHERE-Teil einer SQL-Abfrage. Unter anderem sind folgende Funktionen vorgesehen: CONVERT, LEN, ISNULL, IIF, SUBSTRING und TRIM.

Sie können im Rahmen der Filterbedingung nicht nur auf den aktuellen Datensatz zugreifen, sondern auch zuvor definierte Relationen verwenden. Hierzu stehen die Schlüsselwörter `Child` und `Parent` zur Verfügung. Gefolgt von einem Punkt und dem gewünschten Feldnamen können Sie die verknüpften Daten in die Bedingung einbeziehen.

Im folgenden Beispiel stehen neben der Buchtabelle auch wieder die entsprechenden Autoren zur Verfügung. Beide Tabellen sind relational verknüpft. Über die vorgegebene Filterbedingung werden ausschließlich Bücher angezeigt, deren Titel länger als 15 Zeichen ist und bei denen der Vorname des Autors mit einem „H“ beginnt.

Listing 8.39 RowFilter2.aspx

```
...

DataTable authors = dataset.Tables["Authors"];
DataColumn id = authors.Columns["ID"];

DataTable books = dataset.Tables["Books"];
DataColumn authorid = books.Columns["AuthorID"];

DataRelation relation = new DataRelation("Authors2Books",
    id, authorid);
dataset.Relations.Add(relation);

books.Columns.Add(new DataColumn("Author", typeof(string),
    "Parent.Firstname + ' ' + Parent.Lastname"));
books.Columns.Add(new DataColumn("Titellänge", typeof(int),
    "LEN(Title)"));
books.DefaultView.RowFilter = tb_filter.Text;

dg.DataSource = books;
DataBind();
}

void bt_click(object sender, EventArgs e)
{
    BindDataGrid();
}

</script>
```

```

<form runat="server">

    <p> Filter:
        <ASP:TextBox id="tb_filter" runat="server"
            text="Parent.Firstname LIKE 'H%' AND LEN(Title)>15" width="300"/>
        <ASP:Button runat="server" text="Aktualisieren" onclick="bt_click"/>
    </p>

    <ASP:DataGrid id="dg" runat="server"/>

</form>

```

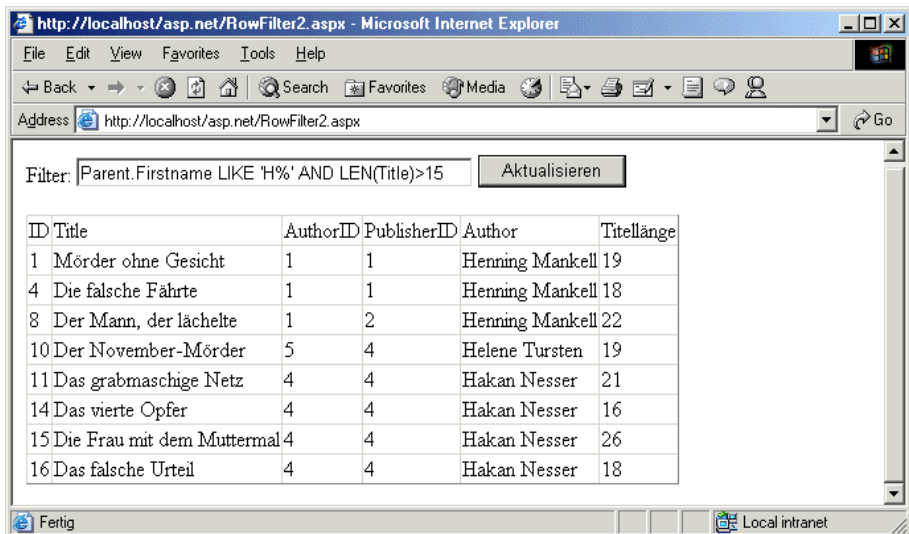


Abbildung 8.18 Auch relational verknüpfte Daten können in den Filter einbezogen werden.

Die Abbildung zeigt das Ergebnis der Filterung. Zur Kontrolle wurden zwei dynamische Spalten mit dem Autorennamen sowie der Länge des Buchtitels angelegt.

Besonders interessant ist der Zugriff auf die relational verknüpften Daten übrigens in Verbindung mit Aggregatfunktionen. Das folgende Listing zeigt den Einsatz von Aggregatfunktionen. Ausgegeben wird diesmal die Liste der Autoren.

Listing 8.40 RowFilter3.aspx

```

DataRelation relation = new DataRelation("Authors2Books",
    id, authorid);
dataset.Relations.Add(relation);

authors.Columns.Add(new DataColumn("Buchanzahl", typeof(int),

```

```
"Count(Child.Title)"));
authors.DefaultView.RowFilter = tb_filter.Text;

dg.DataSource = authors;
DataBind();
}

void bt_click(object sender, EventArgs e)
{
    BindDataGrid();
}

</script>

<form runat="server">

    <p> Filter:
    <ASP:TextBox id="tb_filter" runat="server"
        text="COUNT(Child.Title)>=2" width="300"/>
    <ASP:Button runat="server" text="Aktualisieren" onclick="bt_click"/>
    </p>

    <ASP:DataGrid id="dg" runat="server"/>

</form>
```

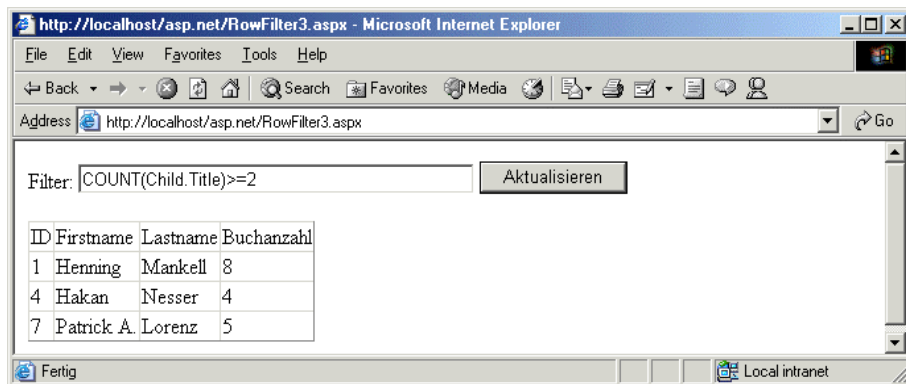


Abbildung 8.19 Die anderen Autoren haben weniger als zwei Bücher in der Datenbank.

Die vorgelegte Filterbedingung verwendet die Aggregatfunktion `COUNT` in Verbindung mit den untergeordneten Datensätzen der Buchtabelle. Es werden nur die Damen und Herren angezeigt, von denen mindestens zwei oder mehr Buchtitel in der Datenbank hinterlegt sind. Auch hier ist eine Kontrollspalte vorhanden.



Neben COUNT können Sie auch weitere Aggregatfunktionen wie SUM, AVG, MIN, MAX, STDEV und VAR benutzen.

8.19 ... ein Memo-Feld im Browser darstellen?

Die allermeisten Datenbanksysteme verfügen über ein Bemerkungsfeld. Im Unterschied zu den mit 256 Zeichen stark begrenzten Textfeldern lassen sich hier beliebig oder zumindest nahezu beliebig lange Zeichenketten ablegen. Diese Möglichkeit wird bei der Web-Entwicklung oftmals für Beschreibungen, Beiträge, Artikel und vieles mehr genutzt. Die Texte liegen meistens unformatiert und somit ohne jegliche HTML-Zeichen vor. Das ist gut, denn es ermöglicht eine individuelle Aufbereitung. Allerdings bringt dieses System auch einen kleinen Nachteil mit sich.

Die Abbildung zeigt den Tabellendialog von Access. Die Tabelle `Memotest` enthält ein einziges Feld `Memo` vom gleichnamigen Typ. Ganz offensichtlich hat sich jemand die Mühe gemacht und einen kleinen Text eingetragen.

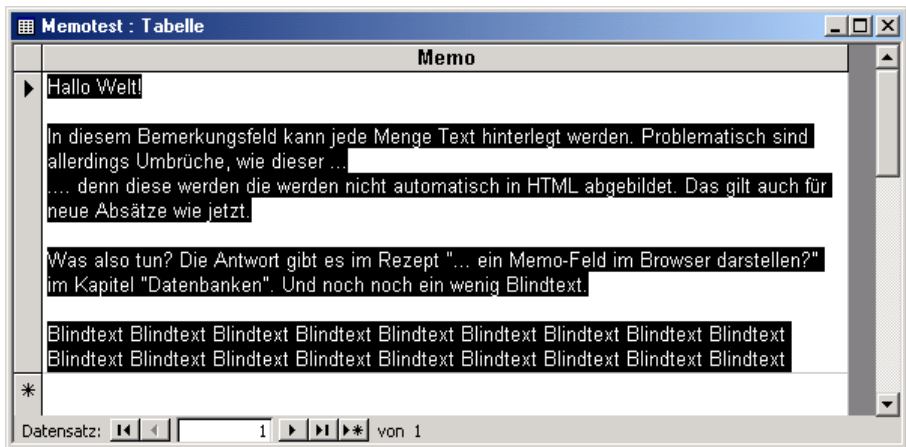


Abbildung 8.20 Die Datenbank enthält den reinen Text ohne HTML-Formatierungen.

Der gezeigte Text soll nun im Browser dargestellt werden. Kein Problem, die Zugriffsklassen von ADO.NET regeln das mit wenigen Zeilen. Aufgrund der einfachen Struktur des Beispiels und der Datenbank habe ich mich für einen Zugriff per `OleDbDataReader` entschieden.

Listing 8.41 memo1.aspx

```
<script runat="server">

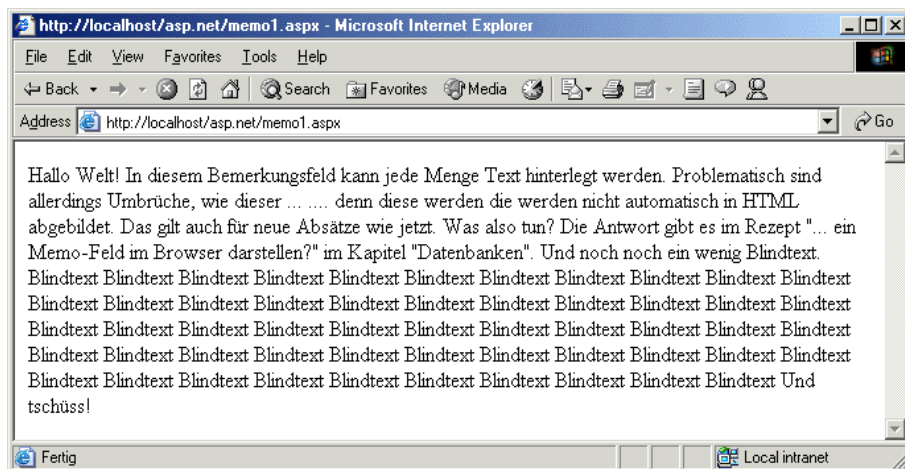
void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Memotest;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    OleDbDataReader reader = cmd.ExecuteReader();
    reader.Read();

    string memo = (string) reader["memo"];
    Response.Write("<p>");
    Response.Write(memo);
    Response.Write("</p>");

    reader.Close();
    conn.Close();
}

</script>
```

**Abbildung 8.21** Die Umbrüche werden nicht dargestellt.

Die Abbildung zeigt die Misere und damit auch den Grund für dieses Rezept. Die hinterlegten Umbrüche werden nicht angezeigt. Kein Wunder, denn HTML ignoriert die regulären Umbruchszeichen und verlangt zur Darstellung von Absätzen nach HTML-Tags wie `p` und `br`.

Statt lange mit HTML zu diskutieren (was ohnehin zwecklos wäre), können Sie dem Wunsch mit Hilfe zweier Replace-Aufrufe nachkommen. Wie im Listing zu sehen, werden zunächst alle doppelten Umbrüche durch „`</p><p>`“ und *anschließend* alle einfachen durch „`<br>`“ ersetzt. Wie in der Abbildung zu erkennen, werden die Umbrüche anschließend korrekt dargestellt.

Listing 8.42 memo2.aspx

```
...

string memo = (string) reader["memo"];

memo = memo.Replace("\r\n\r\n", "</p><p>");
memo = memo.Replace("\r\n", "<br>");

Response.Write("<p>");
Response.Write(memo);
Response.Write("</p>");

...
```

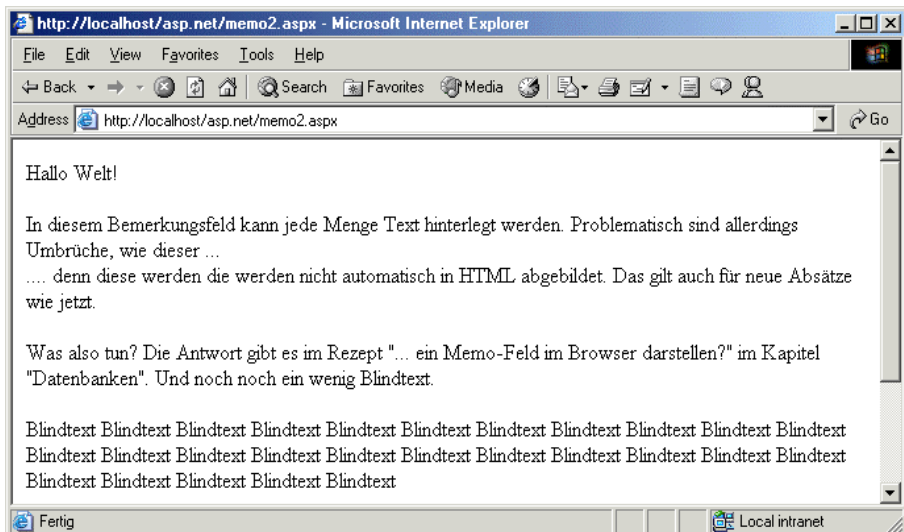


Abbildung 8.22 Dank Ersetzung werden die Umbrüche nun angezeigt.

Sofern Sie mit Visual Basic .NET arbeiten, sieht das Listing ein wenig anders aus. Dort existieren die gezeigten Escape-Sequenzen zur Markierung von Umbrüchen nicht. Hier muss die Konstante `vbCrLf` genutzt werden.

Listing 8.43 memo2_vb.aspx

```
<script runat="server">

Sub Page_Load(Sender as Object, E As EventArgs)

    Dim conn As OleDbConnection = _
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=c:\inetpub\wwwroot\aspnet\misc.mdb")
    conn.Open()

    Dim SQL As String = "SELECT * FROM Memotest;"
    Dim cmd As OleDbCommand = new OleDbCommand(SQL, conn)
    Dim reader As OleDbDataReader = cmd.ExecuteReader()
    reader.Read()

    Dim memo As String = CStr(reader("memo"))

    memo = memo.Replace(vbCRLF + vbCrLf, "</p><p>")
    memo = memo.Replace(vbCRLF, "<br>")

    Response.Write("<p>")
    Response.Write(memo)
    Response.Write("</p>")

    reader.Close()
    conn.Close()

End Sub

</script>
```

8.20 ... einen zufälligen Datensatz abfragen?

Nicht selten steht man vor dem Problem, einen zufälligen Datensatz aus einer Datenbank abzufragen. Ein Beispiel hierfür ist die Ausgabe eines Zufallslinks. Mit Hilfe des SQL-Servers ist die Abfrage nur eine Query entfernt.

Das Listing zeigt eine derartige Auswahl eines zufälligen Links, wie er bei vielen Internet-Angeboten zum Einsatz kommt. Basis bildet eine mittels MSDE und Visual Studio .NET angelegte Datenbank `RandomTable`.

Start Page dbo.RandomTable...-NOTEBOOK.test	
ID	Link
202F2096-6E09-4	http://www.p-l.de
{3C16F45D-D580-4	http://www.asp-buch.de
{C71514D7-3CBC-4	http://www.csharp-shortcut.de
{6A86E6A1-15FC-4	http://www.asp.net
*	

Abbildung 8.23 Die Tabelle enthält eine kleine Linksammlung.

Listing 8.44 RandomRecord1.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.SqlClient" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    SqlConnection conn = new SqlConnection("Initial Catalog=test;
    Data Source=PL-NOTEBOOK;Integrated Security=SSPI;");
    conn.Open();

    string SQL = "SELECT TOP 1 Link FROM RandomTable ORDER BY NEWID();";
    SqlCommand cmd = new SqlCommand(SQL, conn);
    hl.NavigateUrl = (string) cmd.ExecuteScalar();

    conn.Close();
}

</script>

<ASP:Hyperlink id="hl" runat="server" text="Zufallslink"/>
```

Die Funktion `NEWID` im `ORDER BY`-Bereich der Query sorgt für eine zufällige Sortierung der Datensätze. Über `TOP 1` wird der erste Datensatz abgefragt. Alternativ könnten – sofern sinnvoll – auch mehrere zufällige Datensätze abgefragt werden, indem die Anzahl entsprechend erhöht wird.

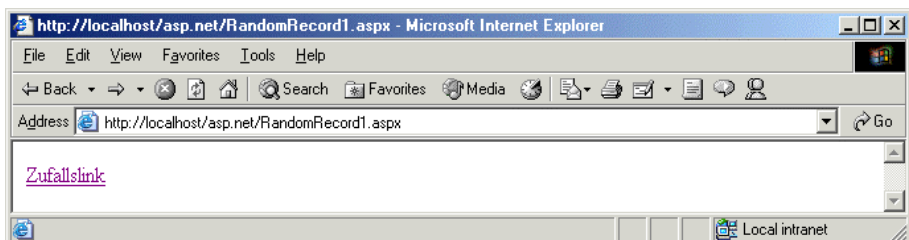


Abbildung 8.24 Bei jedem Aufruf der Seite wird ein zufälliger Link ausgegeben.

Sofern Sie nicht mit dem SQL-Server beziehungsweise der mit Visual Studio .NET ausgelieferten Desktop-Version MSDE, sondern mit Access arbeiten, haben Sie leider schlechte Karten. Eine analoge Funktion `NEWID` existiert leider, die zufällige Abfrage mit Hilfe der Datenbank-Engine ist leider nicht möglich. Sie müssen daher selbst Hand anlegen und eine entsprechende Funktionalität manuell nachrüsten.

Sofern Ihre Datenbank über einen inkrementellen und vor allem konsistent fortlaufenden Primärindex ohne Lücken verfügt, haben Sie leichtes Spiel. In diesem Fall müssen Sie lediglich die Anzahl der Datensätze und über die Random-Klasse einen zufälligen Wert ermitteln. Über diesen Wert können Sie anschließend den zufällig gewählten Datensatz abfragen. Das zweite Beispiel zeigt dies anhand der Büchertabelle der mitgelieferten Beispieldatenbank.

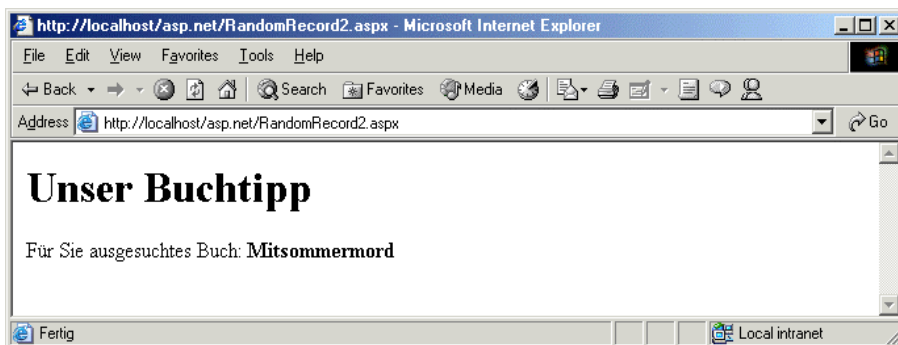


Abbildung 8.25 Bei jedem Aufruf wird ein zufälliger Buchtipp ausgegeben.

Listing 8.45 RandomRecord2.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT COUNT(*) FROM Books;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    int count = (int) cmd.ExecuteScalar();

    Random rnd = new Random();
```

```

int randomID = rnd.Next(1, count);

SQL = string.Format("SELECT Title FROM Books WHERE ID={0}",
    randomID);
cmd = new OleDbCommand(SQL, conn);
lb.Text = (string) cmd.ExecuteScalar();

conn.Close();
}

</script>

<h1> Unser Buchtipps </h1>

<p> Für Sie ausgesuchtes Buch:
    <b> <ASP:Label id="lb" runat="server"/> </b>
</p>

```

Etwas problematischer wird es, wenn Sie nicht sicherstellen können, dass es keine Lücken in Ihrer Datenbank gibt. In diesem Fall haben Sie zwei Möglichkeiten. Zum einen können Sie die Abfrage sicherer machen. Sofern kein Datensatz mit der entsprechenden ID vorhanden ist, wird einfach eine neue Suche gestartet. In diesem Fall können Sie allerdings nicht die Gesamtanzahl der Datensätze zur Ermittlung des zufälligen Wertes verwenden, sondern müssen die niedrigste sowie die höchste ID manuell abfragen.

Listing 8.46 RandomRecord3.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    string SQL = "SELECT MIN(ID) FROM Books;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    int min = (int) cmd.ExecuteScalar();

    SQL = "SELECT MAX(ID) FROM Books;";
    cmd = new OleDbCommand(SQL, conn);
    int max = (int) cmd.ExecuteScalar();

    Random rnd = new Random();

    string booktitle = null;

```

```
while(booktitle == null)
{
    int randomID = rnd.Next(min, max);
    SQL = string.Format("SELECT Title FROM Books WHERE ID={0}",
        randomID);
    cmd = new OleDbCommand(SQL, conn);
    booktitle = (string) cmd.ExecuteScalar();
}

lb.Text = booktitle;

conn.Close();
}

</script>

<h1> Unser Buchtipps </h1>

<p> Für Sie ausgesuchtes Buch:
    <b> <ASP:Label id="lb" runat="server"/> </b>
</p>
```

Variante zwei ist nicht ganz so elegant und dazu auch nicht wirklich performant. Ich demonstriere diese eher der Vollständigkeit halber und rate von einer Verwendung ab. Dieser Ansatz sieht vor, über einen `DataReader` alle ID-Werte abzufragen und in ein dynamisches Array zu kopieren. Über dieses wird anschließend der zufällige Datensatz ermittelt.

Listing 8.47 RandomRecord4.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
    conn.Open();

    ArrayList IDs = new ArrayList();

    string SQL = "SELECT ID FROM Books;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    OleDbDataReader reader = cmd.ExecuteReader();

    while(reader.Read())
        IDs.Add(reader["ID"]);
}
```



```
reader.Close();

Random rnd = new Random();
int randomID = rnd.Next(IDs.Count-1);

SQL = string.Format("SELECT Title FROM Books WHERE ID={0}",
    IDs[randomID]);
cmd = new OleDbCommand(SQL, conn);
lb.Text = (string) cmd.ExecuteScalar();

conn.Close();
}

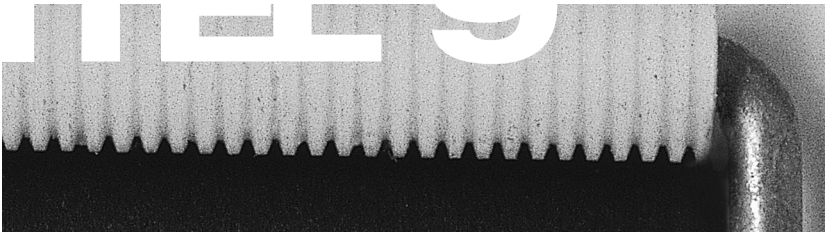
</script>

<h1> Unser Buchtipps </h1>

<p> Für Sie ausgesuchtes Buch:
  <b> <ASP:Label id="lb" runat="server"/> </b>
</p>
```

9

Dateisystem



Wie kann ich ...

9 Dateisystem

Ihr Provider wird es sicher nicht gerne sehen, aber mit ASP.NET können Sie auf einfachste Weise auf der Festplatte Ihr Unwesen treiben. Dies geht natürlich nur dort, wo es Ihnen die aktuellen Benutzerrechte erlauben. Was dabei so alles möglich ist, erfahren Sie hier – aber treiben Sie es nicht zu bunt!

Einbindung

Die Klassen zum Dateizugriff werden bei .NET unterhalb des Namespaces `System.IO` angesammelt. Bevor Sie die Klassen aus den folgenden Rezepten nutzen können, müssen Sie in aller Regel den Namespace zunächst einbinden:

```
<% @Import Namespace="System.IO" %>
```

9.1 ... einen virtuellen Pfad in einen physikalischen umwandeln?

Nichts einfacher als das. Übergeben Sie der Methode `Server.MapPath` den gewünschten virtuellen Pfad, und Sie erhalten das physikalische Gegenstück zurück.

Listing 9.1 MapPath1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write(Server.MapPath(Request.Path));
}

</script>
```

9.2 ... das aktuelle physikalische Verzeichnis ermitteln?

Die Methode `Server.MapPath` ermöglicht das Umwandeln eines virtuellen in einen physikalischen Pfad. Oftmals wird jedoch das physikalische Verzeichnis der aktuellen Seite benötigt. Hier kann die Methode `Path.GetDirectoryName` auf den von `Request.PhysicalPath` zurückgelieferten Pfad angewandt werden. Soll hingegen das physikalische Hauptverzeichnis der aktuellen Web-Applikation erfragt werden, verwenden Sie direkt die Eigenschaft `Request.PhysicalApplicationPath`.

Listing 9.2 PhysicalPath1.aspx

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string directory = Path.GetDirectoryName(Request.PhysicalPath);
    Response.Write("Aktuelles Verzeichnis: ");
    Response.Write(directory + "<br>");

    Response.Write("Verzeichnis der Applikation: ");
    Response.Write(Request.PhysicalApplicationPath);
}

</script>
```

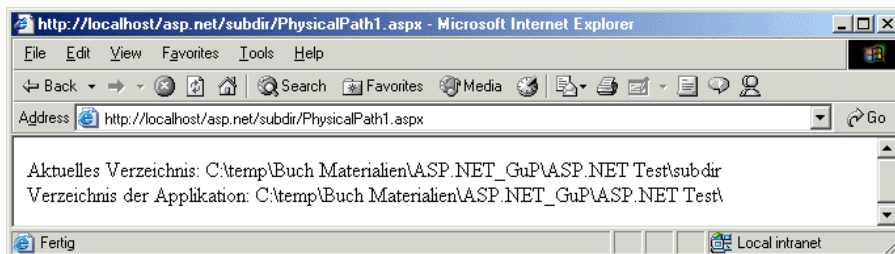


Abbildung 9.1 Das physikalische Verzeichnis lässt sich ganz einfach ermitteln.

9.3 ... eine Verzeichnisliste erstellen?

Die Klasse `Directory` bietet mittels statischer Methoden Zugriff auf eine Reihe von Verzeichnisinformationen. Um eine Liste von Unterverzeichnissen zu erhalten, übergeben Sie beispielsweise der Methode `GetDirectories` den Pfad des übergeordneten Verzeichnisses. Das Resultat ist ein `string`-Array mit den Pfaden der Unterordner, das sich beispielsweise als Datenquelle für ein `DataList`-Control verwenden lässt.

Listing 9.3 Directory1.aspx

```
<script runat="server">

string basedir;

void Page_Load(object sender, EventArgs e)
{
    basedir = "c:\\\\";
    dir.DataSource = Directory.GetDirectories(basedir);
    DataBind();
}

</script>

<h2> Verzeichnisse unter <%= basedir %> </h2>

<ASP:DataList id="dir" runat="server">
    <ItemTemplate>
         <%= Container.DataItem %>
    </ItemTemplate>
</ASP:DataList>
```

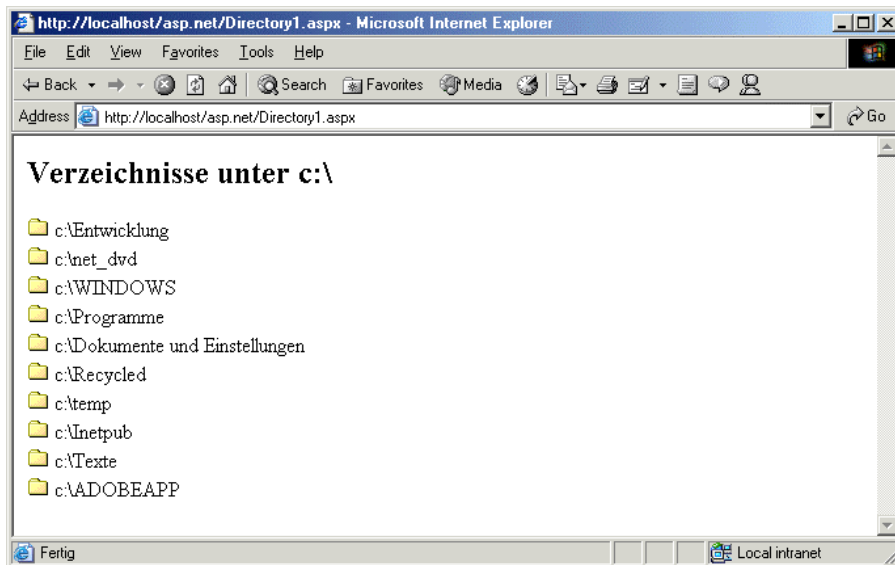


Abbildung 9.2 Eine einfache Auflistung der Verzeichnisse unter c:\

Mit wenigen Handgriffen lässt sich das beispielsweise um eine vollständig funktionierende Verzeichnisnavigation erweitern. Hierzu braucht es lediglich eines Link-Button-Controls innerhalb des DataList-ItemTemplates sowie einer weiteren Deklaration des Controls außerhalb der DataList. Zwei Behandlungen des Command-Ereignisses sorgen dafür, dass das Basisverzeichnis je nach Auswahl aktualisiert wird. Fertig ist die Verzeichnisnavigation.

Listing 9.4 Directory2.aspx

```
<script runat="server">

string basedir;

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        basedir = "c:\\";
        dir.DataSource = Directory.GetDirectories(basedir);
        DataBind();
    }
}

void up_ItemCommand(object sender, CommandEventArgs e)
{
    basedir = e.CommandArgument.ToString();
    DirectoryInfo parent = Directory.GetParent(basedir);
```

```
if (parent != null) basedir = parent.FullName;
dir.DataSource = Directory.GetDirectories(basedir);
DataBind();
}

void dir_ItemCommand(object sender, DataListCommandEventArgs e)
{
    basedir = e.CommandArgument.ToString();
    dir.DataSource = Directory.GetDirectories(basedir);
    DataBind();
}

</script>

<form runat="server">

    <h2> Verzeichnisse unter <%= basedir %> </h2>

    <p>
        <ASP:LinkButton runat="server"
            CommandArgument="<%= basedir %>"
            OnCommand="up_ItemCommand"
            Text="Ebene höher ..."/>
    </p>

    <ASP:DataList id="dir" runat="server"
        OnItemCommand="dir_ItemCommand">

        <ItemTemplate>
            
            <ASP:LinkButton runat="server"
                CommandArgument="<%= Container.DataItem %>"
                Text="<%= Container.DataItem %>"/>
        </ItemTemplate>

    </ASP:DataList>

</form>
```

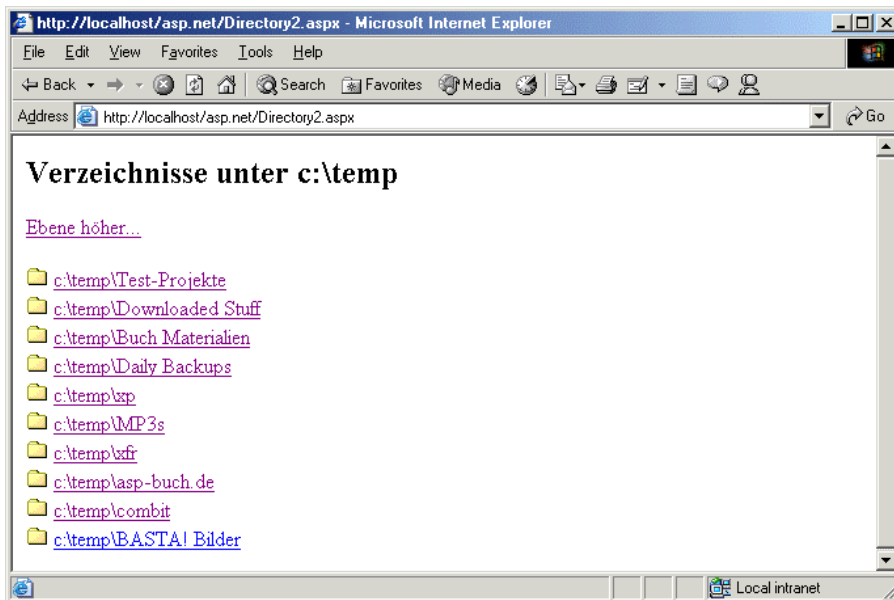


Abbildung 9.3 Ein einfacher Directory-Browser mit wenigen Zeilen Code



Da der Benutzer-Account „ASPNET“ standardmäßig über sehr eingeschränkte Rechte verfügt, müssen Sie diese möglicherweise noch auf Windows ACL-Ebene (Access Control List) erweitern, um das Beispiel ausprobieren zu können.

9.4 ... alle vorhandenen Dateien anzeigen?

Analog zu der Methode `GetDirectories` bietet die Klasse `Directory` eine Methode `GetFiles` an. Diese liefert ein `string-Array` mit den Dateinamen aller im übergebenen Verzeichnis befindlichen Dateien zurück.

Das Listing zeigt die Ergänzung des eben beschriebenen Verzeichnis-Browsers. An ein zweites `DataList-Control` wird das zurückerhaltene Array gebunden. Hier werden nun alle Dateien im aktuellen Verzeichnis gelistet.

Listing 9.5 File1.aspx

```
<script runat="server">

string basedir;

void Page_Load(object sender, EventArgs e)
{
```



```

if(!IsPostBack)
{
    basedir = "c:\\";
    dir.DataSource = Directory.GetDirectories(basedir);
    file.DataSource = Directory.GetFiles(basedir);
    DataBind();
}
}

void up_ItemCommand(object sender, CommandEventArgs e)
{
    basedir = e.CommandArgument.ToString();
    DirectoryInfo parent = Directory.GetParent(basedir);
    if(parent != null) basedir = parent.FullName;
    dir.DataSource = Directory.GetDirectories(basedir);
    file.DataSource = Directory.GetFiles(basedir);
    DataBind();
}

void dir_ItemCommand(object sender, DataListCommandEventArgs e)
{
    basedir = e.CommandArgument.ToString();
    dir.DataSource = Directory.GetDirectories(basedir);
    file.DataSource = Directory.GetFiles(basedir);
    DataBind();
}

</script>

<form runat="server">

    <h2> Verzeichnisse unter <%= basedir %> </h2>

    <p>
        <ASP:LinkButton runat="server"
            CommandArgument="<%= basedir %>"
            OnCommand="up_ItemCommand"
            Text="Ebene höher ..."/>
    </p>

    <ASP:DataList id="dir" runat="server"
        OnItemCommand="dir_ItemCommand">

        <ItemTemplate>
            
            <ASP:LinkButton runat="server"
                CommandArgument="<%= Container.DataItem %>"
                Text="<%= Container.DataItem %>"/>
        </ItemTemplate>
    </ASP:DataList>

```

```
</ASP:DataList>

<ASP:DataList id="file" runat="server">

    <ItemTemplate>
        
        <%# Container.DataItem %>
    </ItemTemplate>

</ASP:DataList>

</form>
```

Natürlich bietet auch die Instanzklasse `DirectoryInfo` eine analoge Methode an. Hier wird jedoch ein Array der Klasse `FileInfo` zurückgeliefert. Dieses lässt sich entsprechend verwenden, bedeutet durch die mehrfach notwendige Klasseninstanziierung aber einen höheren Overhead.

In der Abbildung können Sie erkennen, dass neben den Dateinamen auch das entsprechende Icon ausgegeben wurde. Das Listing zeigt, dass dieses Bild von einer ASP.NET-Seite mit dem Namen `geticon.aspx` geliefert wird. Als Parameter erhält die Seite den gewünschten Dateinamen. Intern verwendet die Seite die Win32 API-Funktion `ExtractAssociatedIcon`, um das zugeordnete Icon zu ermitteln. Zurückgeliefert wird `Handle`, über das mit Hilfe der GDI+-Methoden des Namespace `System.Drawing` das Bild im Browser verkleinert ausgegeben wird.

Listing 9.6 GetIcon.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string filename = Request.QueryString["file"];

    int nIcon = 0;
    IntPtr hIcon = ExtractAssociatedIcon(0, filename, ref nIcon);

    Icon icon = Icon.FromHandle(hIcon);
    Bitmap bitmap = new Bitmap(16, 16);
    Graphics g = Graphics.FromImage(bitmap);
    g.Clear(Color.White);
    g.DrawIcon(icon, new Rectangle(0,0,16,16));

    bitmap.Save(Response.OutputStream, ImageFormat.Gif);
}
```

```
[DllImport("shell32.dll")]
private static extern IntPtr ExtractAssociatedIcon(int hInst, string
filename, ref int nIcon);

</script>
```

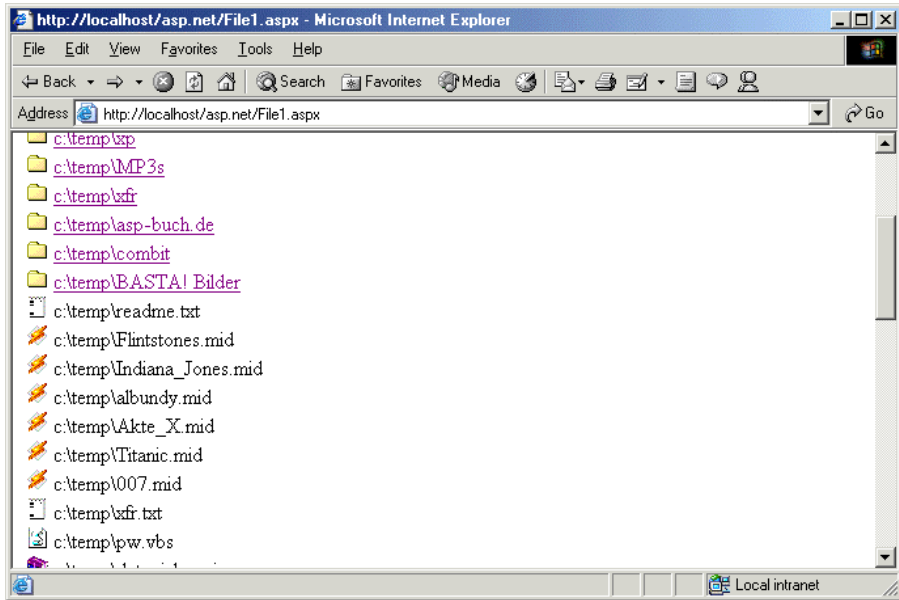


Abbildung 9.4 Alle Dateien im aktuellen Verzeichnis werden samt Icon aufgelistet.



Sofern Sie wie im eben gezeigten Beispiel Win32 API-Funktionen verwenden, benötigen Sie den Namespace `System.Runtime.InteropServices`. Dieser enthält das verwendete Attribut `DllImport`:

```
<% @Import Namespace="System.Runtime.InteropServices" %>
```

9.5 ... mit Wildcards ausgewählte Dateien anzeigen?

Die Methoden `GetFiles` beider Klassen `Directory` und `DirectoryInfo` erlauben dank einer Überladung die Angabe von Wildcards. Zurückgeliefert werden dann nur die diejenigen Dateien in entsprechenden Verzeichnis, auf die das Muster zutrifft. Ich habe das eben gezeigte Listing entsprechend erweitert (`File2.aspx`). Die Abbildung zeigt eine neue Auswahlmöglichkeit im Browser.

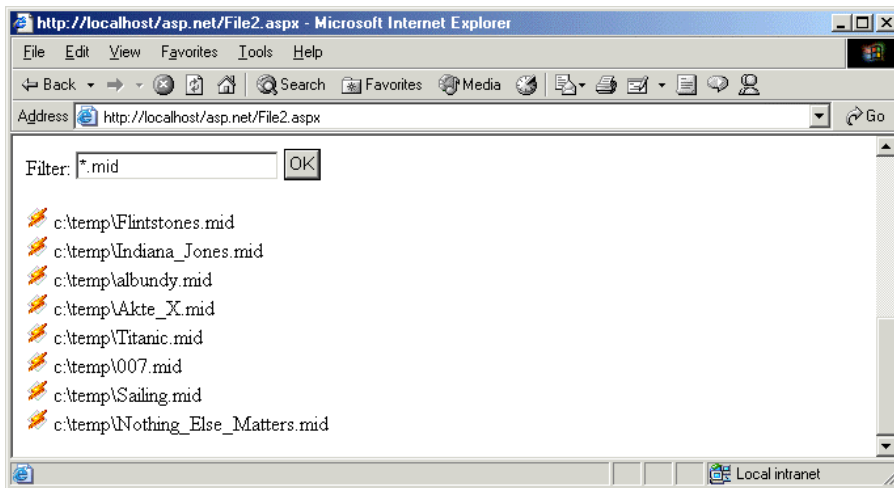


Abbildung 9.5 Dateiauswahl ganz einfach – hier neue Klingeltöne fürs Handy.

9.6 ... eine beliebige Datei zum Client senden?

Nachdem die vorherigen Rezepte die grundlegende Erstellung eines Verzeichnis- und Datei-Browsers gezeigt haben, möchten Sie die angezeigten Dateien vielleicht direkt auf den Client herunterladen können.

Das prinzipielle Problem dabei ist, dass ein direkter Zugriff über eine URL (zum Glück) nicht möglich ist. Um trotzdem eine beliebige Datei an den Client senden zu können, übergeben Sie den Namen einfach an die Methode `Response.WriteFile`. Der gesamte Inhalt der Datei wird anschließend an den Client übertragen.

Ich habe das Beispiel des Verzeichnis- und Datei-Browsers erneut erweitert. Das zweite `DataList`-Control enthält nun für jede angezeigte Datei ein `LinkButton`-Control, über das diese Datei heruntergeladen werden kann. Das Listing zeigt die Behandlung des serverseitigen Ereignisses.

Listing 9.7 File3.aspx

```
...

void file_ItemCommand(object sender, DataListCommandEventArgs e)
{
    string path = e.CommandArgument.ToString();
    string name = Path.GetFileName(path);
    Response.ContentType = "application/x-msdownload;";
```

```

Response.AppendHeader("Content-Disposition",
    string.Format("attachment; filename={0}; alternative: inline",
        name));
Response.WriteFile(path);
Response.End();
}
...

```

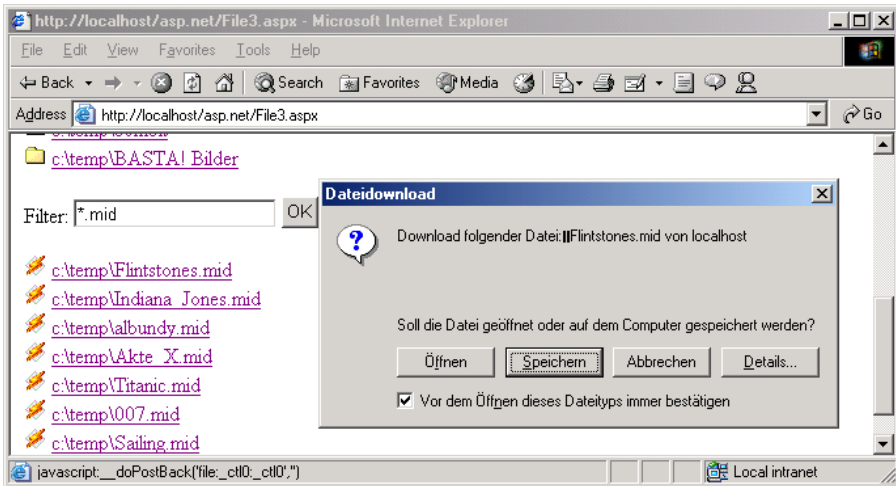


Abbildung 9.6 Der Download beliebiger Dateien ist ganz einfach.

Im Listing wird der Pfad sowie der reine Name der zu übertragenden Datei ermittelt. Damit der Client nicht versucht, die Datei im Browser anzuzeigen, müssen die beiden Kopfzeileinträge `ContentType` und `Content-Disposition` manuell gesetzt werden. Hier wird auch der Dateiname angegeben, der vom Browser im Speichern-Dialog verwendet werden soll. Die Abbildung zeigt das gelungene Ergebnis.

9.7 ... überprüfen, ob ein Verzeichnis oder eine Datei existiert?

Auch die Überprüfung auf die Existenz eines Verzeichnisses beziehungsweise einer Datei kann auf jeweils zwei Wegen durchgeführt werden. Entweder über die statische oder die Instanz-Variante. Die Klassen `Directory` und `File` verfügen jeweils über eine statische Methode `Exists`. `DirectoryInfo` und `FileInfo` verfügen hingegen über eine gleichnamige Eigenschaft.

Das Listing zeigt die Existenzprüfung für eine vom Benutzer eingegebene Datei. Zum Einsatz kommt die statische Methode `File.Exists`.

Listing 9.8 Exists1.aspx

```
<script runat="server">

void UpdatePattern(object sender, EventArgs e)
{
    mess.Text = string.Format("Die Datei {0} existiert{1}!",
        tb_file.Text, (File.Exists(tb_file.Text) ? "" : " nicht"));
}

</script>

<form runat="server">

    <p> Datei
        <ASP:TextBox id="tb_file" runat="server" Text="c:\autoexec.bat"/>
        <ASP:Button runat="server" Text="OK" OnClick="UpdatePattern"/>
    </p>

    <ASP:Label id="mess" runat="server"/>

</form>
```

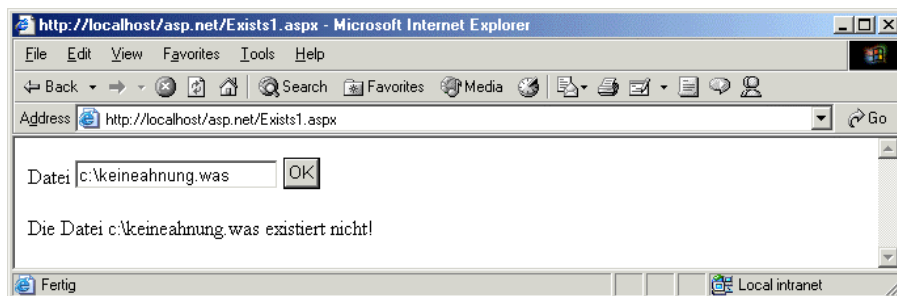


Abbildung 9.7 Die angegebene Datei existiert leider nicht.

9.8 ... eine Datei umbenennen?

Das .NET Framework sieht keine Möglichkeit vor, eine Datei umzubenennen. Es existiert weder eine Methode `Rename` noch kann der neue Name der Eigenschaft `Name` der Klasse `FileInfo` zugewiesen werden. Diese Eigenschaft existiert zwar, verfügt jedoch ausschließlich über einen `get`-Accessor.

Die Gründe für das scheinbare Fehlen liegen in der Win32 API. Auch hier wird alternativ die Datei quasi verschoben. Zum Umbenennen einer Datei verwenden Sie also entweder die Methode `Move` der Klasse `File` oder aber `MoveTo` der Klasse `FileInfo`.

Im folgenden Listing wird die Methode `Move` verwendet, um die aktuelle ASP.NET-Seite umzubenennen. Das ist sicherlich kein echtes Praxisbeispiel, zeigt aber die Verwendung der Methode.

Listing 9.9 Move1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string oldFile = Server.MapPath(Request.Path);
    string newFile = Path.Combine(Path.GetDirectoryName(oldFile),
        "Move2.aspx");

    File.Move(oldFile, newFile);
}

</script>
```

9.9 ... eine Textdatei auslesen?

Die Klassen `File` und `FileInfo` bieten jeweils eine Methode `OpenText`, die es ermöglicht, eine Textdatei zu öffnen und mit Hilfe der zurückgelieferten `StreamReader`-Instanz auszulesen. Das folgende Beispiel gibt auf diese Weise die Datei `autoexec.bat` aus dem Root-Verzeichnis im Browserfenster aus.

Listing 9.10 ReadFile1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    StreamReader reader = File.OpenText("c:\\\\autoexec.bat");
    string contents = reader.ReadToEnd();
    contents = contents.Replace("\r\n", "<br>");
    Response.Write(contents);
}

</script>
```

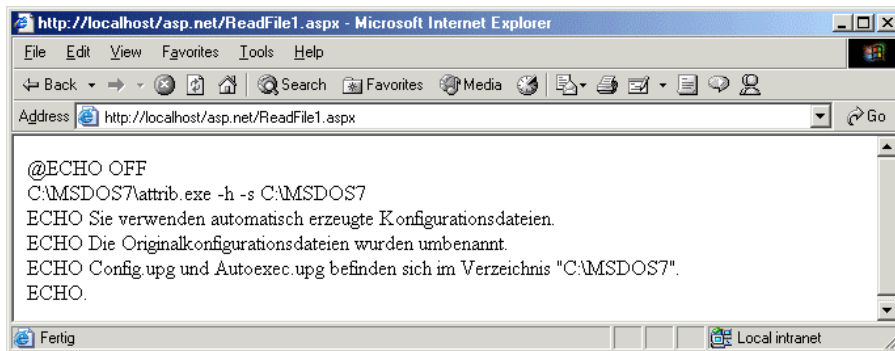


Abbildung 9.8 Die Datei wurde dynamisch ausgelesen.

Eine weitere Möglichkeit, eine Datei auszulesen, bietet der Konstruktor der bereits verwendeten Klasse `StreamReader`. Hier kann der gewünschte Dateiname als Zeichenkette übergeben werden.

```
StreamReader reader = new StreamReader("c:\\autoexec.bat");
```

In beiden vorgestellten Fällen wird das eingestellte Encoding dem System entnommen. Es kann daher sein, dass manche Sonderzeichen beim Öffnen nicht berücksichtigt werden. Sie können daher die Wahl des Encodings der Klasse überlassen. Übergeben Sie dazu einen zusätzlichen booleschen Parameter:

```
StreamReader reader = new StreamReader("c:\\autoexec.bat", true);
```

Möchten Sie das Encoding selbst bestimmen, so nimmt eine weitere Überladung eine Instanz einer von der abstrakten Basis `Encoding` abgeleiteten Klasse entgegen. Mögliche Kandidaten sind beispielsweise `ASCIIEncoding`, `UnicodeEncoding`, `UTF7Encoding` und `UTF8Encoding`.

Der Unterschied lässt sich sehr einfach an einer Datei mit einigen deutschen Umlauten erkennen. Das folgende Listing öffnet eine solche Datei auf die drei vorgestellten Arten. Die Abbildung zeigt, dass nur bei expliziter Wahl des `UTF7`-Encodings die Sonderzeichen ausgelesen und dargestellt werden.

Listing 9.11 `ReadFile2.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Server.MapPath("test.txt");
    StreamReader reader;
    string contents;
```



```
reader = new StreamReader(file);
contents = reader.ReadToEnd();
Response.Write("Standard-Encoding: ");
Response.Write(contents);
Response.Write("<br><br>");

reader = new StreamReader(file, true);
contents = reader.ReadToEnd();
Response.Write("Automatische Erkennung: ");
Response.Write(contents);
Response.Write("<br><br>");

reader = new StreamReader(file, new UTF7Encoding());
contents = reader.ReadToEnd();
Response.Write("UTF8-Encoding: ");
Response.Write(contents);
Response.Write("<br><br>");
}

</script>
```

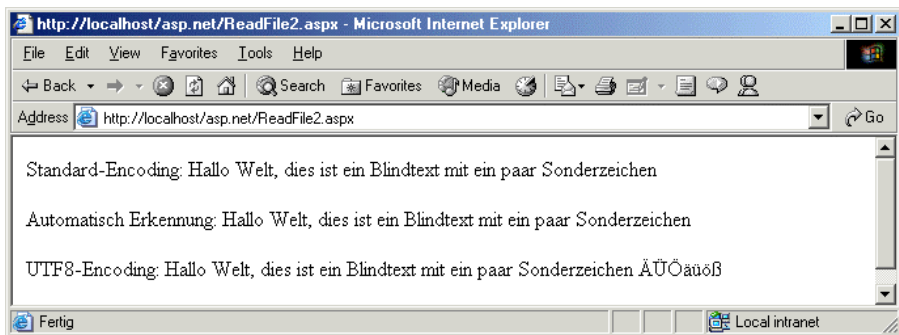


Abbildung 9.9 Die Wahl des Encodings ist nicht zu vernachlässigen.

9.10 ... eine Textdatei erstellen oder ergänzen?

Die im vorherigen Rezept beschriebenen Möglichkeiten zum Auslesen einer Textdatei lassen sich auf das Erstellen und Ergänzen übertragen. Zuständig ist hier die Klasse `StreamWriter`, die mittels der Methoden `CreateText` und `AppendText` instanziiert werden können. Alternativ kann auch hier direkt der Konstruktor der

Klasse benutzt werden. Über einen booleschen Parameter kann angegeben werden, ob die Datei überschrieben oder ergänzt beziehungsweise ganz neu angelegt werden soll. Auch das zu verwendende Encoding kann angegeben werden.

Listing 9.12 WriteFile1.aspx

```
void Page_Load(object sender, EventArgs e)
{
    StreamWriter writer = new StreamWriter("c:\\text.txt", true,
        new UTF7Encoding());
    writer.WriteLine("Hallo Welt");
    writer.Flush();
    writer.Close();
    Response.Write("c:\\text.txt wurde erzeugt ...");
}

</script>
```

9.11 ... eine binäre Datei auslesen?

Binäre Daten werden über den Datentyp `byte` repräsentiert. In der Regel kommt dabei ein `byte`-Array zum Einsatz, schließlich ist ein `Byte` ein bisschenl wenig. Auch beim Zugriff auf eine bestehende binäre Datei wird ein `byte`-Array verwendet.

Vor dem Zugriff muss die Datei zunächst einmal geöffnet werden. Hierzu wird die Methode `OpenRead` der Klassen `File` oder auch `FileInfo` verwendet. Zurückgeliefert wird eine Instanz der Klasse `FileStream`. Diese ist von der abstrakten Basis `Stream` abgeleitet und implementiert unter anderem die üblichen Methoden `Read` und `Write`. Wie der Name schon fast vermuten lässt, dient die Methode `Read` dem lesenden Zugriff auf den Stream. Übergeben wird ein zuvor instanziiertes `byte`-Array als Puffer mit einer frei wählbaren Größe. Typischerweise wird hier ein einstelliger Wert von mehreren Kilobyte gewählt, beispielsweise 4096 für 4 Kilobyte.

Das Listing zeigt das Auslesen einer binären Datei. Innerhalb einer `while`-Schleife wird das `byte`-Array so lange mit Daten gefüllt, bis die Datei komplett ausgelesen wurde. Die Methode `Read` liefert hierbei jeweils die Anzahl der tatsächlich empfangenen Bytes zurück. Dieser Wert wird mitsamt dem Puffer an die Methode `Write` des Ausgabe-Streams übergeben. Bei der Datei handelt es sich um ein Bild, das anschließend im Browser dargestellt wird.

Listing 9.13 Binary1.aspx

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.ContentType = "image/gif";

    string directory = Path.GetDirectoryName(Request.PhysicalPath);
    string filename = Path.Combine(directory, "hollywood-sign.gif");

    FileStream file = File.OpenRead(filename);
    byte[] bytes = new byte[4096];
    int count;
    while((count = file.Read(bytes, 0, bytes.Length)) > 0)
    {
        Response.OutputStream.Write(bytes, 0, count);
    }
}

</script>
```

Die Abbildung zeigt das Ergebnis im Browserfenster. Der Inhalt der Datei wurde korrekt ausgelesen und an den Client übertragen. Dieser stellt das ursprüngliche GIF-Bild dar. Selbstverständlich können Sie die Daten aber auch in einer beliebigen anderen Weise verwenden. Die Möglichkeiten hängen in erster Linie von Ihren Anforderungen und natürlich dem verwendeten Format ab.

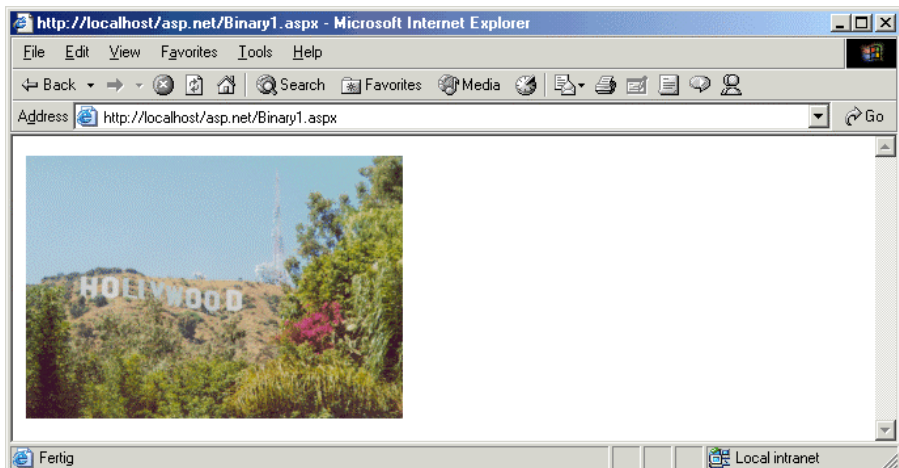


Abbildung 9.10 Die Datei wurde ausgelesen und an den Browser übertragen.



Im Beispiel wird die Methode `OpenRead` zum Öffnen der Datei verwendet. Alternativ können Sie auch die allgemeinere Variante `Open` verwenden; `OpenRead` ist lediglich eine Kurzform und äquivalent zu folgendem Aufruf:

```
FileStream file = File.OpenRead(filename, FileMode.Open,  
    FileAccess.Read, FileShare.Read);
```

9.12 ... eine binäre Datei erstellen oder ergänzen?

So einfach wie das Auslesen einer binären Datei ist auch das Erstellen und Schreiben einer solchen. Sie können hierzu die Methoden `Create` respektive `OpenWrite` verwenden. Analog zum lesenden Zugriff (vergleiche vorheriges Rezept „... eine binäre Datei auslesen?“) wird zur Übertragung der Daten ein `byte`-Array verwendet. Im Listing wird dies genutzt, um eine bestehende in eine neue zu kopieren.

Listing 9.14 Binary2.aspx

```
<% @Import Namespace="System.IO" %>  
  
<script runat="server">  
  
void Page_Load(object sender, EventArgs e)  
{  
    Response.ContentType = "image/gif";  
  
    string directory = Path.GetDirectoryName(Request.PhysicalPath);  
    string sfilename = Path.Combine(directory, "hollywood-sign.gif");  
    string dfilename = Path.Combine(directory, "newpic.gif");  
  
    FileStream sourcefile = File.OpenRead(sfilename);  
    FileStream destfile = File.Create(dfilename);  
  
    byte[] bytes = new byte[4096];  
    int count;  
    while((count = sourcefile.Read(bytes, 0, bytes.Length)) > 0)  
    {  
        destfile.Write(bytes, 0, count);  
    }  
  
    destfile.Close();  
    sourcefile.Close();  
}  
  
</script>
```

Die Verwendung der bestehenden Kopiermethoden ist sicherlich ein ganzes Stück einfacher und vor allem performanter. Dennoch zeigt das Beispiel, wie einfach auch der schreibende Zugriff auf Dateien ist.



Achten Sie unbedingt darauf, dass Sie die neu angelegte oder ergänzte Datei nach dem Schreibzugriff explizit mit Hilfe der Methode `Close` schließen. Ansonsten bleibt diese weiter geöffnet und ist für den Zugriff anderer Prozesse gesperrt.

Die Methode `Open`

Statt der Methoden `Create`, `OpenRead` und `OpenWrite` können Sie auch direkt die allgemeine Variante `Open` verwenden. Bis zu drei zusätzliche Parameter können festlegen, wie auf die Datei zugegriffen werden soll. Es werden die Werte folgender drei Enumerationen übergeben:

- `FileMode` legt fest, wie die Datei geöffnet werden soll. Mögliche Werte sind `Append`, `Create`, `CreateNew`, `Open`, `OpenOrCreate` und `Truncate`.
- `FileAccess` legt die Art des Zugriffs fest. Mögliche Werte sind: `Read`, `ReadWrite` und `Write`. Es handelt sich um eine Flags-Aufzählung, die einzelnen Werte können daher bitweise verodert werden.
- `FileShare` bestimmt, ob und, wenn ja, wie andere Prozesse parallel auf die Datei zugreifen können. Mögliche Werte sind hier `Inheritable`, `None`, `Read`, `ReadWrite` und `Write`. Es handelt sich um eine Flags-Aufzählung, die einzelnen Werte können daher bitweise verodert werden.

Statt die Methode `Open` der Klassen `File` beziehungsweise `FileInfo` können Sie die gewünschten Parameter auch direkt dem Konstruktor der Klasse `FileStream` übergeben.

9.13 ... Änderungen im Dateisystem überwachen?

Sie können Änderungen am Dateisystem automatisch überwachen lassen. Sie kennen dies beispielsweise, wenn Sie mit mehreren Explorer-Fenstern auf ein Verzeichnis zugreifen und eine Datei in einen Fenster löschen – schwups, ist diese auch im zweiten Fenster verschwunden.

Das .NET Framework stellt die Klasse `FileSystemWatcher` zum einfachen Beobachten des Dateisystems zur Verfügung. Wann immer eine Änderung im überwachten Bereich eintritt, wird ein Ereignis ausgelöst. Dies zeigt auch bereits die Problematik bei der Verwendung in ASP.NET auf. Hier steht zunächst kein persistentes

Objekt zur Verfügung, das zur Ereignisbehandlung verwendet werden könnte. Jede einzelne Seite wird nach der Abarbeitung aus dem Speicher geladen. Lediglich das zentrale Application-Objekt steht dauerhaft zur Verfügung.

Sie können die Klasse `FileSystemWatcher` in der zentralen Datei `global.asax` verwenden und dort mit Application-Scope ablegen. Im Listing sehen Sie eine solche Implementierung. Der Watcher wird auf das Hauptverzeichnis „scharf“ gemacht und soll alle Dateiänderungen in einem ebenfalls hinterlegten Ereignis protokollieren. Hier wird der über die Ereignisargumente gelieferte Dateiname in einer `ArrayList` mit Application-Scope abgelegt.

Listing 9.15 global.asax

```
<% @Import Namespace="System.IO" %>

<script language="C#" runat="server">

void Application_OnStart()
{
    FileSystemWatcher watcher = new FileSystemWatcher("c:\\");
    watcher.Changed += new FileSystemEventHandler(OnChanged);
    Application["changes"] = new ArrayList();
    Application["watcher"] = watcher;
    watcher.EnableRaisingEvents = true;
}

void OnChanged(object source, FileSystemEventArgs e)
{
    ArrayList changes = (ArrayList) Application["changes"];
    changes.Add(e.FullPath);
}

</script>
```

Beim nächsten Aufruf einer beliebigen Seite der Web-Applikation wird diese neu gestartet, und der Watcher (quasi ein Mr. Giles fürs Dateisystem) wird aktiv. Alle Änderungen im Hauptverzeichnis `c:\` werden protokolliert. Sie können dies sehr einfach ausprobieren, indem Sie die Datei `autoexec.bat` in den Editor laden und direkt abspeichern. Damit Sie die Änderungen einsehen können, benötigen Sie noch eine zusätzliche Seite, die die protokollierten Einträge anzeigt. Das folgende Listing zeigt eine solche Seite. Die global hinterlegte `ArrayList` wird als Datenquelle für ein `DataList`-Control verwendet. Ein Button erlaubt zudem das Zurücksetzen des Protokolls.

Listing 9.16 ShowChanges1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    dl.DataSource = Application["changes"];
    DataBind();
}

void Reset_Click(object sender, EventArgs e)
{
    ArrayList changes = (ArrayList) Application["changes"];
    changes.Clear();
}

</script>

<form runat="server">

    <h1>Geänderte Dateien:</h1>

    <p>Anzahl: <%=# DataBinder.Eval(Application["changes"], "Count") %>
</p>

    <ASP:DataList id="dl" runat="server">
        <ItemTemplate>
            <%=# Container.DataItem %>
        </ItemTemplate>
    </ASP:DataList>

    <ASP:Button runat="server" text="Zurücksetzen" OnC-
lick="Reset_Click"/>

</form>
```

Die Abbildung zeigt das Ergebnis nach einigen Änderungen. Einige Dateien werden mehrfach angezeigt, da an diesen mehrere Male Änderungen vorgenommen wurden.

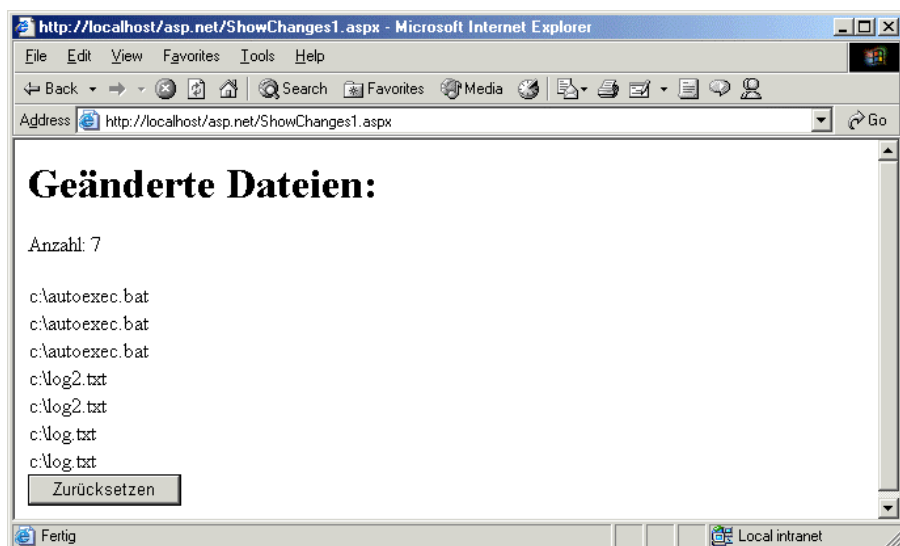


Abbildung 9.11 Alle Änderungen werden protokolliert.

Die Möglichkeiten der Klasse `FileSystemWatcher` gehen weiter als in dem kleinen Beispiel gezeigt. Neben Änderungen an Dateien können Sie über drei weitere Ereignisse auch die Neuanlage, das Löschen und das Umbenennen von Dateien protokollieren. Sie können einen zu überwachenden Pfad angeben und optional auch alle Unterverzeichnisse einschließen. Zudem ist es möglich, die Überwachung auf einen bestimmten Dateifilter wie `*.txt` einzugrenzen.



Leider lassen sich nur (wie auch immer geartete) Änderungen verfolgen. Der reine Lesezugriff oder das Öffnen von Dateien lässt sich nicht nachvollziehen. Zudem liefern die Ereignisargumente keine Informationen über den auslösenden Prozess. Benötigen Sie jedoch weitergehende Informationen, empfehle ich Ihnen das Programm `File Monitor`, das als Freeware von Sysinternals zur Verfügung gestellt wird. Dieses und weitere sehr nützliche Tools können Sie kostenneutral unter folgender Adresse herunterladen:

<http://www.sysinternals.com>

#	Time	Process	Request	Path	Result	Other
9628	14:21:57	Explorer.EXE:724	IRP_MJ_CLEANUP	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	
9629	14:21:57	Explorer.EXE:724	IRP_MJ_CLOSE	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	
9630	14:21:57	Explorer.EXE:724	FSCTL_IS_VOLUME_M...	C:\	SUCCESS	
9631	14:21:57	Explorer.EXE:724	IRP_MJ_CREATE	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	Attributes: N Options: Ope
9632	14:21:57	Explorer.EXE:724	IRP_MJ_QUERY_INFO...	C:\Texte\Veröffentlichungen\Hanser\A...	INVALID P...	FileStreamInformation
9633	14:21:57	Explorer.EXE:724	IRP_MJ_CLEANUP	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	
9634	14:21:57	Explorer.EXE:724	IRP_MJ_CLOSE	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	
9635	14:21:57	Explorer.EXE:724	FSCTL_IS_VOLUME_M...	C:\	SUCCESS	
9636	14:21:57	Explorer.EXE:724	IRP_MJ_CREATE	C:\Texte\Veröffentlichungen\Hanser\A...	SHARING ...	Attributes: N Options: Ope
9637	14:21:57	Explorer.EXE:724	IRP_MJ_CLEANUP	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	
9638	14:21:57	Explorer.EXE:724	IRP_MJ_CLOSE	C:\Texte\Veröffentlichungen\Hanser\A...	SUCCESS	
9639	14:21:57	System:8	IRP_MJ_WRITE*	C:\Programme\Microsoft Works\wklntn...	SUCCESS	Offset: 0 Length: 65536
9640	14:21:57	System:8	IRP_MJ_WRITE*	C:\Programme\Microsoft Works\wklntn...	SUCCESS	Offset: 65536 Length: 65
9641	14:21:57	System:8	IRP_MJ_WRITE*	C:\Programme\Microsoft Works\wklntn...	SUCCESS	Offset: 131072 Length: 6
9642	14:21:57	System:8	IRP_MJ_SET_INFORM...	C:\Programme\Microsoft Works\wklntn...	SUCCESS	FileEndOfFileInformation
9643	14:21:58	System:8	IRP_MJ_WRITE*	C:\Programme\Microsoft Works\wklntn...	SUCCESS	Offset: 196608 Length: 6
9644	14:21:58	System:8	IRP_MJ_WRITE*	C:\Programme\Microsoft Works\wklntn...	SUCCESS	Offset: 262144 Length: 6
9645	14:21:58	System:8	IRP_MJ_SET_INFORM...	C:\Programme\Microsoft Works\wklntn...	SUCCESS	FileEndOfFileInformation

Abbildung 9.12 Der kostenlose File Monitor registriert alle Dateizugriffe.

9.14 ... auf eine ZIP-Datei zugreifen?

Der Entwickler des bekannten C# Editors SharpDevelop, Mike Krüger, hat aus dem offensichtlichen Bedarf heraus eine ZIP-Library von Java auf C# übersetzt. Der Autor stellt die Komponente inklusive Source Code kostenfrei unter der GPL-Lizenz (GNU General Public License) zur Verfügung. Die Lizenz wurde jedoch dahingehend erweitert, dass Projekte mit dieser Komponente selbst nicht unter die GNU-Lizenz fallen. Somit kann die Library auch von kommerziellen Projekten genutzt werden.

NZipLib enthält recht umfangreiche Möglichkeiten zum Zugriff auf ZIP-Dateien und für die Erstellung solcher. Die Integration in das Look and Feel von .NET-Klassen und die allgemeinen Designrichtlinien lassen noch etwas zu wünschen übrig, denn die Herkunft ist deutlich zu erkennen. Dass die Komponente hier auf dem Weg der Besserung ist, lässt eine kürzlich implementierte Klasse `ZipFile` erahnen, die jedoch nicht dokumentiert ist.

Sie können die jeweils aktuelle Version unter folgender Adresse herunterladen:

<http://www.icsharpcode.net>

Dateiinformationen auslesen

Im folgenden Beispiel sehen Sie die einfache Verwendung der Klasse `ZipFile`. Im Konstruktor wird der gewünschte Dateiname oder aber auch ein Stream übergeben. Der Enumerator liefert alle Einträge der ZIP-Datei als Instanzen der Klasse `ZipEntry`. Diese verfügt über zahlreiche Eigenschaften zum Setzen und Abfragen von Attributen wie Name, Größe, Kommentar und so weiter.

Im Listing werden alle Elemente des ZIPs mit einigen Zusatzinformationen im Browser mittels `DataBinding` und einem `DataList-Control` ausgegeben. Um das Beispiel ausführen zu können, müssen Sie die Komponente `NZipLib.dll` in das `bin`-Verzeichnis Ihrer Applikation kopieren.

Listing 9.17 ViewZip1.aspx

```
<% @Import Namespace="NZipLib.Zip" %>

<script runat="server">

ZipFile zipfile;

void Page_Load(object sender, EventArgs e)
{
    string filename = Server.MapPath("test.zip");
    zipfile = new ZipFile(filename);
    dl.DataSource = zipfile;
    DataBind();
    zipfile.Close();
}

</script>

<p> Datei: <%= zipfile.Name %> <br>
    Dateianzahl: <%= zipfile.Size %> </p>

<ASP:DataList id="dl" runat="server">

    <ItemTemplate>
        <h2> <%= DataBinder.Eval(Container.DataItem, "Name") %> </h2>
        <p>
            Größe komprimiert:
            <%= DataBinder.Eval(Container.DataItem, "CompressedSize") %> <br>
            Größe unkomprimiert:
            <%= DataBinder.Eval(Container.DataItem, "Size") %> <br>
            Datum: <%= DataBinder.Eval(Container.DataItem, "DateTime") %>
        </p>
    </ItemTemplate>

</ASP:DataList>
```

In der mir vorliegenden Version enthält der Enumerator einen kleinen Fehler. Der zur Enumeration durch die im ZIP enthaltenen Elemente verwendete Cursor ist mit dem Wert 0 initialisiert. Da bei einer Enumeration vor der Abfrage des ersten Elements die Methode `MoveNext` aufgerufen und hier der Cursor inkrementiert wird, beginnt die Enumeration beim zweiten Element mit Nummer 1 (0-basiert). Da die Komponente samt Quellcode daherkommt, konnte ich den Fehler leicht beseitigen und den Autor informieren.

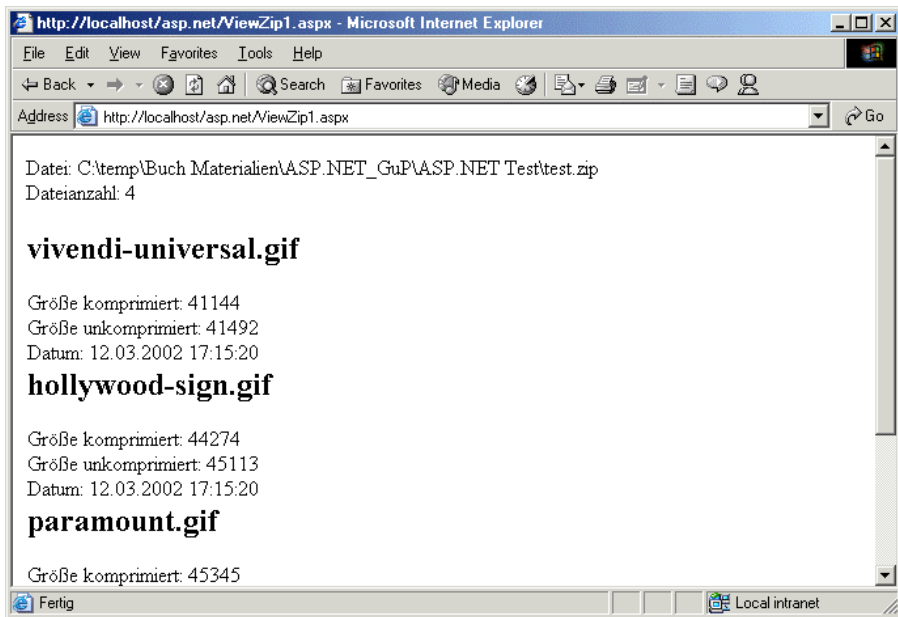


Abbildung 9.13 Alle Einträge der ZIP-Datei werden angezeigt.

Sollte der Fehler in Ihrer Version ebenfalls enthalten sein, können Sie die Änderung sehr einfach nachziehen. Suchen Sie hierzu die Datei `ZipFile.cs` aus dem Verzeichnis `src\NZipLib\Zip`. Suchen Sie in der Datei nach der Klasse `ZipEntryEnumeration`, und ändern Sie den Wert des `int`-Feldes `ptr` auf `-1`.

Listing 9.18 `ZipFile.cs`

```
class ZipEntryEnumeration : IEnumerator {
    ZipEntry[] array;
    int ptr = -1;

    public ZipEntryEnumeration(ZipEntry[] arr) {
        array = arr;
    }
    ...
}
```

Anschließend können Sie die Komponente aus dem übergeordneten Verzeichnis heraus neu kompilieren. Verwenden Sie dazu das Kommandozeilenprogramm `csc.exe` mit folgendem Aufruf:

```
csc /t:library /out:NZipLib.dll /recurse:*.cs
```

Sie erhalten eine neue Datei `NZipLib.dll`, die Sie in das `bin`-Verzeichnis Ihrer Web-Applikation kopieren müssen. Nun können Sie korrekt auf die Elemente zugreifen.

Elemente on the fly komprimieren

Selbstverständlich können Sie auch auf die Inhalte der einzelnen Elemente zugreifen. Die Methode `GetInputStream` liefert auf Basis eines übergebenen `ZipEntry`-Elements einen `Stream` mit den dekomprimierten Daten. Sie müssen sich also gar nicht explizit um die Dekomprimierung bemühen, sondern überlassen dies einfach der Klasse.

Um eine gute Arbeit zu ermöglichen, waren auch hier minimale Anpassungen an der Klasse `ZipFile` notwendig. Ich habe einen Indexer implementiert, der auf Basis des Dateiindexes die entsprechende `ZipEntry`-Instanz liefert. Zudem musste die Methode `GetInputStream` angepasst werden. Diese lieferte bisher einen internen `InflaterInputStream`, der die Komprimierung on the fly durchführt. Daher kann dieser die spätere Länge nicht definitiv angeben. Dies ist jedoch für eine Weiterverarbeitung des Streams unter Umständen wichtig. Die Methode `GetInputStream` kopiert nun den Inhalt in eine `MemoryStream`-Instanz.

Das folgende Beispiel zeigt den Zugriff auf ein zufälliges Element der ZIP-Datei. Da es sich ausschließlich um GIF-Bilder handelt, kann auf Basis des zurückgelieferten Streams eine `Bitmap`-Instanz erstellt werden. Über diese kann das dekomprimierte Bild wie gewohnt im Browserfenster ausgegeben werden.

Listing 9.19 ViewZip2.aspx

```
<% @Import Namespace="System.Drawing" %>
<% @Import Namespace="System.Drawing.Imaging" %>
<% @Import Namespace="System.IO" %>
<% @Import Namespace="NZipLib.Zip" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string filename = Server.MapPath("test.zip");
    ZipFile zipfile = new ZipFile(filename);
```

```
Random rnd = new Random((int)DateTime.Now.Ticks);  
int index = rnd.Next(zipfile.Size);  
Stream stream = zipfile.GetInputStream(zipfile[index]);  
  
Bitmap b = new Bitmap(stream);  
Response.ContentType = "image/gif";  
b.Save(Response.OutputStream, ImageFormat.Gif);  
  
zipfile.Close();  
}  
  
</script>
```

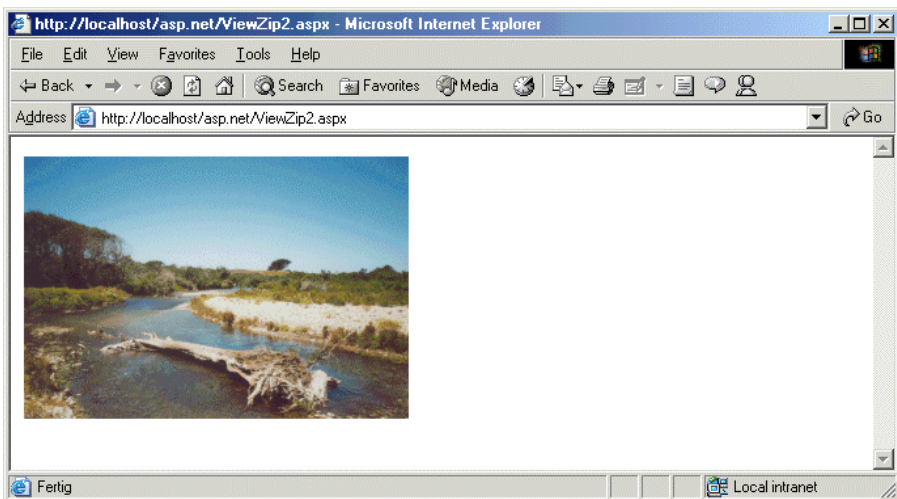


Abbildung 9.14 Das zufällig ausgewählte Bild wurde on the fly dekomprimiert.

ZIP-Dateien komprimieren und abspeichern

Die Verwendung einer ZIP-Komponente auf einem Web-Server hat insbesondere dann Sinn, wenn das Format lediglich dem komprimierten Austausch dient. So können Sie beispielsweise einen einfachen Datei-Upload anbieten. Die enthaltenen Dateien werden automatisch auf dem Server extrahiert und abgespeichert.

Das folgende Listing zeigt einen solchen Einsatz der Komponente. Über ein `HtmlInputFile`-Control kann eine ZIP-Datei auf den Server geladen werden. Hier wird diese geöffnet und durchlaufen. Alle Elemente werden im aktuellen Verzeichnis abgespeichert. Hierzu wird der von `GetInputStream` gelieferte `Stream` in einen `FileStream`, also eine physikalische Datei, kopiert.

Listing 9.20 ExtractZip1.aspx

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="NZlib.Zip" %>

<script runat="server">

void upload_click(object sender, EventArgs e)
{
    HttpPostedFile file = filebox.PostedFile;

    if(file == null)
        message.Text = "Sie müssen eine Datei auswählen";
    else if(Path.GetExtension(file.FileName) != ".zip")
        message.Text = "Sie müssen eine ZIP-Datei auswählen";
    else
    {
        message.Text = "Die folgenden Dateien wurden extrahiert:<br>";

        ZipFile zipfile = new ZipFile(file.InputStream, file.FileName);
        foreach(ZipEntry entry in zipfile)
        {
            message.Text += "&bull; " + entry.Name + "<br>";

            string filename = Server.MapPath(entry.Name);
            FileStream filestream = new FileStream(filename, FileMode.Create);
            Stream inputstream = zipfile.GetInputStream(entry);

            byte[] bytes = new byte[4096];
            int count;
            do
            {
                count = inputstream.Read(bytes, 0, 4096);
                if(count > 0) filestream.Write(bytes, 0, count);
            } while(count > 0);

            filestream.Close();
            inputstream.Close();
        }
    }
}

</script>

<form runat="server" enctype="multipart/form-data">

<p> Bitte wählen Sie eine Datei zum Upload aus: <br/>
<input type="file" id="filebox" runat="server"/>
</p>
```

```
<ASP:Button runat="server" text="Uploaden" onclick="upload_click"/>

<p> <ASP:Label id="message" runat="server"/> </p>

</form>
```

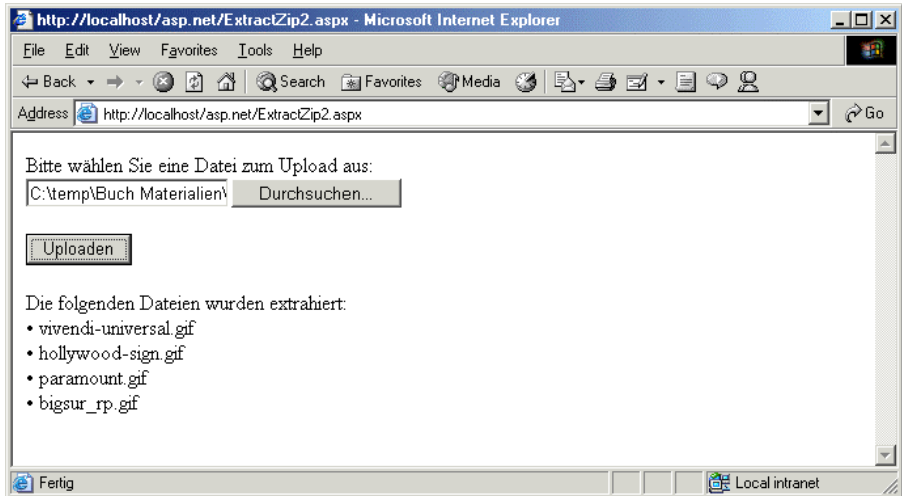


Abbildung 9.15 Die hochgeladene Datei wurde automatisch entpackt.



Vergessen Sie bei einem Upload-Formular bitte nie den Zusatz `enctype="multipart/form-data"` beim serverseitigen `form`-Tag. Ansonsten können Sie die Datei zwar auswählen und hochladen, die `PostedFile`-Eigenschaft liefert dann aber immer `null`.

ZIP-Datei erstellen

Die bisherigen Beispiele haben den Zugriff auf eine bestehende ZIP-Datei gezeigt. Die Komponente ermöglicht jedoch auch die Erstellung einer neuen Datei. Das folgende Beispiel soll genau dies tun. Das Web-Formular ermöglicht über ein List-Box-Control die Auswahl von einem oder mehreren GIF-Bildern im aktuellen Server-Verzeichnis. Ein Klick auf den Button erzeugt eine neue ZIP-Datei und fügt das oder die ausgewählten Bilder hinzu. Das im Speicher des Servers angelegte ZIP wird anschließend an den Client übertragen.

Das Listing zeigt das Eingabeformular. Die Ereignisbehandlung des Buttons legt zunächst eine neue `MemoryStream`-Instanz und auf Basis derer ein `ZipOutputStream`-Objekt an. Die einzelnen Elemente werden nur über `ZipEntry`-Instanzen angefügt. Die Daten müssen jedoch gesondert direkt in den `Stream` geschrieben werden. Auch die Checksumme muss für jedes Element einzeln berechnet zugewiesen werden.

Ist die virtuelle ZIP-Datei erstellt, so wird der darunter liegende `MemoryStream` in den `OutputStream` der Server-Antwort übertragen. Dieser `Stream` kann nicht direkt verwendet werden, da er das von der Komponente benötigte `Seek` (Eigenschaft `CanSeek`) nicht unterstützt. Damit die Datei angezeigt wird, müssen der korrekte `Content-Type` gesetzt und die Daten als `Attachment` gekennzeichnet werden.

Listing 9.21 CreateZIP1.aspx

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="NZlib.Zip" %>
<% @Import Namespace="NZlib.Checksums" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string[] files = Directory.GetFiles(Server.MapPath(""), "*.gif");
    filebox.DataSource = files;
    if(!IsPostBack) DataBind();
}

void create_click(object sender, EventArgs e)
{
    MemoryStream memstream = new MemoryStream();
    ZipOutputStream zipstream = new ZipOutputStream(memstream);
    zipstream.SetLevel(6);
    Crc32 crc = new Crc32();

    foreach(ListItem item in filebox.Items)
    {
        if(item.Selected)
        {
            string filename = item.Value;
            FileStream file = File.OpenRead(filename);

            byte[] bytes = new byte[file.Length];
            file.Read(bytes, 0, bytes.Length);

            ZipEntry entry = new ZipEntry(Path.GetFileName(filename));
            entry.DateTime = File.GetLastWriteTime(filename);
            entry.Size = file.Length;
```



```

    file.Close();

    crc.Reset();
    crc.Update(bytes);
    entry.Crc = crc.Value;

    zipstream.PutNextEntry(entry);
    zipstream.Write(bytes, 0, bytes.Length);
}
}
zipstream.Finish();

Response.ContentType = "application/x-compressed" + ";
    filename=newzipfile.zip";
Response.AppendHeader("Content-Disposition", "attachment;
    filename=newzipfile.zip; alternative: inline");

memstream.Seek(0, SeekOrigin.Begin);
byte[] outbytes = new byte[4096];
int count;
do
{
    count = memstream.Read(outbytes, 0, 4096);
    if(count > 0) Response.OutputStream.Write(outbytes, 0, count);
} while(count > 0);

zipstream.Close();
memstream.Close();
Response.End();
}

</script>

<form runat="server">

    <p> Bitte wählen Sie die Dateien aus, die komprimiert werden sollen:
</p>

    <p> <ASP:Label id="message" runat="server"/> </p>

    <ASP:ListBox id="filebox" runat="server" selectionmode="multiple"/>

    <p>
        <ASP:Button runat="server" text="ZIP erstellen"
            onclick="create_click"/>
    </p>

</form>

```

Das System mutet ein wenig umständlich an. Ich könnte mir vorstellen, dass der Autor für die Zukunft eine Erweiterung der Klasse `ZipFile` plant. Diese könnte wie eine Collection ausgestattet werden und einfache(re) Methoden zum Anfügen der einzelnen Dateien bieten. Dank dem mitgelieferten Quellcode lassen sich solche Änderungen aber gegebenenfalls auch individuell implementieren.

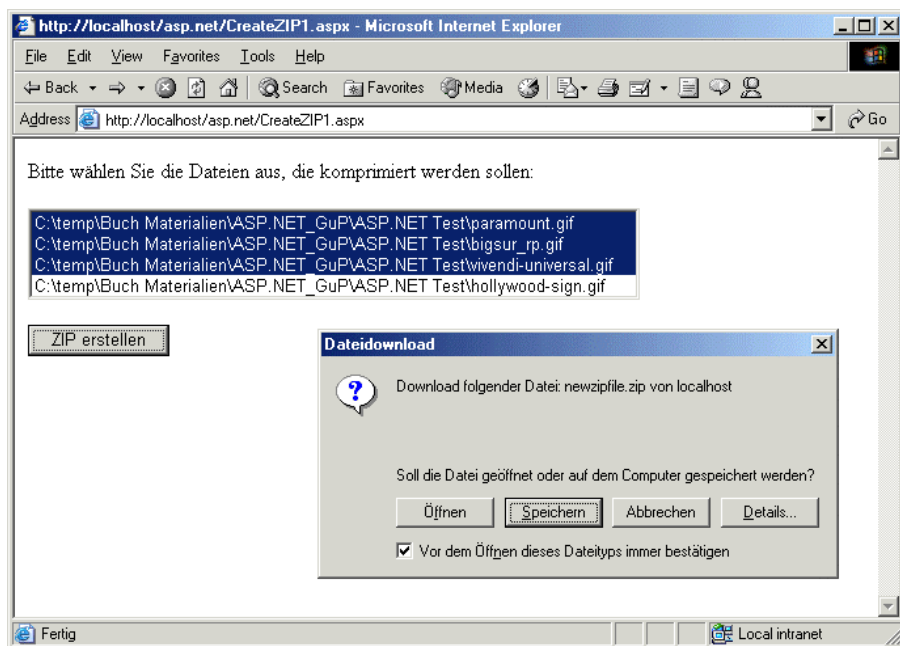


Abbildung 9.16 Die erstellte ZIP-Datei wird direkt an den Client übertragen.

ZIP-Datei erstellen und per Email versenden

Natürlich können Sie eine erstellte ZIP-Datei auch direkt per Email an Ihre Besucher versenden. Das Gleiche gilt etwa auch für individuelle Datei-Uploads Ihrer Kunden und Interessenten. Hier offenbart sich jedoch eine Schwachstelle der Klasse `MailMessage`, die im .NET Framework zum Versenden von Nachrichten verwendet wird. Diese erlaubt zwar das Anhängen von Dateien, sie müssen jedoch physikalisch auf der Festplatte vorhanden sein. Die direkte Angabe eines (Memory-)Streams ist leider nicht möglich.

Ich habe das vorherige Beispiel ein wenig modifiziert. Es wird nun zunächst mit Hilfe der statischen Methode `Path.GetTempFileName` ein temporärer Dateiname erzeugt. Die Endung muss manuell von `.tmp` auf `.zip` geändert werden. Anschließend kann die Datei über einen `FileStream` neu erstellt und an den Konstruktor der

Klasse `ZipOutputStream` übergeben werden. Die Erstellung des ZIP erfolgt nun wie gewohnt, allerdings als Datei und nicht im Speicher. Wurden alle Elemente komprimiert, kann die erzeugte Datei als Attachment an die zu versendende Email angehängt werden. Die Empfängeradresse kann über ein zusätzliches Eingabefeld bestimmt werden. Nach dem Versand wird die temporär auf dem Server erzeugte ZIP-Datei wieder gelöscht.

Listing 9.22 CreateZIP2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string[] files = Directory.GetFiles(Server.MapPath(""), "*.gif");
    filebox.DataSource = files;
    if(!IsPostBack) DataBind();
}

void create_click(object sender, EventArgs e)
{
    string tempfile = Path.GetTempFileName();
    tempfile = tempfile.Replace(".tmp", ".zip");
    FileStream filestream = File.Create(tempfile);

    ZipOutputStream zipstream = new ZipOutputStream(filestream);
    zipstream.SetLevel(6);
    Crc32 crc = new Crc32();

    foreach(ListItem item in filebox.Items)
    {
        ...
    }
    zipstream.Finish();
    zipstream.Close();
    filestream.Close();

    MailMessage mail = new MailMessage();
    mail.From = "pl@p-l.de";
    mail.To = tb_email.Text;
    mail.Subject = "ZIP-Datei";
    mail.Body = "Anbei erhalten Sie die  
gewünschten Daten als ZIP-Datei.";
    mail.Attachments.Add(new MailAttachment(tempfile));
    SmtpMail.Send(mail);

    File.Delete(tempfile);

    message.Text = "Die Dateien wurden Ihnen per Email zugesandt.";
```

```
}  
  
</script>  
  
<form runat="server">  
  
<p> Bitte wählen Sie die Dateien aus, die komprimiert werden sollen:  
</p>  
  
<p> <ASP:Label id="message" runat="server"/> </p>  
  
<ASP:ListBox id="filebox" runat="server" selectionmode="multiple"/>  
  
<p>  
  Email: <ASP:TextBox id="tb_email" runat="server"/>  
</p>  
  
<p>  
  <ASP:Button runat="server" text="ZIP erstellen"  
    onclick="create_click"/>  
</p>  
  
</form>
```

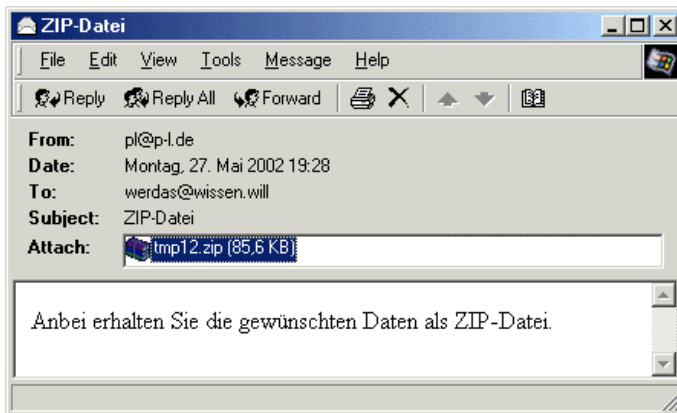


Abbildung 9.17 Die erzeugte ZIP-Datei wurde per Email versendet.

Zusatzinformationen



Die Komponente hat im Rahmen der hier vorgestellten Beispiele einige Änderungen erhalten. Um die Beispiele nachempfinden zu können, benötigen Sie daher die auf der beiliegenden CD-ROM abgelegte Version der Assembly. Diese liegt im Quellcode vor, und Sie müssen sie im Rahmen der GPL-Lizenz entsprechend der readme-Datei distribuieren.



Aus Platzgründen konnten nicht alle Möglichkeiten der Assembly vorgestellt und ausführlich beschrieben werden. Das macht aber nichts, denn mit dem Quellcode werden auch eine umfangreiche Hilfe sowie einige weitere Beispiele des Autors Mike Krüger ausgeliefert.

9.15 ... den kurzen DOS-Dateinamen abfragen?

Das .NET Framework selbst bietet keine eingebaute Möglichkeit, den kurzen DOS-Namen einer Datei abzufragen. Dies erklärt sich aus der Tatsache, dass das Framework plattformunabhängig konzipiert wurde und eine derartige Funktionalität ganz klar auf die Windows-Plattform ausgerichtet ist.

Wird der kurze Dateiname dennoch benötigt, so kann dieser über P-Invoke, also die Win32 API, abgefragt werden. Es wird so auf unmanaged Code im Kernel zugegriffen. Das Listing zeigt die Abfrage des kurzen Dateinamens für die aktuelle Seite mit Hilfe der Funktion `GetShortPathName`. Dieser wird ein Puffer übergeben, der in Form einer Instanz der Klasse `StringBuilder` zur Verfügung gestellt wird.

Listing 9.23 DOSName1.aspx

```
<% @Import Namespace="System.Runtime.InteropServices" %>

<script runat="server">

[DllImport("kernel32.dll")]
static extern int GetShortPathName(string name, StringBuilder buffer,
    int buffersize);

void Page_Load(object sender, EventArgs e)
{
    StringBuilder buffer = new StringBuilder(256);
    GetShortPathName(Request.PhysicalPath, buffer, buffer.Capacity);
    Response.Write(buffer.ToString());
}

</script>
```

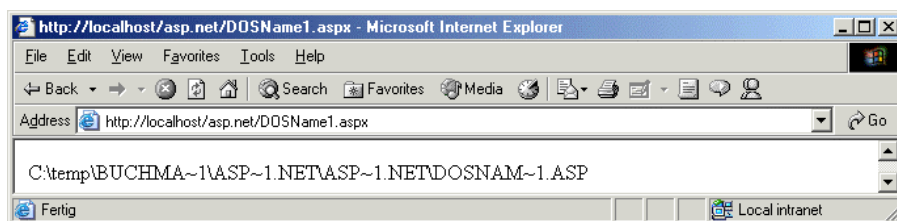


Abbildung 9.18 Der DOS-Dateiname der aktuellen Seite wurde per Win32 API ermittelt.

9.16 ... einen eindeutigen, temporären Dateinamen erstellen?

Wenngleich der Bedarf an temporären Dateien dank der Architektur von ASP.NET eher selten ist, kann es hier und dann doch notwendig sein, Daten kurzfristig auf der Festplatte auszulagern. In diesem Fall gilt es, einen eindeutigen Dateinamen im Temp-Verzeichnis zu erhalten. Die Klasse `Path` aus dem Namespace `System.IO` stellt hierzu eine statische Methode `GetTempFileName` zur Verfügung. Zurückgeliefert wird der eindeutige Name einer temporären Datei, die automatisch mit der Größe 0 – also leer – angelegt wurde.

Das Listing demonstriert die Verwendung der Methode. In die erzeugte Datei wird anschließend ein kurzer Text geschrieben. Zu guter Letzt wird die Datei wieder ordnungsgemäß entfernt.

Listing 9.24 TempFile1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string filename = Path.GetTempFileName();
    StreamWriter writer = File.AppendText(filename);
    writer.Write("Hallo Welt");
    writer.Close();

    Response.Write("Es wurde folgende temporäre Datei angelegt: ");
    Response.Write(filename + "<br>");

    File.Delete(filename);
    Response.Write("Und jetzt wurde die Datei wieder gelöscht.");
}

</script>
```

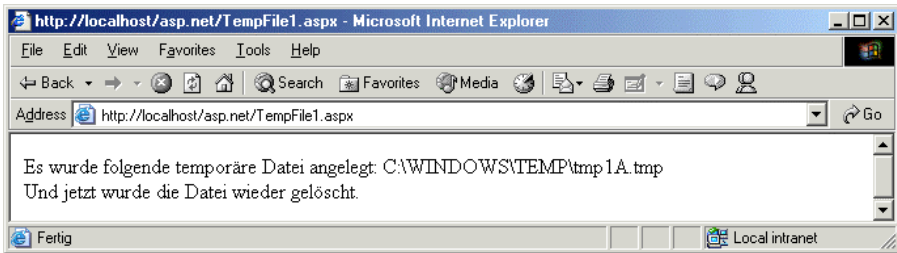


Abbildung 9.19 Die temporäre Datei wurde angelegt und wieder gelöscht.



Bitte denken Sie unbedingt daran, temporär erstellte Dateien wieder zu löschen. Sollte dies nicht direkt möglich sein, so können Sie beispielsweise eine Hashtable mit Session-Scope erstellen und dort die Namen aller angelegten Dateien abspeichern. Wird die Session terminiert, löschen Sie alle Dateien auf einen Schlag.



Beachten Sie bitte des Weiteren, dass der Benutzer-Account „ASPNET“ über die notwendigen Rechte zum Anlegen und Löschen von Dateien im temporären Verzeichnis verfügen muss.

Selbstverständlich können Sie den Namen von temporären Dateien auch selbst bestimmen. Um lediglich den Pfad des dafür vorgesehenen Ordners zu erfragen, verwenden Sie die Methode `Path.GetTempPath`.

Sollten Sie hingegen einen festen Dateinamen verwenden, der unter Umständen von mehreren Besuchern gleichzeitig verwendet wird, so müssen Sie manuell ein temporäres Verzeichnis anlegen. Dies muss über einen eindeutigen Namen verfügen. Hierzu eignet sich ein Hashcode über einen eventuell vorhandenen Benutzernamen oder besser noch die eindeutige Session-ID. Das zweite Listing zeigt dies. Es wird ein temporäres Verzeichnis unterhalb des aktuellen angelegt und anschließend wieder gelöscht.

Listing 9.25 TempFile2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string directory = Path.GetDirectoryName(Request.PhysicalPath);
    directory = Path.Combine(directory, Session.SessionID);
    Directory.CreateDirectory(directory);

    //...
    Directory.Delete(directory, true);
}

</script>
```

9.17 ... einzelne Elemente einer Pfadangabe ermitteln?

Oftmals benötigt man einzelne Elemente eines vollständigen Pfades. Die Klasse `Path` aus dem Namespace `System.IO` stellt hierzu einige statische Methoden zur Verfügung, die statt umständlicher Zeichenkettenoperationen oder übertriebener regulärer Ausdrücke verwendet werden sollten. Die Tabelle zeigt eine Übersicht der Methoden.

Tabelle 9.1 Wichtige Methoden der Klasse `Path`

Methode	Beschreibung
<code>GetDirectoryName</code>	Liefert den kompletten Verzeichnisnamen eines Pfades ohne Dateinamen
<code>GetExtension</code>	Liefert die Dateierweiterung
<code>GetFileName</code>	Liefert den Dateinamen ohne Verzeichnis
<code>GetFileNameWithoutExtension</code>	Liefert den Dateinamen ohne Verzeichnis und ohne Dateierweiterung
<code>GetFullPath</code>	Liefert den vollständigen, absoluten Pfad und wandelt gegebenenfalls einen relativen Pfad um
<code>GetPathRoot</code>	Liefert das Root-Verzeichnis eines Pfades

9.18 ... zwei Pfadelemente verbinden?

Sollen zwei Pfadelemente wie Verzeichnis und Dateiname miteinander verbunden werden, so werden oftmals einfache Zeichenkettenoperationen genutzt. Dies sieht beispielsweise so aus:

```
string path = directory + "\\\" + filename;
```

Das ist ganz offensichtlich nicht sehr schön und vor dem Hintergrund einer plattformunabhängigen Entwicklung regelrecht gefährlich, denn schließlich wird bei Linux der Slash statt des Backslashes verwendet.

Alternativ verwenden Sie besser die Methode `Path.Combine`, die analog zwei Pfadelemente verbindet:

```
string path = Path.Combine(directory, filename);
```


9.19 ... ein Verzeichnis samt Inhalt löschen?

Wenn Sie versuchen, mit Hilfe der Methode `Directory.Delete` oder `DirectoryInfo.Delete` ein Verzeichnis zu löschen, indem sich noch Dateien oder Ordner befinden, so erhalten Sie eine `IOException` mit dem nicht unwahren Hinweis „Das Verzeichnis ist nicht leer“. Das folgende Listing zeigt diesen Effekt.

Listing 9.26 Delete1.aspx

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string directory = Path.GetDirectoryName(Request.PhysicalPath);
    directory = Path.Combine(directory, "test");
    Directory.CreateDirectory(directory);

    StreamWriter writer = File.CreateText(Path.Combine(directory, "test-
file.txt"));
    writer.Close();

    Directory.Delete(directory);
}

</script>
```

Statt das Verzeichnis nun rekursiv zu durchlaufen und alle Dateien und Ordner zu löschen, können Sie einfach eine Überladung der jeweiligen Methode verwenden. Den zusätzlichen booleschen Parameter übergeben Sie als `true`. Anschließend werden auch eventuell vorhandene Dateien und Verzeichnisse unterhalb des Ordners gelöscht.

Listing 9.27 Delete2.aspx

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string directory = Path.GetDirectoryName(Request.PhysicalPath);
    directory = Path.Combine(directory, "test");
    Directory.CreateDirectory(directory);

    StreamWriter writer = File.CreateText(Path.Combine(directory,
"testfile.txt"));
```

```
writer.Close();  
  
Directory.Delete(directory, true);  
}  
  
</script>
```

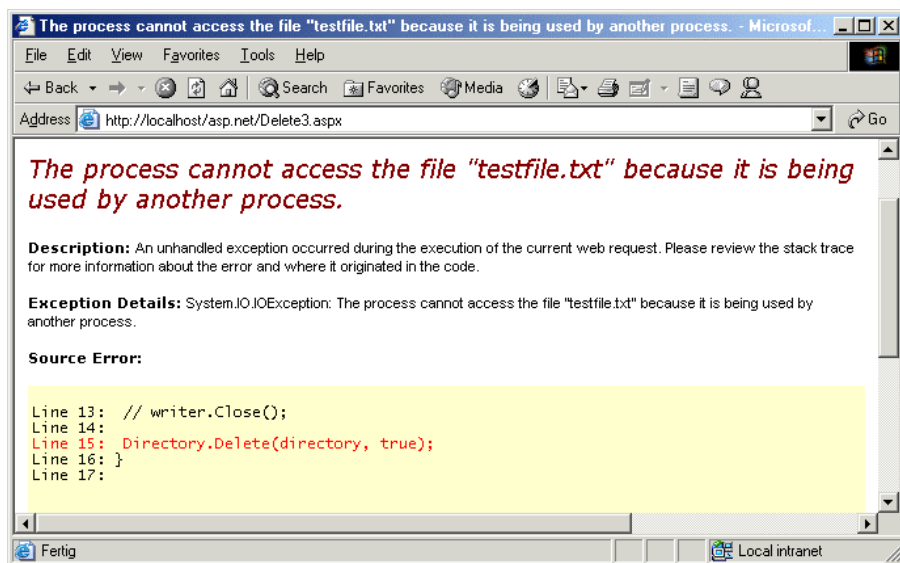


Abbildung 9.20 Eine exklusiv geöffnete Datei kann nicht gelöscht werden.



Möglicherweise erhalten Sie beim Löschen eines Ordners mit Inhalten trotz des zusätzlichen Parameters eine `IOException` mit dem Hinweis „The process cannot access the file ... because it is being used by another process.“ In diesem Fall ist eine Datei noch durch einen anderen oder auch den aktuellen Prozess exklusiv geöffnet und kann daher nicht gelöscht werden. Dies passiert unabsichtlich, wenn Sie beispielsweise das explizite Schließen der Datei über die Methode `Close` vergessen. Sie können diesen Effekt sehr leicht nachvollziehen, indem Sie den Aufruf von `writer.Close()`; im vorherigen Beispiel auskommentieren.

9.20 ... die Version einer Datei ermitteln?

Es ist häufig wichtig zu wissen, welche Version einer bestimmten Datei vorhanden ist. So kann beispielsweise auf verschiedene Installationen und Umgebungen unterschiedlich reagiert werden. Windows bietet hierzu eine ureigene Versionierung von

Dateien an. Diese umfasst neben einer 64-Bit-Versionsnummer (4 x 16 Bit) auch Klartextangaben. Insbesondere DLLs und direkt ausführbare Programme verwenden diesen Mechanismus.

Das .NET Framework bietet mit der Klasse `FileVersionInfo` aus dem Namespace `System.Diagnostics` Zugriff auf die Versionsinformationen einer Datei. Diese wird als Zeichenkette der statischen Methode `GetVersionInfo` übergeben, die eine Instanz der Klasse zurückliefert. Über ganze Reihen von Eigenschaften lassen sich die einzelnen Detailinformationen abfragen. Das Beispiel-Listing zeigt dies.

Listing 9.28 `FileVersionInfo1.aspx`

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="System.Diagnostics" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string sysdir = Environment.SystemDirectory;
    string filename = Path.Combine(sysdir, "kernel32.dll");
    FileVersionInfo version = FileVersionInfo.GetVersionInfo(filename);

    Response.Write("<h1>kernel32.dll</h1>");

    Response.Write("<h2>Version</h2>");
    Response.Write("Major: " +
        version.FileMajorPart.ToString() + "<br>");
    Response.Write("Minor: " +
        version.FileMinorPart.ToString() + "<br>");
    Response.Write("Build: " +
        version.FileBuildPart.ToString() + "<br>");
    Response.Write("Private: " +
        version.FilePrivatePart.ToString() + "<br>");

    Response.Write("<h2>Zusatzinformationen</h2>");
    Response.Write("Beschreibung: " + version.FileDescription + "<br>");
    Response.Write("Produkt: " + version.ProductName + "<br>");
    Response.Write("Copyright: " + version.LegalCopyright + "<br>");
}

</script>
```

In der Abbildung sind einige Versionsangaben der Datei `kernel32.dll` aus dem Windows-Systemverzeichnis zu erkennen. Es handelt sich um eine – wie der Name bereits andeutet – nicht ganz unwichtige Datei des Betriebssystems.



Abbildung 9.21 Die Versionsangaben des installierten Kernels

9.21 ... eine Datei exklusiv sperren?

Wenn Sie eine Datei mit Hilfe der `FileStream`-Klasse öffnen, können andere Prozesse parallel ebenfalls auf diese Datei zugreifen. Sofern Sie schreibend zugreifen, können andere Prozesse die Datei nur lesen. Dennoch ist es oft erwünscht, eine Datei temporär ganz oder teilweise vor dem Zugriff durch andere Prozesse zu sperren. Die Klasse `FileStream` bietet hierzu die Methoden `Lock` und `Unlock` an, die in Kombination verwendet werden. Übergeben wird dabei jeweils die Startposition des zu sperrenden Bereichs und dessen Länge in Byte.

Das Beispiel zeigt das vollständige Locking einer Datei. Hierzu wird als Startwert 0 und als Endwert die komplette Länge der Datei übergeben. Damit anschließend die Datei vollständig entsperrt wird, muss die Methode `Unlock` mit identischen Parametern aufgerufen werden.

Listing 9.29 lock1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    FileStream file = new FileStream(Server.MapPath("paramount.gif"),
    FileMode.Open);
    file.Lock(0, file.Length);

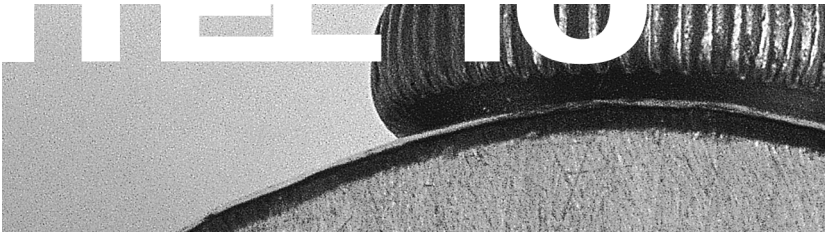
    // Zugriff durch andere Prozesse misslingt

    file.Unlock(0, file.Length);
    file.Close();
}

</script>
```

10

Sessions



Wie kann ich ...

10 Sessions

Das Session-Management ist ein wichtiger Bestandteil der Web-Entwicklung mit ASP.NET. Auf einfache Weise können benutzerrelevante Informationen über die gesamte Dauer eines Besuches vorgehalten werden. Die Anlage und der Zugriff auf Session-Variablen erfolgt über das Session-Dictionary, das über die gleichnamige Eigenschaft `Session` der Klasse `Page` auf jeder Seite zur Verfügung steht.

10.1 ... globale Benutzerinformationen als Session-Variablen realisieren?

Ein prinzipieller Nachteil beim Zugriff auf Session-Variablen ist die Notwendigkeit der Typenkonvertierung. Diese muss durchgeführt werden, da das Session-Dictionary die Daten mit dem Typen `object` zurückliefert. Bestimmte Informationen wie beispielsweise Name und Email-Adresse eines Benutzers oder auch ein Warenkorb sollen auf jeder Seite Ihres Web-Angebots verfügbar sein. Die ständige Typenkonvertierung kann dabei schnell lästig werden.

Die folgenden Zeilen zeigen das Problem; sie liefern einen Laufzeitfehler:

```
int i = 5;
Session["i"] = i;
i = Session["i"];
```

Nach einer zusätzlichen, expliziten Typenkonvertierung läuft das Beispiel problemlos durch:

```
int i = 5;
Session["i"] = i;
i = (int) Session["i"];
```

Damit Sie diese Umwandlung nicht auf jeder Seite durchführen müssen, können Sie den Zugriff auf wichtige Session-Variablen über Eigenschaften in einer zentralen Code Behind-Datei implementieren. Die einzelnen Seiten leiten sich von dieser

Klasse ab und können somit direkt auf die benötigten Daten zugreifen. Auf diese Weise verbergen Sie zusätzlich den Speicherort und trennen so Präsentations- und Geschäftsschicht Ihrer Web-Applikation.

Nachfolgend sehen Sie eine Code Behind-Datei, die eine Eigenschaft `i` mit dem Typen `int` implementiert. Der Zugriff ist dank `get`- und `set`-Accessor lesend und schreibend möglich.

Listing 10.1 SessionVar3.cs

```
using System;
using System.Web;
using System.Web.UI;

public class SessionVar3 : Page
{
    public int i
    {
        get { return((int) Session["i"]); }
        set { Session["i"] = value; }
    }
}
```

Das zweite Listing zeigt die Verwendung der Eigenschaft. Sowohl der schreibende als auch der lesende Zugriff ist direkt und ohne Einschränkung möglich. Selbstverständlich können Sie auch über eine zweite Seite direkt auf den Wert zugreifen, sofern sich diese von der zentralen Code Behind-Datei ableitet. Die Ausgabe dieses Beispiels liefert im Browserfenster die Zahl 5.

Listing 10.2 SessionVar3.aspx

```
<% @Page Language="C#" Debug="true"
    inherits="SessionVar3" src="sessionvar3.cs" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    this.i = 5;
    Response.Write(this.i);
}

</script>
```

Bei einfachen Projekten reicht dieses System aus, bei größeren hat sich jedoch eine erweiterte Version als praktikabel erwiesen. Sie können eine Klasse als Assembly kompilieren und über das `bin`-Verzeichnis global verfügbar machen. Die Klasse

enthält für alle benötigten Session-Variablen separate, statische Eigenschaften, über die Sie die Variablen setzen und die Inhalte auch wieder abfragen können – selbstverständlich immer unter Berücksichtigung des jeweiligen Datentyps.

Das folgende Listing zeigt eine abstrakte Basisklasse `BaseSessionVariables`, die standardisierte Zugriffsmechanismen auf die Session-Variablen bietet. Die Methoden sind analog zu den `DataReader`-Klassen benannt, hier allerdings statisch implementiert. Eine Ableitung der Klasse bietet zwei Session-Variablen `Firstname` und `Lastname` an, die über die Methoden der Basisklasse im Session-Scope abgelegt werden.

Listing 10.3 `SessionVariables.cs`

```
using System;
using System.Web;
using System.Web.SessionState;

public abstract class BaseSessionVariables
{
    public static HttpSessionState Session
    {
        get { return(HttpContext.Current.Session); }
    }

    public static string GetString(string key)
    {
        return((string) Session[key]);
    }

    public static int GetInt32(string key)
    {
        return((int) Session[key]);
    }

    public static double GetDouble(string key)
    {
        return((double) Session[key]);
    }

    public static DateTime GetDateTime(string key)
    {
        return((DateTime) Session[key]);
    }

    public static bool GetBoolean(string key)
    {
        return((bool) Session[key]);
    }
}
```

```
public static void SetValue(string key, object value)
{
    Session[key] = value;
}

public class SessionVariables : BaseSessionVariables
{
    public static string Firstname
    {
        get { return(GetString("firstname")); }
        set { SetValue("firstname", value); }
    }

    public static string Lastname
    {
        get { return(GetString("lastname")); }
        set { SetValue("lastname", value); }
    }
}
```

Ist die Klasse kompiliert und im bin-Verzeichnis abgelegt, können Sie sofort innerhalb jeder Seite auf die statischen Eigenschaften zugreifen und somit implizit den Session-Scope ansprechen. Das folgende Beispiel zeigt dies in Verbindung mit zwei Eingabefeldern. Sind diese einmal gefüllt, behalten Sie während der gesamten Session ihren Inhalt. Dieser wird im Page_Load-Ereignis über die gezeigte Klasse typensicher zugewiesen.

Listing 10.4 SessionVariables1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        tb_fn.Text = SessionVariables.Firstname;
        tb_ln.Text = SessionVariables.Lastname;
    }
}

void bt_click(object sender, EventArgs e)
{
    SessionVariables.Firstname = tb_fn.Text;
    SessionVariables.Lastname = tb_ln.Text;
}

</script>
```

```

<form runat="server">

    <p> Vorname:
    <ASP:TextBox id="tb_fn" runat="server"/>
    </p>

    <p> Nachname:
    <ASP:TextBox id="tb_ln" runat="server"/>
    </p>

    <ASP:Button runat="server" Text="OK" OnClick="bt_click"/>

</form>

```

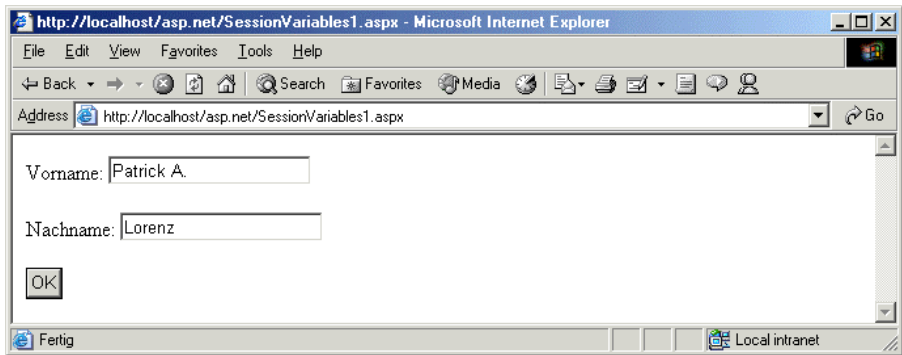


Abbildung 10.1 Nie wieder Typenkonvertierung von Session-Variablen

Es mutet als überflüssige Arbeit an, bei jeder neu eingeführten Session-Variablen die Quellcode-Datei manuell zu erweitern und neu zu kompilieren. Doch diese augenscheinliche Mehrarbeit bringt drei entscheidende Vorteile:

1. Session-Variablen benötigen Platz im Arbeitsspeicher und sollten nicht allzu verschwenderisch genutzt werden. Eine gewisse Hürde bei der Einführung neuer Variablen ist daher nicht als schlechtes Geschäft anzusehen.
2. In der Regel greifen Sie innerhalb Ihres Quellcodes direkt über den vergebenen Namen der Session-Variablen auf diese zu. Sofern Sie nicht mit Konstanten arbeiten, fügen Sie ein und dieselbe Zeichenkette an vielerlei Stellen im Code ein. Dies benötigt einerseits Platz und erhöht andererseits die Wahrscheinlichkeit eines Schreibfehlers an einer der vielen Stellen.
3. Zu guter Letzt sparen Sie bei diesem Ansatz natürlich auch wieder die sonst notwendige Typenkonvertierung.

Alles in allem ist dieses System ausgesprochen leistungsfähig und insbesondere bei großen Web-Applikationen mit zahlreichen Session-Variablen sehr sinnvoll einzusetzen.

Um andere Entwickler in Ihrem Projekt und natürlich auch sich selbst in gewisser Weise zu einer Benutzung dieser globalen Klasse zu bewegen, können Sie sich eines einfachen Tricks in einer globalen Code Behind-Datei bemächtigen. Sie können die Eigenschaft `Session` überschreiben und mit einer `NotImplementedException` ausrüsten. Eine direkte Verwendung über die Eigenschaft der Klasse `Page` ist anschließend nicht mehr möglich.

Listing 10.5 CommonPage.cs

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.SessionState;

public class CommonPage : Page
{
    public override HttpSessionState Session
    {
        get
        {
            throw(new NotImplementedException("Property Session is not
            available, please use global class SessionVariables to
            access session scope."));
        }
    }
}
```

Damit sich diese Änderung auch ja auf alle Seiten auswirkt, können Sie diese kurzer Hand zur neuen Standardvorlage erklären. Ein kleiner Eintrag in der Konfigurationsdatei `web.config` genügt.

Listing 10.6 web.config

```
<configuration>
  <system.web>
    <pages pageBaseType="CommonPage, CommonPage"/>
  </system.web>
</configuration>
```

Wie die Abbildung zeigt, ist nun ein direkter Zugriff auf das Session-Objekt nicht mehr möglich. Ein Umweg beispielsweise über die analoge Eigenschaft der Klasse `HttpContext` ist selbstverständlich weiter möglich.

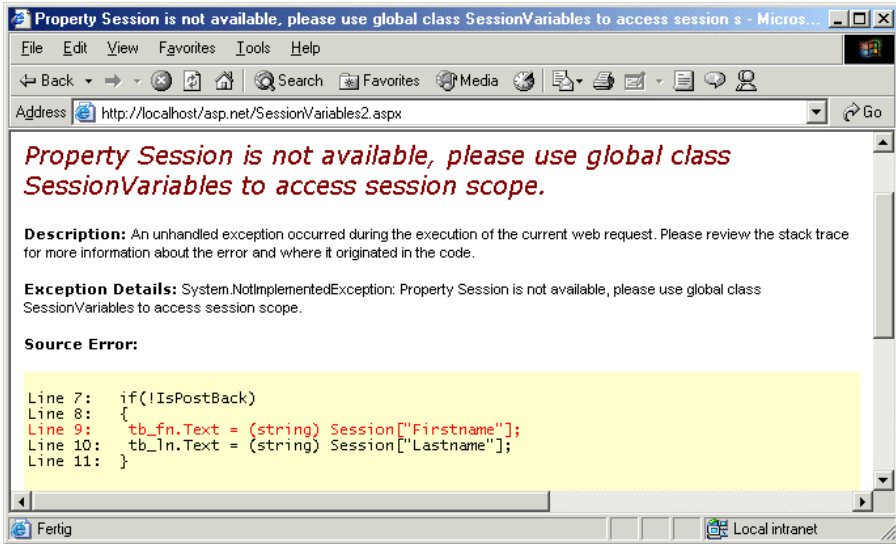


Abbildung 10.2 Die Session-Eigenschaft ist vor dem Zugriff geschützt.

10.2 ... eine Session ohne Cookies erzeugen?

Standardmäßig wird beim ersten Aufruf einer ASP.NET-Seite ein Cookie vom Server an den Client gesendet. Dieser enthält eine verschlüsselte ID, über die der Server spätere Anfragen der Sitzung des Benutzers zuordnen kann. Das Protokoll HTTP sieht hierfür keine eingebauten Mechanismen vor.

Die Verwendung von Cookies ist durchaus nicht unproblematisch, denn viele Benutzer verwenden Firewalls oder Filterprogramme, die die Anlage von Cookies unter Umständen verhindern. Auch lässt sich das Empfangen von Cookies direkt im Browser verhindern oder zumindest einschränken. Insbesondere Version 6.0 des Internet Explorers hat hier begrüßenswerte neue Möglichkeiten für den Benutzer geschaffen.

Die Einschränkung von Cookies ist ein wichtiges Feature für Surfer, für Entwickler jedoch ein ernst zu nehmendes Hindernis bei der Entwicklung mit Sessions. Sie können sich nie darauf verlassen, dass der Benutzer Cookies akzeptiert. Wenn Sie daher das Session-Management verwenden, sollten Sie die Annahme unbedingt zuvor überprüfen. Beachten Sie hierzu das Rezept „... überprüfen, ob Cookies akzeptiert werden?“ im Kapitel „Basics“.

ASP.NET bietet Ihnen ein alternatives System ohne Cookies. Hier wird die ID als virtueller Ordner und somit Teil der Seitenadresse übermittelt. Rufen Sie das erste Mal eine Seite auf, so wird ein Redirect auf den virtuellen Ordner durchgeführt. Die Adresse sieht anschließend beispielsweise so aus:

```
http://localhost/asp.net/(ukisy155a4oki03hjgmbktff)/cookieless1.aspx
```

Bei dem Wert in Klammern handelt es sich um die Session-ID. Damit die cookie-lose Variante verwendet wird, müssen Sie lediglich einen entsprechenden Eintrag in der Konfigurationsdatei `web.config` vornehmen.

Listing 10.7 web.config

```
<configuration>
  <system.web>
    <sessionState cookieless="true"/>
  </system.web>
</configuration>
```



Das Geniale an dieser Variante ist die Tatsache, dass die Arbeit dem Browser überlassen wird. Jeder Link wird automatisch ausgehend vom aktuellen Verzeichnis aufgerufen und enthält somit ohne weiteres Zutun die zur Identifizierung benötigte ID. Bei der Verwendung des cookie-losen Session-Managements müssen Sie daher unbedingt relative und *keinerlei* absolute Verweise verwenden. Diese würden zu einem Verlust der Session-ID und somit der Zuordnung des Benutzers führen.

Falsch:

```
<a href="/asp.net/seite2.aspx">Zur zweiten Seite</a>
```

Richtig:

```
<a href="seite2.aspx">Zur zweiten Seite</a>
```

10.3 ... Session-Daten im State Service speichern?

Im Regelfall werden sämtliche Session-Informationen direkt im Prozess der ASP.NET-Engine abgelegt. Sie können als Datenspeicher alternativ den so genannten Session State Service benutzen. Es handelt sich um einen Windows Service, der von der ASP.NET-Engine mittels TCP/IP angesprochen wird. Dieses System hat mehrere Vorteile:

- Der Service ist unabhängig vom IIS-Service beziehungsweise von der ASP.NET-Engine. Werden diese Dienste neu gestartet, bleibt der State Service aktiv und die abgelegten Session-Informationen somit erhalten.
- Dank der Verbindung über TCP/IP kann der State Service auf einem beliebigen anderen Rechner im lokal erreichbaren Verbund gestartet werden. Der Dienst kann somit autark vom eigentlichen Web-Server betrieben werden. Selbst bei einem Ausfall dieses Rechners stehen die Session-Informationen weiter zur Verfügung und können beispielsweise von einem Backup-System genutzt werden.

Einrichtung des State Service

Nach der Installation des .NET Frameworks auf dem System steht der neue Windows-Dienst „ASP.NET State Service“ zur Verfügung. Sie können den Service in der Management-Konsole „Dienste“ über den Verwaltungsbereich der Systemsteuerung aktivieren. Wollen Sie diese Speichermöglichkeit verwenden, so setzen Sie das Startverhalten auf „Automatisch“ und starten den Dienst gegebenenfalls einmalig manuell.

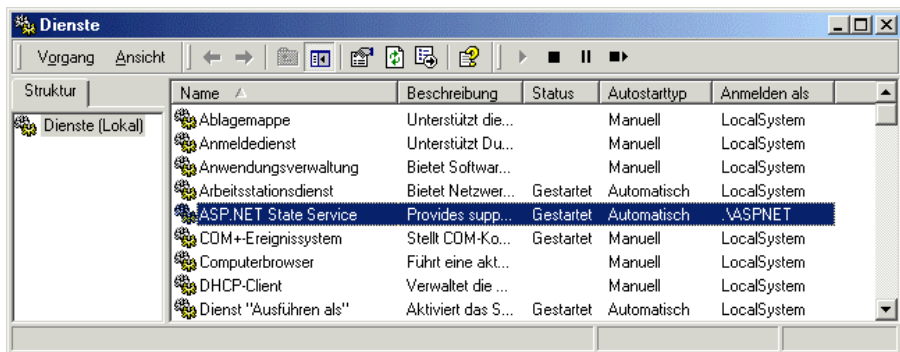


Abbildung 10.3 Der Session State Service in der Microsoft Management Console

Der State Service „lauscht“ auf dem TCP/IP-Port Nummer 42424. Sofern der Dienst auf einem anderen Rechner als dem Web-Server laufen soll, schalten Sie diesen Port unbedingt bei einer eventuell vorhandenen Firewall frei. Der Web-Server muss auf diesem Port die Kommunikation zum State Service aufnehmen können.

Damit Ihre Web-Applikation den Dienst auch tatsächlich nutzt, müssen Sie diesen in der Konfiguration hinterlegen. Hierzu passen Sie die lokale Datei `web.config` wie folgt an:

Listing 10.8 web.config

```
<configuration>
  <system.web>
    <sessionState mode="StateServer"
      stateConnectionString="tcpip=localhost:42424"/>
  </system.web>
</configuration>
```

Statt „localhost“ können Sie selbstverständlich auch eine beliebige andere IP-Adresse oder einen auflösbaren Servernamen angeben, über die beziehungsweise den der Dienst erreichbar ist.

Um auszuprobieren, ob die Daten nun tatsächlich über diesen Dienst gespeichert werden, können Sie die folgende Seite verwenden. Beim ersten Aufruf sollte keine Ausgabe im Browser erscheinen. Beim zweiten und weiteren Aufrufen wird hingegen „hallo welt“ ausgegeben. Starten Sie den Dienst nun über die MMC neu, so erscheint wieder der initiale leere Zustand, denn mit dem Neustart wurde auch das „Gedächtnis“ des Dienstes gelöscht.

Listing 10.9 SessionState1.aspx

```
<% @Page Language="C#" Debug="true" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
  if(Session["hallo"] != null)
    Response.Write(Session["hallo"].ToString());
  else
    Session["hallo"] = "hallo welt";
}

</script>
```



Anders als die standardmäßige Speicherung von Session-Informationen im Prozess der ASP.NET-Engine unterliegt der Session State Service einer Beschränkung. Es können nicht jegliche Informationen abgelegt werden, da die dazu notwendigen Objektinstanzen nicht über TCP/IP am „Leben“ erhalten werden können.

Jegliche Informationen, die im Session State abgelegt werden sollen, werden in einen binären Stream umgewandelt und in dieser Form übertragen und gespeichert. Werden die Daten wieder abgefragt, wird der Stream zurückübertragen und das entsprechende Objekt daraus neu erstellt. Man nennt diesen Vorgang Serialisierung beziehungsweise Deserialisierung.

Das .NET Framework bietet integrierte Funktionalitäten zur Serialisierung von Objekten. Die Standarddatentypen können direkt und ohne Umwege im Session State Service abgelegt werden. Das obige Beispiel hat dies gezeigt. Wie Sie auch komplexe Strukturen und eigene Klassen ablegen können, erfahren Sie in den entsprechenden Rezepten weiter unten.

10.4 ... Session-Daten im SQL-Server speichern?

Neben der Speicherung von Session-Informationen im Prozess der ASP.NET-Engine sowie im Session State Server (vergleiche vorheriges Rezept) können Sie die Daten auch auf einem SQL-Server ablegen. Es gelten hier die gleichen Vorteile wie beim State Service:

- Der SQL-Server ist unabhängig vom IIS-Service beziehungsweise von der ASP.NET-Engine. Werden diese Dienste neu gestartet, bleibt der SQL-Server aktiv, und die abgelegten Session-Informationen werden somit erhalten.
- Dank der optionalen Netzwerkverbindung kann der SQL-Server auf einem beliebigen anderen Rechner im lokal erreichbaren Verbund gestartet werden. Der SQL-Server kann somit autark vom eigentlichen Web Server betrieben werden. Selbst bei einem Ausfall dieses Rechners stehen die Session-Informationen weiter zur Verfügung und können beispielsweise von einem Backup-System genutzt werden.



Beachten Sie bitte, dass Sie den SQL-Server ab Version 2000 benötigen. Ältere Versionen oder auch andere Datenbanksysteme wie MySQL lassen sich zumindest derzeit nicht verwenden. Auch die eingeschränkte Desktop-Version MSDE, die mit der Entwicklungsumgebung Visual Studio .NET ausgeliefert wird, kann nicht verwendet werden.

Einrichtung des SQL-Servers

Damit die Session-Daten auf einem SQL-Server abgelegt werden können, muss dort eine entsprechende Datenbank mit den benötigten Tabellen angelegt werden. Damit Sie diese Arbeit nicht manuell durchführen müssen, bietet Ihnen ASP.NET ein vorkonfiguriertes SQL-Skript an, das Sie lediglich per Doppelklick ausführen müssen. Sie finden das Skript unter folgendem Dateinamen:

```
<WINDIR>\Microsoft.NET\Framework\<Version>\InstallSqlState.sql
```

Nach der erfolgreichen Einrichtung steht Ihnen eine neue Datenbank „ASPState“ zur Verfügung. Damit diese genutzt werden kann, müssen Sie die Web-Applikation nun noch entsprechend konfigurieren. Nachfolgend sehen Sie einen Ausschnitt der Konfigurationsdatei `web.config`.

Listing 10.10 `web.config`

```
<configuration>
  <system.web>
    <sessionState mode="SQLServer"
      sqlConnectionString="data source=localhost;user id=sa; password="/>
    </system.web>
  </configuration>
```

Unter dem Attribut `sqlConnectionString` können Sie einen beliebigen, gültigen Connection-String zu einem verfügbaren SQL-Server angeben. Dieser kann sich – wie beschrieben – auch auf einem anderen Rechner befinden. Geben Sie in diesem Fall einfach die entsprechende IP-Adresse beziehungsweise den Servernamen an.

Sie können die eingerichtete Datenbank im SQL-Server auch wieder löschen, wenn Sie nachträglich auf eine andere Datenquelle wechseln möchten. Auch hierzu bietet Ihnen ASP.NET ein vorkonfiguriertes SQL-Script an. Sie finden dieses unter dem folgenden Dateinamen im oben genannten Verzeichnis:

```
UninstallSqlState.sql
```



Wie beim vorher beschriebenen Session State Service werden auch bei der Speicherung von Session-Informationen im SQL-Server die Daten in einen binären Stream umgewandelt und später wieder zurückgewandelt. Auch hier müssen die Daten also serialisierbar sein. Informationen finden Sie in den nachfolgenden Rezepten.

10.5 ... ein DataSet im Session-Scope ablegen?

Ein DataSet ist der ultimative .NET-Daten-Container im Arbeitsspeicher. Sie können hier nahezu beliebige Daten in strukturierter Form ablegen. Neben der manuellen Anlage können diese Daten auch aus einer Datenbank übernommen werden.

Unter Umständen kann es sehr nützlich sein, ein erstelltes DataSet über mehrere Aufrufe einer Seite hinweg persistent im Speicher zu halten. Die Klasse besitzt eingebaute Mechanismen zur Serialisierung als XML-Stream. Es ist daher möglich, ein DataSet als Session-Variable abzulegen. Dies ist sowohl im Prozess der ASP.NET-Engine („InProc“), aber auch bei Verwendung der beiden externen Datenspeicher Session State Service und SQL-Server möglich.

Das Listing zeigt die Verwendung eines DataSets mit Session-Scope. Existiert die Session-Variable „Authors“ nicht, wird die gleichnamige Tabelle aus einer Datenbank gelesen und über einen DataSet in der Session abgelegt. Spätere Aufrufe werden direkt aus dieser Kopie bedient. Der Unterschied lässt sich anhand einer Statusausgabe im Browserfenster sowie eines DataGrid-Controls erkennen.

Listing 10.11 SessionDataSet1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    if(Session["Authors"] == null)
    {
        OleDbConnection conn = new
            OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
        conn.Open();

        string SQL = "SELECT * FROM AUTHORS";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);

        OleDbDataAdapter adapter = new OleDbDataAdapter();
        adapter.SelectCommand = cmd;

        DataSet dataset = new DataSet();
        adapter.Fill(dataset, "Authors");
        conn.Close();

        Session["Authors"] = dataset;
        Response.Write("<p>Daten wurden neu abgefragt.<p>");
    }

    grid.DataSource = Session["Authors"];
    grid.DataBind();
}

</script>

<ASP:DataGrid id="grid" runat="server"/>
```

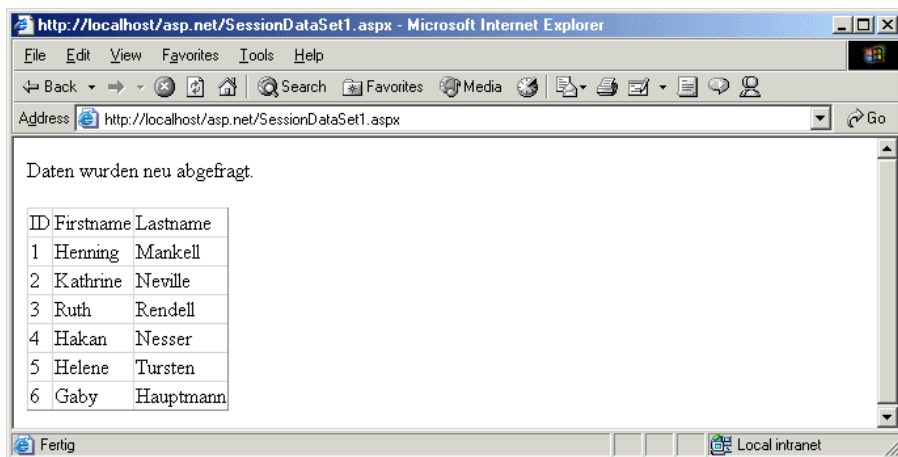


Abbildung 10.4 Die Daten kommen beim zweiten Aufruf aus dem Session-Speicher.



Bedenken Sie bei der Ablage von DataSet-Objekten im Session-Speicher, dass deren gesamter Inhalt sowie alle benötigten Meta-Informationen abgespeichert werden müssen. Je nach gewähltem Speichermedium werden die Daten beispielsweise im lokalen Arbeitsspeicher im Prozess der ASP.NET-Engine abgelegt. Der Arbeitsspeicher wird mit jedem neuen Benutzer weiter in Anspruch genommen. Entfernt werden die Daten erst beim impliziten Session-Timeout. Haben Sie hingegen den Session State Service oder den SQL-Server als Speicherort gewählt, müssen die Daten zusätzlich noch zwischen der ASP.NET-Engine und diesem Medium ausgetauscht werden.

Beschränken Sie sich aus den beschriebenen Gründen unbedingt bei der Ablage von derart komplexen Objekten wie einem DataSet im Session-Speicher. Wenn es sinnvoll ist, achten Sie zumindest darauf, dass so wenige Daten wie möglich abgelegt werden. Zudem sollten Sie die Session-Variable explizit löschen, wenn die Daten nicht weiter benötigt werden:

```
Session.Remove("Authors");
```

Alternativ können Sie die Session auch explizit beenden:

```
Session.Abandon();
```

10.6 ... eigene Strukturen und Klassen mit Session-Scope ablegen?

Eine wichtige Neuerung von ASP.NET gegenüber den alten ASP-Versionen ist die Möglichkeit, auch komplexe Daten wie Strukturen und Klasseninstanzen im Session-Speicher abzulegen. Denken Sie beispielsweise an Benutzerdaten oder einen Warenkorb, der über mehrere Seitenaufrufe hinweg persistent zur Verfügung stehen soll.

Sofern Sie mit dem eingebauten Session-Speicher im Prozess der ASP.NET-Engine arbeiten, können Sie in der Tat beliebige Daten ablegen. Intern wird eine Hashtable verwendet, die die Objektverweise aufnimmt und somit am Leben erhält. Bei Abfrage der Session-Variablen wird einfach der Objektverweis aus der Hashtable kopiert.

Bei den anderen möglichen Speicherorten Session State Service und SQL-Server müssen die Daten in einen binären Stream umgewandelt und dadurch quasi in ihre Einzelteile zerlegt werden. Beim erneuten Aufruf werden die Daten aus dem Stream wieder in Objektform gebracht – deserialisiert. Diese Umwandlung ist bei den Standardwerttypen von Haus aus möglich. Auch viele wichtige .NET-Klassen können serialisiert werden, beispielsweise die abstrakte Klasse `Image` und somit deren Ableitungen `Bitmap` und `Metafile`. Auch eigene Klassen und Strukturen können serialisiert werden.

Das Listing zeigt die Eingabe von Benutzervor- und -nachname. Diese Daten werden in einer Struktur `User` hinterlegt und in einer Session-Variablen vorgehalten. Als Speichermedium ist der Session State Service gewählt.

Listing 10.12 `Serialize1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    bool hasData = (Session["user"] != null);
    ph1.Visible = !hasData;
    ph2.Visible = hasData;

    if (hasData)
    {
        User user = (User) Session["user"];
        lb_user.Text = string.Format("{0} {1}", user.Firstname,
            user.Lastname);
    }
}

void bt_Click(object sender, EventArgs e)
```

```
{
    User user = new User();
    user.Firstname = tb_firstname.Text;
    user.Lastname = tb_lastname.Text;
    lb_user.Text = string.Format("{0} {1}",
        user.Firstname, user.Lastname);
    Session["user"] = user;
    ph1.Visible = false;
    ph2.Visible = true;
}

public struct User
{
    public string Firstname;
    public string Lastname;
}

</script>

<form runat="server">

    <ASP:Placeholder id="ph1" runat="server">
        Vorname: <ASP:TextBox id="tb_firstname" runat="server"/> <br>
        Nachname: <ASP:TextBox id="tb_lastname" runat="server"/> <br>
        <p> <ASP:Button runat="server" text="OK" OnClick="bt_Click"/> </p>
    </ASP:Placeholder>

    <ASP:Placeholder id="ph2" runat="server">
        <p> Ihr Name lautet: <ASP:Label id="lb_user" runat="server"/> </p>
    </ASP:Placeholder>

</form>
```

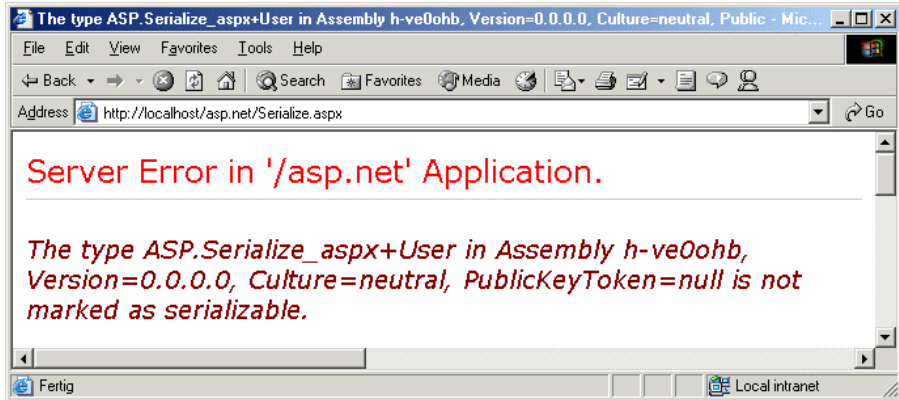


Abbildung 10.5 So einfach ist die Ablage einer Struktur nicht.

Nach Eingabe und Bestätigung des Web-Formulars wird ein Laufzeitfehler angezeigt, den Sie in der Abbildung sehen können. Da als Speicherort der Session State Service gewählt wurde, wird versucht, die Daten zu serialisieren. Dies ist allerdings nicht möglich, da die Struktur hierzu mit einem speziellen Attribut `Serializable` versehen werden muss.

Listing 10.13 `Serialize2.aspx`

```
...
[Serializable]
public struct User
{
    public string Firstname;
    public string Lastname;
}
...
```

Das Attribut kennzeichnet eine Struktur aber auch eine Klasse als serialisierbar. Sie können dieses Attribut immer dann direkt anwenden, wenn alle enthaltenen Mitglieder ihrerseits serialisierbar sind. Ist dies nicht der Fall, müssen Sie die Serialisierung explizit und manuell vornehmen.

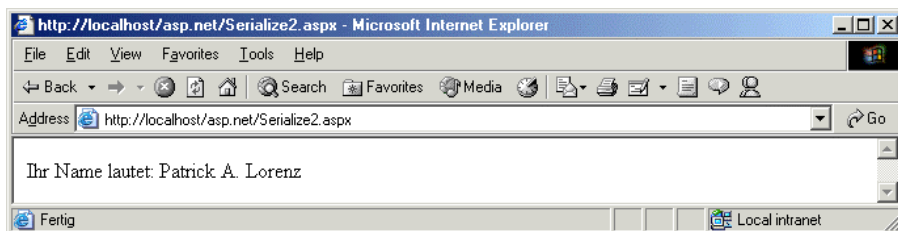


Abbildung 10.6 Ist eine Struktur serialisierbar, kann sie im Session-Scope abgelegt werden.

10.7 ... ein Objekt individuell serialisieren?

Standarddatentypen können automatisch vom .NET Framework serialisiert werden. Das gilt auch für Mitglieder von Strukturen und Klassen mit dem entsprechenden Typ. Sie können dieser automatischen Serialisierung aber auch eine individuelle vorziehen. Sofern Ihre Klasse Mitglieder enthält, die nicht automatisch umgewandelt werden können, müssen Sie dies sogar tun.

Um eine individuelle Serialisierung zu implementieren, müssen Sie die Schnittstelle `ISerializable` unterstützen. Diese liefert Ihnen über die Methode `GetObjectData` eine Instanz der Klasse `SerializationInfo`. Die Methode wird aufgerufen, wenn das Objekt serialisiert werden soll. Hier können Sie ähnlich wie bei einem Dictionary Daten in einer Schlüssel-Wert-Beziehung hinterlegen und somit den Status Ihres Objekts speichern. Umgekehrt läuft es dann beim Konstruktor. Auch hier erhalten Sie eine Instanz der genannten Klasse und können die abgelegten Daten wieder auslesen und den jeweiligen Mitgliedsvariablen zuweisen.

Das Listing zeigt eine individuelle Serialisierung der Klasse `User` aus dem vorherigen Rezept. Beachten Sie, dass zusätzlich zur Unterstützung der Schnittstelle `ISerializable` immer auch das Attribut `Serializable` gesetzt werden muss.

Listing 10.14 Serialize3.aspx

```
...
[Serializable]
public class User : ISerializable
{
    public string Firstname;
    public string Lastname;

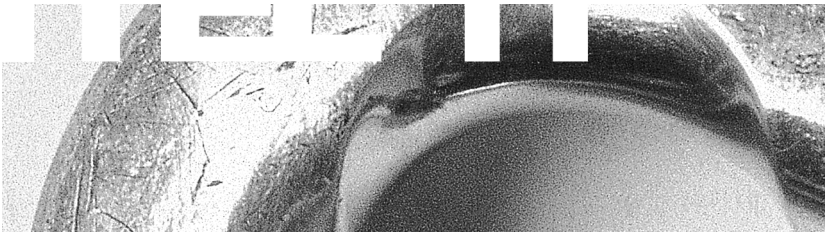
    public User() {}

    protected User(SerializationInfo info, StreamingContext context)
    {
        this.Firstname = info.GetString("firstname");
        this.Lastname = info.GetString("lastname");
    }

    void ISerializable.GetObjectData(SerializationInfo info,
        StreamingContext context)
    {
        info.AddValue("firstname", this.Firstname);
        info.AddValue("lastname", this.Lastname);
    }
}
...
```

11

Sicherheit



Wie kann ich ...

11 Sicherheit

Das Thema Sicherheit gehört zu den umstrittensten im Bereich des Internets. Spätestens wenn Sie mit persönlichen Daten Ihrer Benutzer hantieren, sollten Sie sich ernsthafte Gedanken über die Sicherheit dieser Daten machen. Das gilt insbesondere dann, wenn Kunden bei Ihnen online Einkaufen sollen. Sie benötigen ein ausgewogenes Konzept, das alle Bereiche abdeckt, an denen die Daten verarbeitet werden. Dies betrifft auch die Übertragung von Daten vom Server an Ihre Firmenzentrale, beispielsweise per Email.

Mit ASP.NET und den Internet Information Services stehen Ihnen zahlreiche Möglichkeiten offen, Techniken zur Verbesserung der Sicherheit zu implementieren. Sie können ein SSL-Zertifikat installieren und so ohne weiteres Zutun eine verschlüsselte Verbindung zwischen Client-Browser und Ihrem Server anbieten. Auch andere Übertragungsformen lassen sich individuell mit den Klassen des Frameworks verschlüsseln. Damit bestimmte Bereiche Ihrer Website nur den dazu befugten Benutzern zugänglich sind, bietet ASP.NET Ihnen verschiedene Wege zur Authentifizierung und Autorisation.

Wenn Ihre Website gesichert ist, dann sollten Sie dies auch kommunizieren. Gehen Sie offen mit Ihren Sicherheitsvorkehrungen um, ohne Details zu verraten. Geben Sie Ihrem Kunden das Versprechen, dass seine Daten sorgfältig und nach dem besten Gewissen sicher verarbeitet werden. Transparenz schafft Vertrauen. Vertrauen schafft eine gute Kundenbeziehung, und diese wiederum ist bekanntermaßen ein wichtiger Grundstein für langfristige, erfolgreiche Geschäfte.

11.1 ... feststellen, ob es sich um eine sichere Verbindung handelt?

Ruft der Benutzer Ihre Website auf, so gibt er meistens `www.firmenname.tld` ein. Die Verbindung wird nun mittels HTTP aufgebaut. Alle Daten werden hierbei unverschlüsselt übertragen. Bevor der Benutzer persönliche Daten eingibt, sollten Sie sicherstellen, dass eine per SSL geschützte Verbindung verwendet wird.

Die Eigenschaft `Request.IsSecureConnection` liefert die gewünschte Information. Ist der zurückgelieferte boolesche Wert `false`, so findet der Datenaustausch unverschlüsselt statt. Sie können nun beispielsweise über einen Redirect auf die verschlüsselte Verbindung umschalten. Alternativ überlassen Sie dem Benutzer die Auswahl, ob er eine sichere Verbindung wünscht oder nicht.

Das Listing zeigt die Verwendung der Eigenschaft. Handelt es sich nicht um eine geschützte Verbindung, wird ein Redirect auf dieselbe Seite durchgeführt. Hierzu wird die aktuelle Adresse ermittelt und die Angabe des Protokolls „http:“ durch „https:“ ersetzt.

Listing 11.1 `IsSecureConnection1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!Request.IsSecureConnection)
    {
        string url = Request.Url.AbsoluteUri;
        url = url.Replace("http:", "https:");
        Response.Redirect(url);
    }
    else
    {
        Response.Write("Die Verbindung ist sicher!");
    }
}

</script>
```

Der erste Aufruf der Seite ohne SSL resultiert automatisch in einem Redirect auf die sichere Variante. Diese wird durch ein Symbol in der Statusleiste des Browsers angezeigt. Beim Internet Explorer handelt es sich um ein Schloss, bei Netscape um einen Schlüssel.

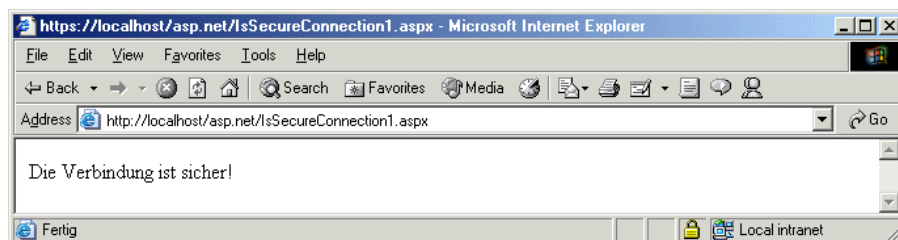


Abbildung 11.1 Die Verbindung findet verschlüsselt statt.

11.2 ... die Bit-Stärke der SSL-Verschlüsselung

Bei einer über HTTPS verschlüsselten Verbindung ist die Bit-Stärke des verwendeten SSL-Zertifikats absolut maßgeblich. Je länger der Schlüssel ist, je mehr Bits er also umfasst, desto besser ist die Verschlüsselung und desto größer der Schutz.

Sie können die Bit-Stärke des verwendeten Zertifikats über die Server-Variable „HTTPS_KEYSIZE“ ermitteln. Idealerweise sollte diese bei mindestens 128, also 2^7 liegen.

Listing 11.2 KeySize1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("Bit-Stärke der Verschlüsselung: ");
    Response.Write(Request.ServerVariables["HTTPS_KEYSIZE"]);
}

</script>
```

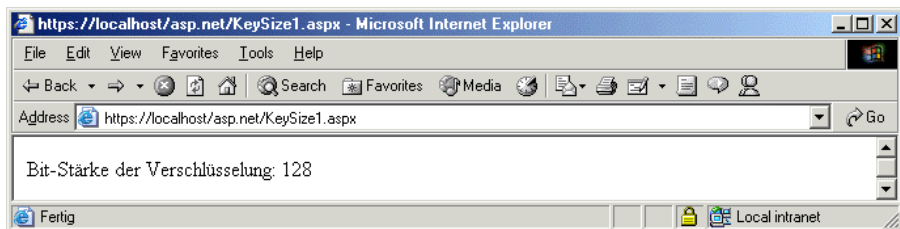


Abbildung 11.2 Die Website ist mit 128 Bit gut geschützt.

11.3 ... die Email-Adresse des Benutzers verifizieren?

Wenn Sie eine Anmeldung von Benutzern ermöglichen, werden Sie es vermutlich ungern sehen, dass die Anwarter statt ihrer persönlichen Daten John Doe oder Janet Smith angeben. Doch wie können Sie die Benutzer dazu bekommen, echte Daten anzugeben? Nun, zunächst einmal müssen Ihre Inhalte einen derartigen Mehrwert bieten, dass die Benutzer überhaupt zu einer Anmeldung und Angabe Ihrer Daten bereit sind.

Ist diese Grundvoraussetzung geschaffen, können Sie sich eines einfachen, aber genauso effektiven Tricks bedienen. Statt dem Benutzer die Eingabe eines Passworts zu ermöglichen, übernehmen Sie diese Arbeit für ihn. Das automatisch erzeugte Passwort wird an die angegebene Email-Adresse gesendet. Nur wenn diese wahrheitsgemäß angegeben wurde, erhält der Benutzer das Passwort und bekommt erst so Zugriff auf den geschützten Bereich Ihrer Web-Applikation.

Optional können Sie dem Benutzer die Möglichkeit anbieten, nach der Verifizierung und erstmaligen Anmeldung sein Passwort individuell zu ändern.

11.4 ... ein Passwort erstellen?

Es gibt unterschiedliche Verfahren, ein Passwort zu erstellen. Besonders gern verwendet werden die folgenden drei Verfahren. Jedes davon hat seine Vor- und Nachteile.

- Am einfachsten ist das zufällige Aneinanderfügen von Buchstaben und Zahlen. So ergibt sich ein relativ sicheres, wenngleich schwer zu merkendes Passwort.
- Eine andere Möglichkeit ist die Verwendung einer Wortdatenbank. Statt das Passwort neu zu generieren, suchen Sie sich ein zufälliges Wort aus der Datenbank heraus. Natürlich können Sie auch mehrere Wörter zufällig aneinanderhängen. Ein derartiges Passwort ist zwar leicht zu merken, dafür aber relativ unsicher. Verwendet der potenzielle Eindringling ebenfalls eine Wortdatenbank, kann er dieses automatisiert abgleichen und so möglicherweise Zugang erhalten.
- Ich persönlich bevorzuge so genannte mnemonische Passwörter. Diese werden ebenfalls zufällig erstellt, basieren aber auf einer Kombination von nacheinander folgenden Konsonanten und Vokalen. Erstaunlicherweise generiert dieses Verfahren außerordentlich leicht les- und merkbare Passwörter. Mitunter klingen die Wörter sogar sehr witzig¹. Auf der anderen Seite bieten diese Passwörter aber auch einen guten Kompromiss an Sicherheit.

Nachfolgend möchte ich Ihnen die verschiedenen Algorithmen zur Erstellung von Passwörtern vorstellen.

¹ Wer sich schon einmal gefragt hat, woher ein großes schwedisches SB-Möbelhaus seine überaus skandinavisch klingenden Namen hat, der findet hier vielleicht die Lösung ... ;-))

Das zufällige Passwort

Basis für ein zufälliges Passwort sind mehrere aneinander gehängte Zufallszeichen. Diese zu ermitteln wird einer Instanz der Klasse `Random` überlassen. Die mehrfach überladene Methode `Next` ermöglicht die Abfrage eines zufälligen Werts. Hierbei kann der Maximalwert oder auch ein Bereich angegeben werden, indem sich der Wert befinden soll.

Das Listing zeigt eine statische Methode `GetRandom`. Dieser können die gewünschte Passwortlänge sowie drei boolesche Werte übergeben werden. Diese legen fest, ob Zahlen, Großbuchstaben und/oder Kleinbuchstaben zur Erstellung des Passworts verwendet werden sollen. Eine `while`-Schleife sorgt dafür, dass das Passwort mit der gewünschten Größe entsprechend den Vorgaben erstellt wird.

```
public static string GetRandom(int length, bool AllowNumeric,
    bool AllowUpperCase, bool AllowLowerCase)
{
    if(!AllowNumeric && !AllowUpperCase && !AllowLowerCase)
        throw new ArgumentException("At least one type must be allowed!");

    string pw = string.Empty;
    Random rnd = new Random((int)DateTime.Now.Ticks);

    while(pw.Length < length)
    {
        switch(rnd.Next(3))
        {
            case 0:
                if(AllowNumeric) pw += Convert.ToChar(rnd.Next(48, 58));
                break;

            case 1:
                if(AllowUpperCase) pw += Convert.ToChar(rnd.Next(65, 91));
                break;

            case 2:
                if(AllowLowerCase) pw += Convert.ToChar(rnd.Next(97, 123));
                break;
        }
    }
    return(pw);
}
```

Im Listing sehen Sie die Verwendung der statischen Methode `Convert.ToChar`. Diese wird verwendet, um einen `int`-Wert in das entsprechende ASCII-Zeichen zu konvertieren. Dieses wird der Passwort-Zeichenkette angehängt, bis diese die angegebene Länge umfasst.

Die Anzahl der möglichen Kombination berechnet sich wie folgt:

$$(26+26+10)^{\text{Zeichenzahl}}$$

Für ein acht Zeichen langes, rein zufälliges Passwort ergeben sich somit 218.340.105.584.896 mögliche Kombinationen.

Das wortbasierte Passwort

Die Erstellung eines wortbasierten Passwortes ist simpel. Sie müssen lediglich die gewünschten Wörter aus einer Datenbanktabelle zufällig abfragen. Sie benötigen hierzu lediglich eine entsprechende Datenbank. Je mehr Wörter diese enthält, desto sicherer wird das Passwort. Dennoch ist diese Variante nicht zu empfehlen.

Die Anzahl der möglichen Kombinationen berechnet sich wie folgt:

$$\text{Datensätze}^{\text{Wortanzahl}}$$

Das mnemonische Passwort

Mnemonische Passwörter sind wirklich nett. Sie bestehen aus einer Kombination von hintereinander gesetzten Konsonanten und Vokalen. Durch diese Kombination lässt sich jedes Passwort aussprechen und daher gut merken.

Die Erstellung des Passwortes basiert auf zwei char-Arrays mit den Konsonanten beziehungsweise Vokalen. Diese werden zufällig aneinander gehängt, bis sich die gewünschte Länge ergibt. Diese muss immer gerade sein. Eine ungerade Länge wird über eine Modulo-Operation korrigiert.

```
public static string GetMnemonic(int length)
{
    string pw = string.Empty;
    Random rnd = new Random((int)DateTime.Now.Ticks);

    if((length % 2) != 0) length++;

    char[] consonants = {'b','c','d','f','g','h','j','k','l','m','n',
        'p','q','r','s','t','v','w','x','y','z'};
    char[] vowels = {'a','e','i','o','u'};

    for(int i=0; i<(length/2); i++)
    {
        pw += consonants[rnd.Next(21)];
        pw += vowels[rnd.Next(5)];
    }
    return(pw);
}
```


Die Anzahl der möglichen Kombinationen berechnet sich wie folgt:

$$(21 * 5)^{\text{Zeichenzahl}/2}$$

Für ein acht Zeichen langes mnemonisches Passwort ergeben sich somit 121.550.625 mögliche Kombinationen. Das ist im Vergleich zum rein zufälligen Passwort zwar deutlich unsicherer, jedoch dennoch ein guter Wert.

Das Beispiel

Damit Sie die beiden vorgestellten Algorithmen ausprobieren können, habe ich ein kleines Web-Formular entworfen. Hier kann der Benutzer entscheiden, ob er ein mnemonisches oder zufälliges Passwort wünscht. In letzterem Fall kann er angeben, welche Zeichen (Zahlen, Kleinbuchstaben und Großbuchstaben) im Passwort enthalten sein sollen. Die Klasse `PasswordGenerator` erlaubt über die gezeigten statischen Klassen die Erstellung der Passwörter. Die Abbildung zeigt das Beispiel im Einsatz.

Listing 11.3 passwordgenerator1.aspx

```
...

void Page_Load(Object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        radioMnemonic.Checked = true;
        Generate_Click(null, null);
    }
}

void Generate_Click(Object sender, EventArgs e)
{
    int length = int.Parse(pwLength.Value);

    if (radioMnemonic.Checked)
    {
        pw.Value = PasswordGenerator.GetMnemonic(length);
    }
    else
    {
        pw.Value = PasswordGenerator.GetRandom(length,
            rndNumeric.Checked, rndUpperCase.Checked, rndLowerCase.Checked);
    }
}

</script>
```

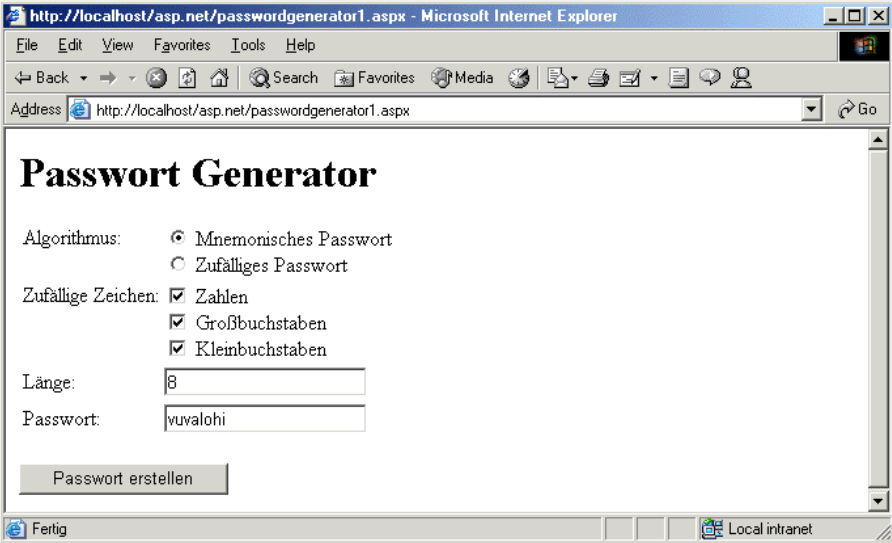
```
<h1>Passwort Generator</h1>

<form runat="server">

<table>
  <tr>
    <td valign="top">Algorithmus:</td>
    <td>
      <input type="radio" id="radioMnemonic" runat="server"/>
      Mnemonisches Passwort <br>
      <input type="radio" id="radioRandom" runat="server"/>
      Zufälliges Passwort
    </td>
  </tr>
  <tr>
    <td valign="top">Zufällige Zeichen:</td>
    <td>
      <input type="checkbox" id="rndNumeric" checked="true"
      runat="server"/> Zahlen <br>
      <input type="checkbox" id="rndUpperCase" checked="true"
      runat="server"/> Großbuchstaben <br>
      <input type="checkbox" id="rndlowerCase" checked="true"
      runat="server"/> Kleinbuchstaben <br>
    </td>
  </tr>
  <tr>
    <td>Länge:</td>
    <td>
      <input type="text" id="pwLength" value="8" runat="server"/>
    </td>
  </tr>
  <tr>
    <td>Passwort:</td>
    <td>
      <input type="text" id="pw" value="" runat="server"/>
    </td>
  </tr>
</table>

<p>
  <input type="button" id="Generate" value="Passwort erstellen"
  Onserverclick="Generate_Click" runat="server"/>
</p>

</form>
```



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/asp.net/passwordgenerator1.aspx`. The page title is "Passwort Generator". The form contains the following elements:

- Algorithmus:** Two radio buttons. The first, "Mnemonisches Passwort", is selected. The second is "Zufälliges Passwort".
- Zufällige Zeichen:** Three checked checkboxes: "Zahlen", "Großbuchstaben", and "Kleinsbuchstaben".
- Länge:** A text input field containing the number "8".
- Passwort:** A text input field containing the string "yuvalohi".
- Buttons:** A "Passwort erstellen" button and a "Fertig" button at the bottom left.

The status bar at the bottom indicates "Local intranet".

Abbildung 11.3 Das Web-Formular erlaubt die Erstellung von Passwörtern.

11.5 ... einen Hashcode erstellen?

Das Problem an Passwörtern ist, dass Sie in der Regel an irgendeiner Stelle hinterlegt sind. Bei einer Web-Applikation verwenden Sie typischerweise eine Datenbank, in der alle Passwörter gespeichert sind. Will sich ein Benutzer anmelden, vergleichen Sie das von ihm eingegebene Passwort mit dem Datenbankinhalt. Das System funktioniert auf diese Weise einwandfrei, doch bleibt immer ein schaler Beigeschmack übrig. Was, wenn jemand die Datenbank in die Finger bekommt? Hier sind alle Passwörter hinterlegt und dem entsprechenden Benutzer-Account zugeordnet. Ein echter GAU für Ihre Website ...

Abhilfe schaffen so genannte Passwort-Hashes. Dabei handelt sich um einen aus dem Passwort errechneten Code. Die Berechnung ist jedoch in eine Richtung möglich, so dass sich aus dem Hashcode das ursprüngliche Passwort nicht mehr ermitteln lässt. Nach Erstellung des Passworts errechnen Sie den Hashcode und speichern diesen in der Datenbank ab. Gibt der Benutzer sein Passwort ein, wird auch hierfür der Hashcode erstellt und mit dem Inhalt in der Datenbank verglichen. Selbst wenn nun jemand die Datenbank in die Finger bekommt, kann er daraus das Passwort nicht ermitteln. Das eigentliche Passwort ist nirgends gespeichert.

Das .NET Framework bietet über die statische Methode `FormsAuthentication.HashPasswordForStoringInConfigFile` die Möglichkeit, Hashcodes mit zwei unterschiedlichen Algorithmen zu erstellen: SHA1 und MD5. Das Beispiel zeigt die Verwendung der hier etwas zweckentfremdeten Methode.

Listing 11.4 HashPassword1.aspx

```
<script language="C#" runat=server>

void CreateHash_Click(object sender, EventArgs e)
{
    string hashFormat = rb_SHA1.Checked ? "SHA1" : "MD5";
    tb_pw_hash.Text = FormsAuthentication.
        HashPasswordForStoringInConfigFile(tb_pw.Text, hashFormat);
}

</script>

<form runat="server">

<h1>Passwort-Hash erstellen</h1>

<p> Hash-Verfahren:
  <ASP:RadioButton id="rb_SHA1" runat="server"
    GroupName="hashFormat"
    Text="SHA1"
    Checked="true"/>

  <ASP:RadioButton id="rb_MD5" runat="server"
    GroupName="hashFormat"
    Text="MD5"/>
</p>

<p> Klartext-Passwort:
  <ASP:TextBox id="tb_pw" runat="server"/>
</p>

<p> Passwort-Hash:
  <ASP:TextBox id="tb_pw_hash" runat="server"
    Width="400"/>
</p>

<ASP:Button id="CreateHash" runat="server"
  Text="Passwort-Hash erstellen"
  OnClick="CreateHash_Click"/>

</form>
```

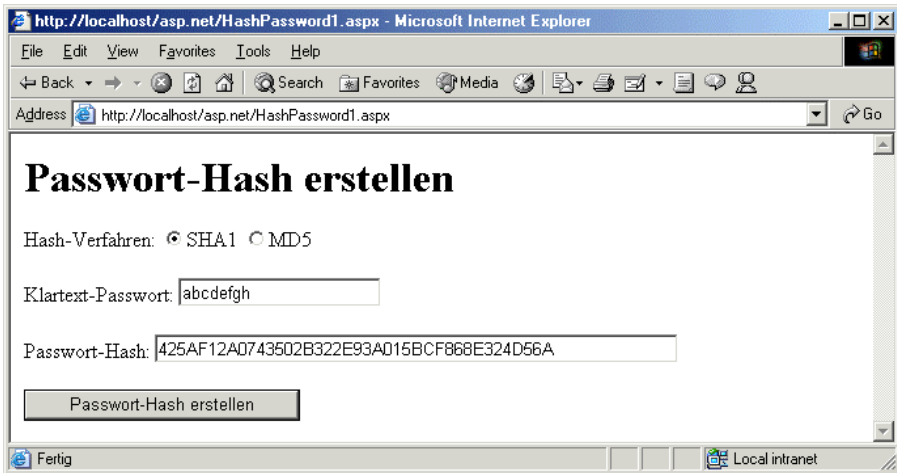


Abbildung 11.4 Aus dem Hashcode lässt sich das Passwort nicht mehr errechnen.



Die Erstellung von Hashcodes scheint sehr wirksam zu sein. Das ist sie auch, dennoch gibt es einen kleinen Nachteil. Benutzer haben die unangenehme Eigenschaft, ihr Passwort zu vergessen. Viele Websites bieten daher die Möglichkeit, den vergebenen Zugangscode bei Verlust erneut an die angegebene Email-Adresse zu senden. Das klappt bei einem Hashcode natürlich nicht. Hier bietet es sich an, ein neues Passwort zu vergeben, das dem Benutzer zugesandt wird. In der Datenbank wird einfach der neue Hashcode abgelegt.

11.6 ... Forms Authentication mit Window-Benutzerdaten überprüfen?

Die .NET Framework SDK-Dokumentation gibt an, dass sich die Forms Authentication nicht in Verbindung mit der Windows Authentication nutzen lässt. Das ist im Prinzip durchaus logisch, denn es handelt sich schließlich um zwei alternative Techniken zur Identifizierung von Benutzern. Dennoch mag es unter Umständen sinnvoll sein, die Forms Authentication auf Basis der hinterlegten Windows-Benutzer-Accounts durchzuführen. Das Framework bietet hierzu keine Möglichkeiten, mit Hilfe von zwei Win32 API-Funktionen lässt sich die Überprüfung jedoch einfach realisieren.

Das Listing zeigt den Import der Funktion `LogonUser` aus der DLL `advapi32.dll` mit Hilfe des `DllImport`-Attributs. Das Web-Formular ermöglicht die Eingabe von Benutzername und Passwort. Nach einem Buttonklick werden diese mit Hilfe der Funktion gegen die Windows-Daten überprüft. Liefert `LogonUser` das O.K., wird die Freigabe für die Forms Authentication manuell erteilt.

Listing 11.5 login.aspx

```
<% @Import Namespace="System.Runtime.InteropServices" %>

<script language="C#" runat=server>

[DllImport("advapi32.dll")]
public static extern bool LogonUser(string lpszUsername, string
    lpszDomain, string lpszPassword, int dwLogonType, int
    dwLogonProvider, out int phToken);

[DllImport("Kernel32.dll")]
public static extern int GetLastError();

void Submit_Click(object sender, EventArgs e)
{
    int usertoken;
    bool LoginOK = LogonUser(tb_username.Text, "", tb_password.Text,
        3, 0, out usertoken);

    if(LoginOK)
    {
        FormsAuthentication.RedirectFromLoginPage(tb_username.Text, false);
    }
    else
    {
        {
            int LastError = GetLastError();
            lb_info.Text = "Die Anmeldung konnte nicht bestätigt werden.
                Bitte versuchen Sie es noch einmal. (" + LastError + ")";
        }
    }
}

</script>

<form runat="server">

<h1>Login</h1>

<p> Bitte geben Sie Ihre Benutzerdaten an:</p>

<p>
    Benutzername:
    <ASP:TextBox id="tb_username" runat="server"/>
</p>

<p>
    Benutzername:
    <ASP:TextBox id="tb_password" runat="server"
        TextMode="Password"/>
</p>
```

```

<p> <ASP:Label id="lb_info" runat="server"/> </p>

<ASP:Button id="Submit" runat="server"
  Text="Login"
  OnClick="Submit_Click"/>

</form>

```

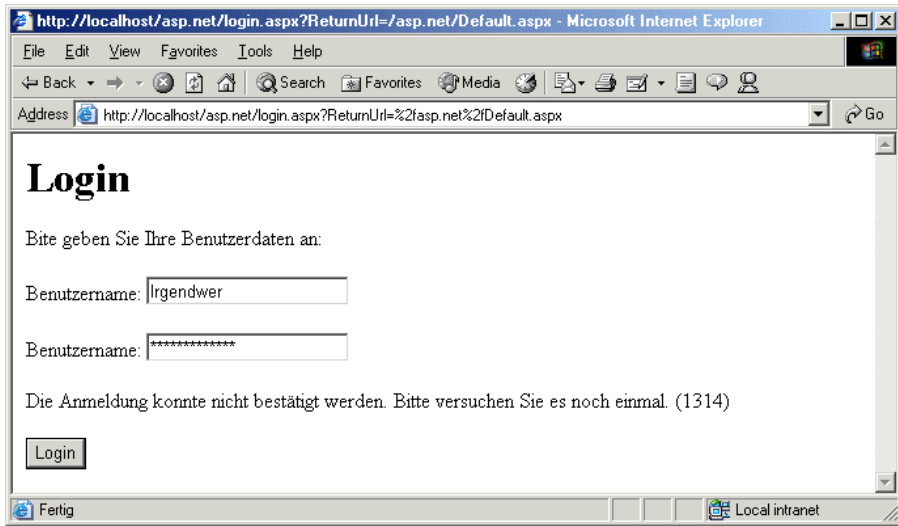


Abbildung 11.5 Der Zugriff ist nur mit gültigen Benutzerdaten möglich.



Sollte die Anmeldung trotz korrekter Benutzerdaten nicht glücken, so verfügen Sie vermutlich nicht über die entsprechenden Logon-Rechte. Überprüfen Sie in diesem Fall die für den Benutzer gültige Sicherheitsrichtlinie. Sie benötigen das Recht „Ersetzen eines Tokens auf Prozessebene“.

11.7 ... Impersonation mit Forms Authentication nutzen?

Impersonation ist ein nützliches Feature, um innerhalb von ASP.NET einen Benutzer und dessen Rechte anzunehmen. Impersonation wird vom Framework nur in Verbindung mit der Windows Authentication angeboten. Mit einem kleinen Trick und der Win32 API geht es jedoch auch mit Forms Authentication.

Die Überprüfung der Benutzerdaten erfolgt mittels der Win32 API-Funktion `LogonUser` wie im vorherigen Rezept „... Forms Authentication mit Windows-Benutzerdaten überprüfen?“ beschrieben. Der letzte Parameter der Funktion ist als `out` markiert und liefert einen Benutzer-Token zurück. Es handelt sich quasi um ein Handle auf den entsprechenden Account. Der `IntPtr`-Wert wird in innerhalb der Login-Seite in einer Session-Variablen abgelegt.

Listing 11.6 login.aspx

```
int usertoken;
bool LoginOK = LogonUser(tb_username.Text, "", tb_password.Text,
    3, 0, out usertoken);

if(LoginOK)
{
    Session["usertoken"] = usertoken;
}
...
```

Mittels der statischen Methode `FormsAuthentication.RedirectFromLoginPage` wird wie üblich von der Login-Seite auf die ursprünglich angeforderte Seite zurückgewechselt. Hier kann nun der gespeicherte Token abgefragt und dem Konstruktor der Klasse `WindowsIdentity` übergeben werden. Mit Hilfe der angebotenen Methode `Impersonate` lassen sich nun der Benutzer und dessen Rechte auf den aktuellen Prozess anwenden.

Listing 11.7 default.aspx

```
<% @Import Namespace="System.Security.Principal" %>

<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    int usertoken = (int) Session["usertoken"];

    WindowsIdentity ImpersonationIdentity =
        new WindowsIdentity((IntPtr) usertoken);

    WindowsImpersonationContext ImpersonationContext =
        ImpersonationIdentity.Impersonate();

    WindowsIdentity Identity = WindowsIdentity.GetCurrent();
    lb_identity.Text = Identity.Name;
}

</script>
```



```
<form runat="server">

<p> Sie sind angemeldet als:
<ASP:Label id="lb_identity" runat="server"/>
</p>

</form>
```

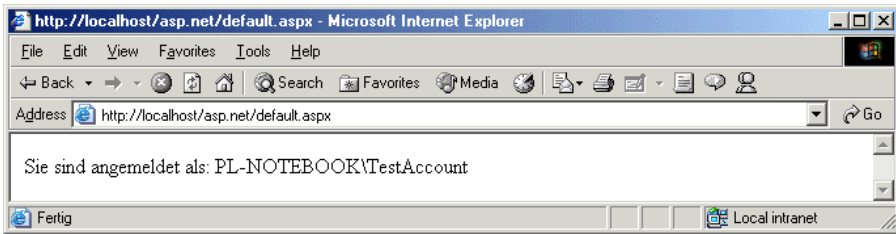


Abbildung 11.6 Der Prozess hat den angegebenen Benutzer angenommen.



Nach Ermittlung des Benutzer-Tokens arbeitet die Impersonation wie von der Windows Authentication gewohnt. Das bedeutet auch, dass Sie Bearbeitung der aktuellen Anfrage abläuft und bei einer neuen Anfrage erneut durchgeführt werden muss. Sollen alle weiteren Seitenabfragen des Benutzers die Impersonation nutzen, empfiehlt sich die Verwendung einer zentralen Code Behind-Klasse.

11.8 ... alle Dateien mit Forms Authentication schützen?

Die Windows Authentication basiert auf den Internet Information Services. Ist ein Verzeichnis vor dem Zugriff geschützt, erkennen die IIS dies und fordern den Benutzer zur Eingabe seiner Daten auf. Dies geschieht unabhängig von der angeforderten Datei. Es sind somit sowohl Scripte, ASP.NET-Seiten als auch beliebige andere Dateien wie Grafiken und Sounds geschützt.

Die Forms Authentication wird autark von der ASP.NET-Engine ohne die IIS realisiert. Ruft ein Benutzer eine Seite auf, wird die Anfrage von den IIS an die ASP.NET-Engine weitergereicht. Diese überprüft den Schutz und fordert gegebenenfalls zur Eingabe der Benutzerdaten auf. Dies gilt allerdings nur für diejenigen Dateien, die von den IIS an die Engine weitergereicht werden. Die Erkennung erfolgt auf Basis der Dateieindung. Die Endungen `aspx`, `asmx`, `cs`, `vb` und viele mehr werden übergeben. Andere Formate werden jedoch direkt von den IIS bedient,

hierzu zählen beispielsweise auch Grafiken. Die Sicherheitsabfragen von ASP.NET kommen hier nicht zum Einsatz, und so können derartige Dateien in jedem Fall abgerufen werden.

Die Lösung des Problems ist ziemlich einfach. Sie müssen lediglich dafür sorgen, dass die zu schützenden Dateien von den IIS an die ASP.NET-Engine weitergeleitet werden. Dies können Sie im Internetdienste-Manager im Verwaltungsbereich der Systemsteuerung konfigurieren.

1. Öffnen Sie den Internetdienste-Manager.
2. Öffnen Sie das Eigenschaftenfenster der Website oder der untergeordneten Web-Applikation, für die die Einstellung gelten soll.
3. Wählen Sie die Lasche „Basisverzeichnis“ beziehungsweise „Virtuelles Verzeichnis“ aus.
4. Klicken Sie auf den Button „Konfigurieren“.
5. Öffnen Sie den Eintrag für die Dateieindung `.aspx`, kopieren Sie den Wert des ersten Eingabefeldes „Ausführbare Datei“ in die Zwischenablage, und schließen Sie den Dialog.
6. Fügen Sie einen neuen Eintrag hinzu. Fügen Sie im ersten Eingabefeld den Inhalt der Zwischenablage ein, geben die Dateieindung `.gif` (inklusive Punkt!) an, aktivieren die Checkbox „Überprüfen, ob Datei existiert“ und bestätigen den Dialog.

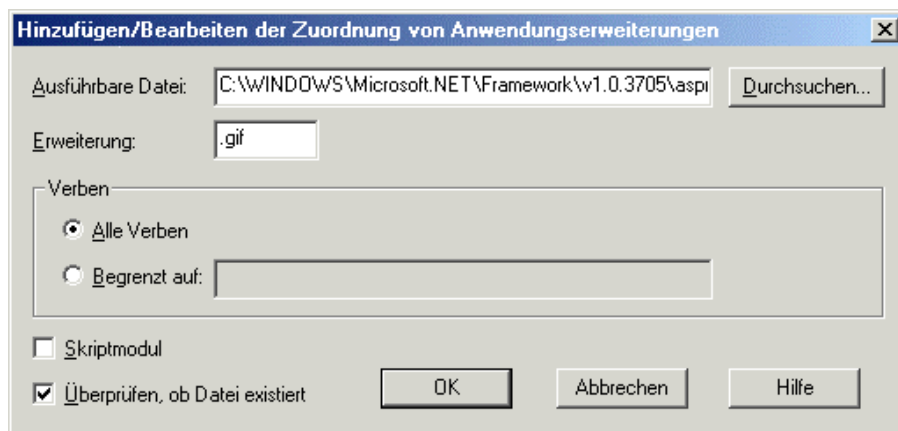


Abbildung 11.7 Die Dateieindung ist nun der ASP.NET-Engine zugewiesen.

Nun wurde die Dateieindung `gif` mit der ASP.NET-Engine verknüpft. Rufen Sie eine Grafik im Browser auf, wird – falls aktiviert – zunächst die Forms Authentication durchgeführt. Erst wenn diese das O.K. gibt, wird die Datei tatsächlich an den

Client übertragen. Die Grafik kann also ohne gültige Benutzerdaten nicht mehr angezeigt werden. Möchten Sie weitere Dateien schützen, so richten Sie analog weitere Verknüpfungen ein.



Das System funktioniert, da die Forms Authentication auf einem Cookie basiert. Dieses so genannte Authentication Ticket wird bei jedem Client-Aufruf zurück an den Server gesandt, selbst wenn es sich um eine Grafik handelt.

11.9 ... Download-Dateien vor dem direkten Zugriff schützen?

Viele Websites bieten Dokumente und Programme zum Download an. Oftmals sollen zunächst Benutzerdaten abgefragt werden, bevor das Herunterladen der Datei möglich ist. Kennt der Besucher jedoch deren Adresse, kann er die Eingabe umgehen und die Datei direkt herunterladen. Dies ist insbesondere dann ärgerlich, wenn die Datei einem besonderen Schutz unterstehen und nur von einer ausgewählten Besuchergruppe heruntergeladen werden soll.

Eine Möglichkeit, den Download einzuschränken, ist die Verwendung von Windows oder Forms Authentication. In letzterem Fall kann die Dateiendung wie im Rezept „... alle Dateien mit Forms Authentication schützen?“ beschrieben mit der ASP.NET-Engine verknüpft und somit nur nach einer erfolgreichen Authentifizierung heruntergeladen werden.

Eine andere Möglichkeit ist das vollständige Verhindern eines direkten Zugriffs auf die Datei. Diese wird vielmehr in einem von außerhalb nicht zugänglichen Ordner abgelegt, zum Beispiel `c:\downloads\`. Hierbei ist zu beachten, dass der von ASP.NET verwendete Benutzer-Account unbedingt explizite Leserechte auf das Verzeichnis benötigt.

Das im Listing gezeigte Web-Formular enthält eine `TextBox` zur Eingabe eines Passworts. Dieses wird beim Bestätigen des Formulars über ein `CustomValidatorControl` geprüft. Ist die Eingabe korrekt, wird eine Datei aus dem externen Download-Verzeichnis mittels der Methode `Response.WriteFile` an den Client gesendet.

Listing 11.8 download1.aspx

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    if(this.IsValid)
    {
```

```
string filename = "test.exe";
string filepath = "c:\\temp\\" + filename;

Response.ContentType = "application/x-msdownload" +
    ",filename=" + filename;
Response.AppendHeader("Content-Disposition",
    "attachment; filename=" + filename + "; alternative: inline");
Response.WriteFile(filepath);
Response.End();
}
}

void validatepw(object sender, ServerValidateEventArgs e)
{
    e.IsValid = (e.Value == "geheim");
}

</script>

<h1>Download</h1>

<p> Bitte geben Sie das Passwort ein, um den Download zu starten. </p>

<form runat="server">

    <p> Passwort:
        <ASP:TextBox id="tb_pw" runat="server" textmode="password"/>
    </p>

    <ASP:RequiredFieldValidator runat="server"
        ControlToValidate="tb_pw"
        display="dynamic"
        text="<p>Bitte geben Sie ein Passwort ein!</p>" />

    <ASP:CustomValidator runat="server"
        ControlToValidate="tb_pw"
        OnServerValidate="validatepw"
        display="dynamic"
        text="<p>Das Passwort ist leider falsch!</p>" />

    <ASP:Button runat="server" text="Download starten"
        onclick="bt_click" />

</form>
```

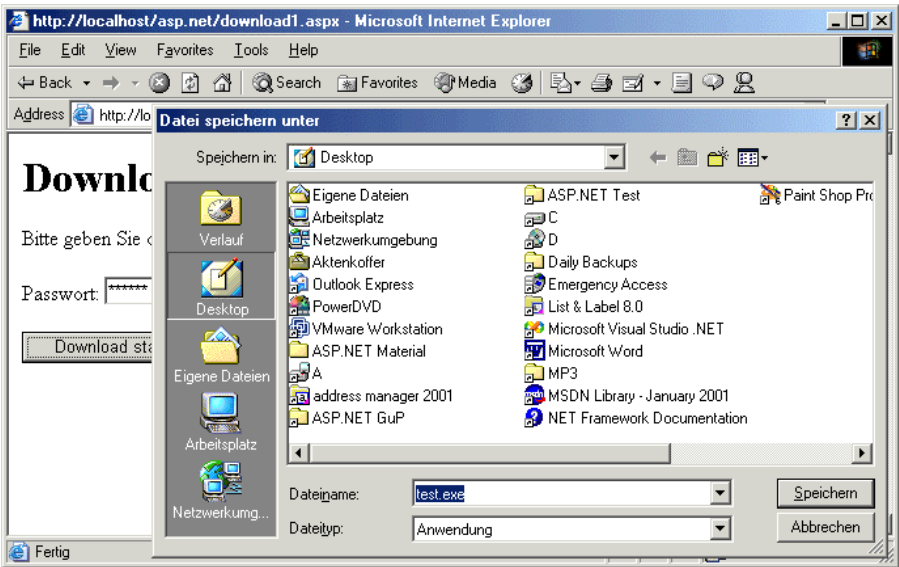


Abbildung 11.8 Der Download funktioniert nur nach Eingabe des Passworts.

Damit der Client die Datei korrekt verarbeiten kann, muss diesem die Art der Datei mitgeteilt werden. Es handelt sich um den MIME-Typ, der im Fall eines ausführbaren Programms „application/x-msdownload“ lautet. Dieser Wert wird der Eigenschaft `Response.ContentType` zugewiesen. Damit im Browser der korrekte Dateiname angezeigt wird, muss auch dieser explizit übergeben werden. Hierzu wird der Kopfzeileintrag „Content-Disposition“ benutzt.

 Beachten Sie bitte unbedingt den Aufruf von `Response.End` nach der Übertragung der Datei. Dieser Befehl ist notwendig, damit der Rest der Seite nicht weiter verarbeitet und an den Client gesendet wird. Ansonsten würde der HTML-Inhalt der Seite der Datei hinzugefügt werden.

Wenn Sie andere Dateien als Programme zum Download anbieten, müssen Sie den individuellen MIME-Typen übergeben. Die Tabelle zeigt einige davon. Weitere können Sie über die Registrierung (`registry.exe`) ermitteln. Im Zweig `HKEY_CLASSES_ROOT` finden Sie unterhalb der entsprechenden Dateieindung den Eintrag „Content-Type“, der dem MIME-Typen entspricht.

Tabelle 11.1 Wichtige MIME-Typen

Dateieindung	Dateiart	MIME-Type
doc	Word-Dokument	application/msword
exe	Programm	application/x-msdownload
gif	Grafik	image/gif
jpg	Grafik	image/jpeg

Dateiendung	Dateiart	MIME-Type
pdf	PDF	application/pdf
png	Grafik	image/png
txt	Text	text/plain
xml	XML	text/xml
zip	Archiv	application/x-compressed



Wann immer die Datei heruntergeladen wird, wird zuvor immer ihr Quellcode durchlaufen. Ein direkter Zugriff ist nicht mehr möglich. Sie können daher auf einfache Weise einen Datenbankzähler einrichten, der über jeden Download Buch führt. Auf diese Weise erhalten Sie mit wenigen Handgriffen eine Download-Statistik.

11.10 ... Links von bestimmten Seiten verhindern?

Es gehört zu einer modernen Unsitte, Inhalte von fremden Seiten zu übernehmen. Besonders dreist sind diejenigen, die die fremden Inhalte einfach in einem Frameset anzeigen und so als ihre eigenen ausgeben. Mit ein paar Tricks können Sie ASP.NET dazu veranlassen, diesen dreisten Content-Klauern die Übernahme wenn nicht ganz zu verhindern, so doch zumindest zu erschweren.

Ziel des Projekts soll es sein, den Aufruf von Seiten über externe Links zu verhindern. Die notwendigen Informationen liefert die Referer-Kopfzeile des Protokolls HTTP. Hier wird angegeben, woher die Seite angelinkt wurde. Bei jedem Aufruf soll dieser Eintrag überprüft und die Anzeige der Seite gegebenenfalls unterbunden werden.

Die technische Basis liefert ein `HttpModule`. Hierbei handelt es sich um den Quasi-Nachfolger der ISAPI-Filter. Bei jedem Aufruf an die ASP.NET-Engine läuft das `HttpModule` nebenher und kann falls notwendig in die Bearbeitung eingreifen beziehungsweise diese verhindern.

Das Listing zeigt ein solches `HttpModule` in Form einer C#-Quellcode-Datei. Die Klasse `MyHttpModule` unterstützt die Schnittstelle `IHttpModule` und implementiert deren beiden Methoden. `Init` wird zur Initialisierung des Moduls verwendet. Hier wird das Ereignis `BeginRequest` angemeldet. Dieses wird immer dann aufgerufen, wenn eine neue Client-Anfrage zur Bearbeitung ansteht.

Die Ereignisbehandlung fragt zunächst den aktuellen Hostnamen ab. Nun wird überprüft, ob die Seite über einen Link erreicht wurde. Ist dies der Fall, wird der lokale Host im Referrer-String gesucht. Wird dieser nicht gefunden, muss es sich

um einen externen Link handeln. In diesem Fall wird die Bearbeitung der Anfrage vollständig abgebrochen und mit einem entsprechenden Statuscode und -text an den Client gesendet. Die Seite kann nicht aufgerufen werden.

Listing 11.9 ReferrerCheck1.cs

```
using System;
using System.IO;
using System.Web;

namespace PAL.HttpModule
{
    class MyHttpModule : IHttpModule
    {

        public void Init(HttpApplication app)
        {
            app.BeginRequest += new EventHandler(BeginRequest);
        }

        public void BeginRequest(object sender, EventArgs e)
        {
            HttpApplication app = (HttpApplication) sender;
            HttpRequest Request = app.Request;
            HttpResponse Response = app.Response;

            string host = Request.Url.Host;

            if(Request.UrlReferrer != null &&
                Request.UrlReferrer.AbsoluteUri.IndexOf(host) == -1)
            {
                Response.StatusCode = 403;
                Response.Write("<h1>Verlinkung nicht erlaubt!</h1>");
                Response.End();
            }
        }

        public void Dispose()
        {
        }
    }
}
```

Das HttpModule muss nun als DLL kompiliert werden. Hierzu wird der Kommandozeilen-Kompiler `csc.exe` verwendet. Anschließend wird die erzeugte DLL im bin-Verzeichnis der Web-Applikation abgelegt.

```
csc /t:library /r:System.dll,System.Web.dll ReferrerCheck1.cs
```

Damit das neue `HttpModule` verwendet wird, muss es in der Konfigurationsdatei `web.config` hinterlegt werden. Hierzu wird im Abschnitt `httpModules` ein neuer Eintrag angelegt, der folgendem Schema genügen muss:

```
<add type="<Namespace+Klassenname>,<DLL ohne Endung>" name="<Name>" />
```

Für das gezeigte `HttpModule` sieht die Konfiguration beispielsweise wie folgt aus:

Listing 11.10 Web.config

```
<configuration>
  <system.web>
    <httpModules>
      <add type="PAL.HttpModule.MyHttpModule,ReferrerCheck1"
        name="MyHttpModule" />
    </httpModules>
  </system.web>
</configuration>
```

Ist alles konfiguriert, steht dem Einsatz des neuen Moduls nichts mehr im Wege. Sofern Sie eine Seite direkt im Browser aufrufen, wird diese angezeigt. Das gleiche Ergebnis erzielen Sie bei einer Verlinkung innerhalb der Website. Rufen Sie eine Seite jedoch über einen Link von einem anderen Server aus auf, so wird die Bearbeitung abgebrochen und mit einer Fehlermeldung quittiert. Sie sehen diese in der Abbildung. Die Verlinkung von außerhalb wurde erfolgreich verhindert!



Abbildung 11.9 Externe Links sind nicht erlaubt.



Beachten Sie, dass mit dem vorliegenden `HttpModule` keinerlei Verlinkung von außen möglich ist. In der Realität ist dies Quatsch, denn zumindest die Startseite sollte auch über externe Links zu erreichen sein.

Um dies zu erreichen, können Sie entweder eine Bedingung für das Verlinken der Startseite implementieren oder jede ungültige Anfrage einfach auf selbige umleiten. Hierzu können Sie innerhalb der aktuellen Web-Applikation die Methode `Context.RewritePath` verwenden. Ausgehend vom vorherigen Listing sieht dies wie folgt aus:

Listing 11.11 ReferrerCheck2.cs

```
...  
  
if (Request.UrlReferrer != null &&  
    Request.UrlReferrer.AbsoluteUri.IndexOf(host) == -1)  
{  
    Context.RewritePath("default.aspx");  
}  
  
...
```

Viel extremer als bei Inhalten bedienen sich die wenig kreativen Inhaltsklauer bei Grafiken. Oft werden auch diese einfach von ihrer Ursprungsseite in die neue Web-site-Umgebung eingebunden. Selbstverständlich können Sie auch dieser Übernahme einen Riegel vorschieben. Zunächst müssen Sie sicherstellen, dass Anfragen auf die Dateiendung `gif` von den Internet Information Services an die ASP.NET-Engine weitergeleitet werden. Wie dies geht, erfahren Sie im Rezept „... alle Dateien mit Forms Authentication schützen?“.

Ist die Endung registriert, werden alle Anfragen für das Bildformat GIF über das `HttpModule` abgewickelt. Dieses muss nun derart modifiziert werden, dass es nur auf diese Dateiendung reagiert. Einfache Zeichenoperationen erledigen dies. Im Unterschied zu Seitenabfragen sollte bei Bildern nicht einfach ein Text zurückgeliefert werden. Stattdessen können Sie eine Grafik an den Client senden, die ihn über den Content-Klau informiert. Die Methode `Response.WriteFile` übernimmt diese Aufgabe für Sie.

Listing 11.12 ReferrerCheck3.cs

```
if (Request.UrlReferrer != null &&  
    Request.UrlReferrer.AbsoluteUri.IndexOf(host) == -1)  
{  
    if (Path.GetExtension(Request.Path).ToLower() == ".gif")  
    {  
        Response.StatusCode = 200;  
        Response.ContentType = "image/gif";  
        Response.WriteFile("contentklau.gif");  
        Response.End();  
    }  
}
```

Die Abbildung zeigt, dass die externe Einbindung der Grafik nicht erlaubt wird. Das `HttpModule` blendet statt des Fotos der Paramount Studios einen einfachen wie deutlichen Hinweis ein.



Abbildung 11.10 Dem Klau von Bildern wird ein Riegel vorgeschoben.



Sofern Sie das zweite beziehungsweise dritte Listing ausprobieren möchten, müssen Sie zuvor unbedingt die Einbindung des Moduls in der Konfigurationsdatei `web.config` aktualisieren.

11.11 ... einen Stream oder eine Datei verschlüsseln?

Das .NET Framework stellt umfangreiche Klassen zur Verschlüsselung von Daten zur Verfügung. Diese enthalten zahlreiche, gängige Kryptografiealgorithmen. Hierzu zählen insbesondere folgende:

- Asymmetrische Algorithmen: DAS und RSA
- Hash-Algorithmen: MD5, SHA1, SHA265, SHA384 und SHA512
- Symmetrische Algorithmen: DES, RC2, Rijndael und TripeDES

Alle benötigten Klassen sind unterhalb des Namespaces `System.Security.Cryptography` zu finden. Vor der Verwendung muss dieser explizit eingebunden werden:

```
<% @Import Namespace="System.Security.Cryptography" %>
```

Die beiden nachfolgenden Beispiele beziehen sich auf den symmetrische Algorithmus DES (Data Encryption Standard). Dieser wird zunächst verwendet, um eine Datei zu kodieren.

Die Klasse `DESCryptoServiceProvider` implementiert den Provider zur Ver- und Entschlüsselung von beliebigen Streams mit Hilfe von DES. Im Listing wird dieser Klasse über die Eigenschaft `Key` der zu verwendende Schlüssel übergeben. Dieser muss 64 Bit, also 8 Byte und somit 8 Zeichen lang sein.

Die von der abstrakten Basis `Stream` abgeleitete Klasse `CryptoStream` wird unabhängig vom Algorithmus zur Verschlüsselung verwendet. Neben dem Ausgabe-Stream wird im Konstruktor der zu nutzende Provider übergeben. Bei DES wird dieser mittels der Methode `CreateEncryptor` instanziiert.

Die Klasse `CryptoStream` arbeitet wie ein Proxy. Alle Daten, die in den Stream geschrieben werden, werden durch den Verschlüsselungsprovider kodiert. Das Ergebnis wird in den, im Konstruktor übergebenen, Ausgabe-Stream weitergereicht. Im Beispiel ist dies ein `FileStream`, der auf eine neue Datei verweist.

Ist eine Instanz der Klasse `CryptoStream` erstellt, können Daten zur Kodierung hineingeschrieben werden. Auch dies erfolgt auf Basis eines `FileStreams`, der im Beispiel komplett durchlaufen wird. Der Inhalt der Datei wird also kodiert und in einer zweiten Datei abgelegt.

Listing 11.13 Encrypt1.aspx

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="System.Security.Cryptography" %>

<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    FileStream instream = new FileStream(Server.MapPath("file1.txt"),
        FileMode.Open, FileAccess.Read);
    FileStream ostream = new FileStream(Server.MapPath("file2.txt"),
        FileMode.OpenOrCreate, FileAccess.Write);

    DESCryptoServiceProvider des = new DESCryptoServiceProvider();
    string key = "abcdefgh";
    des.Key = GetBytesFromString(key);
    des.IV = des.Key;

    CryptoStream cryptostream = new CryptoStream(ostream,
        des.CreateEncryptor(), CryptoStreamMode.Write);

    long pos = 0;
    long length = instream.Length;
    while(pos < length)
    {
        byte[] bytes = new byte[100];
        int lenread = instream.Read(bytes, 0, 100);
        cryptostream.Write(bytes, 0, lenread);
    }
}
```

```
pos += lenread;
}

cryptostream.Close();
outstream.Close();
instream.Close();
}

byte[] GetBytesFromString(string value)
{
    return(ASCIIEncoding.ASCII.GetBytes(value));
}

</script>

Die Datei wurde kodiert!
```

Lange Rede, kurzer Sinn. Das Listing zeigt die Verschlüsselung einer bestehenden Datei mit Hilfe des Data Encryption Standard. Das Ergebnis wird in einer zweiten kodierte Datei abgelegt. Die Abbildung zeigt die beiden Dateien im Vergleich. Der Größenunterschied liegt bei genau vier Byte, die die verschlüsselte Datei größer ist als das Original.



Die verschiedenen Algorithmen haben meist vorgegebene Schlüsselstärken. Beim hier eingesetzten Verfahren DES sind dies 64 Bit. Beachten Sie bitte, dass die Angaben immer in Bit erfolgen. Die Anzahl der möglichen Zeichen ergibt sich also aus der Division durch 8, denn ein Byte besteht wie bekannt aus acht Bits.

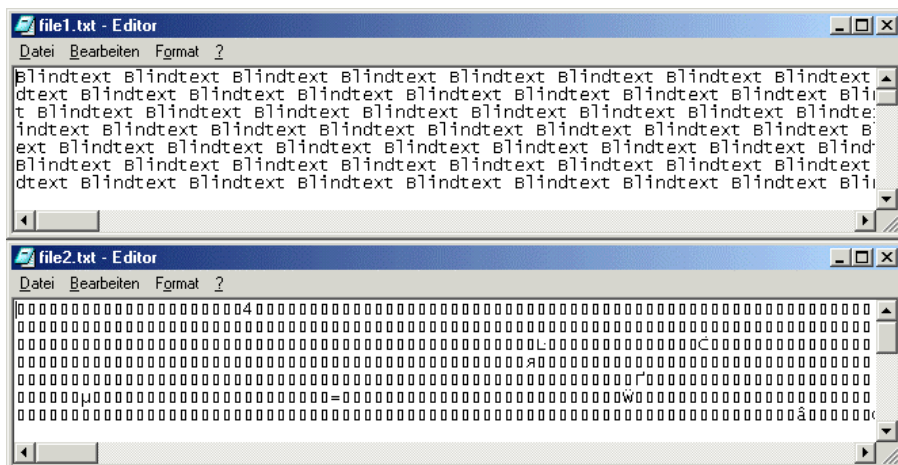


Abbildung 11.11 Der gleiche Text vor und nach der Verschlüsselung.

Das Beispiel zeigt die Verwendung eines `FileStream` für Eingabe und Ausgabe. Durch die offene Architektur der Klasse `CryptoStream` können Sie jedoch auch alle anderen (eigenen) Klassen verwenden, die von der abstrakten Basis `Stream` abgeleitet wurden.

Über `MemoryStream` können Sie die Verschlüsselung zum Beispiel ausschließlich im Arbeitsspeicher vornehmen. Sie können anschließend direkt wieder auf die verschlüsselten Daten zugreifen. Hierzu ist es jedoch notwendig, dass die Arbeit der Klasse `CryptoStream` mittels `FlushFinalBlock` abgeschlossen und der Cursor des Speicher-Streams mittels `Seek` an den Anfang gesetzt wird.

Das Listing zeigt das Verschlüsseln von Daten im Arbeitsspeicher. Eine übergebene Zeichenkette wird kodiert und das Ergebnis ebenfalls als Zeichenkette zurückgeliefert.

Listing 11.14 Encrypt2.aspx

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="System.Security.Cryptography" %>

<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    string key = "abcdefgh";
    string crypt = Crypt("Hallo Welt!", key);
    Response.Write(crypt);
}

string Crypt(string data, string key)
{
    MemoryStream memorystream = new MemoryStream();

    DESCryptoServiceProvider des = new DESCryptoServiceProvider();
    des.Key = GetBytesFromString(key);
    des.IV = des.Key;

    CryptoStream cryptostream = new CryptoStream(memorystream,
        des.CreateEncryptor(), CryptoStreamMode.Write);
    StreamWriter writer = new StreamWriter(cryptostream);
    writer.Write(data);
    cryptostream.FlushFinalBlock();
    memorystream.Seek(0, SeekOrigin.Begin);

    StreamReader reader = new StreamReader(memorystream);
    string crypt = reader.ReadToEnd();

    cryptostream.Close();
    memorystream.Close();
}
```

```

    return(crypt);
}

byte[] GetBytesFromString(string value)
{
    return(ASCIIEncoding.ASCII.GetBytes(value));
}

</script>

```



Beachten Sie bitte, dass die Darstellung der verschlüsselten, binären Daten in einer Zeichenkette nicht fehlerfrei möglich ist. Nach der Übertragung an den Client sind die Daten nicht mehr zu entschlüsseln.

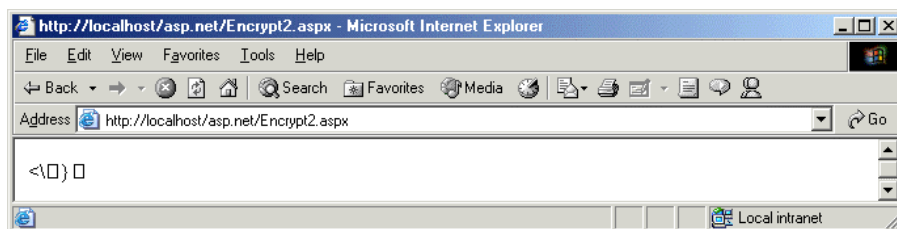


Abbildung 11.12 Hallo Welt einmal anders

Selbstverständlich können Sie die verschlüsselte Datei auch wieder dekodieren. Hierzu benötigen Sie lediglich das korrekte Passwort. Der benötigte Quellcode entspricht nahezu vollständig der Kodierung aus dem ersten Beispiel weiter oben. Statt dem Encryption Provider wird jedoch ein Decryption Provider benötigt. Dieser wird über die Methode `CreateDecryptor` der Klasse `DESCryptoServiceProvider` instanziiert und an den `CryptoStream`-Konstruktor übergeben:

```

...

CryptoStream cryptostream = new CryptoStream(outstream,
    des.CreateDecryptor(), CryptoStreamMode.Write);

...

```

Nach dem Aufruf des Beispiels enthält die dritte Datei `file3.txt` das Eins-zu-eins-Abbild der Originaldatei.

Analog zu der hier vorgestellten DES-Verschlüsselung können Sie auch die alternativen Algorithmen verwenden. Durch das offene Schema können Sie zudem individuelle Implementierungen einsetzen. Interessante Kandidaten wären beispielsweise

Blowfish oder Twofish, möglicherweise sogar mit optionaler MIME-Kodierung, so dass Sie das Ergebnis direkt weiter beispielsweise in einer Email weiterverarbeiten können.

11.12 ... eine Email verschlüsseln?

Natürlich können Sie mit den gegebenen Möglichkeiten des .NET Frameworks auch Emails verschlüsseln. Mangels Möglichkeit zur direkten Übertragung von binären Daten seitens entsprechender Protokolle (SMTP, POP3) sind hier jedoch kleinere Änderungen gegenüber der Verschlüsselung von Dateien notwendig.

Zunächst einmal finden Sie im folgenden Listing den Versand einer normalen, unverschlüsselten Email. Einer neuen Instanz der Klasse `MailMessage` werden notwendige Daten wie Absender, Empfänger, Betreff und Text als Zeichenketten zugewiesen. Die statische Methode `SmtpMail.Send` sorgt anschließend dafür, dass die Nachricht über den Windows SMTP-Dienst versendet wird.

Listing 11.15 Mail1.aspx

```
<% @Import Namespace="System.Web.Mail" %>

<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    MailMessage mail = new MailMessage();
    mail.From = "pl@p-l.de";
    mail.To = "pl@p-l.de";
    mail.Subject = "Hallo Welt";
    mail.Body = "Hallo Welt!\nIch wollte nur mal Guten Tag sagen und
        fragen, wie es so geht. Das war es auch schon.\nCiao";
    SmtpMail.Send(mail);
}

</script>

Die Email wurde versendet!
```

Auf Basis dieses Listings soll nun eine mit DES verschlüsselte Email generiert werden. Hierzu wird eine Ableitung der Klasse mit dem Namen `CryptedMailMessage` angelegt. Im Konstruktor wird der zur Verschlüsselung zu verwendende Code übergeben. Anschließend wird die Eigenschaft `Body` überschrieben. Nur der hier zugewiesene Inhalt soll verschlüsselt werden. Die anderen Werte wie Absender und Empfänger müssen aus recht offensichtlichen Gründen unverschlüsselt bleiben.

Da die Mitglieder der Klasse `MailMessage` leider nicht als `virtual` gekennzeichnet sind, muss `Body` explizit mit dem Schlüsselwort `new` ausgetauscht werden. Im `set-Accessor` wird der Text zunächst mit Hilfe der Methode `Crypt` verschlüsselt und anschließend mittels `BreakText` in Zeilen à 76 Zeichen zerlegt. Dies stellt sicher, dass der Inhalt korrekt mittels SMTP übertragen werden kann.

Die Verschlüsselung erfolgt auf Basis von DES und ist dem Beispiel im vorherigen Rezept „... einen Stream oder eine Datei verschlüsseln?“ recht ähnlich. Da die Daten als Zeichenkette und nicht als binärer Stream übertragen werden, muss eine Konvertierung stattfinden. Hierzu wird der Inhalt der `MemoryStream`-Instanz zunächst in ein `byte-Array` und daraus mittels der statischen Methode `Convert.ToBase64String` in eine Base64-kodierte Zeichenkette umgewandelt.

Listing 11.16 Mail2.aspx

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="System.Security.Cryptography" %>
<% @Import Namespace="System.Web.Mail" %>

<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    CryptedMailMessage mail = new CryptedMailMessage("abcdefgh");
    mail.From = "pl@p-l.de";
    mail.To = "pl@p-l.de";
    mail.Subject = "Hallo Welt (verschlüsselt)";
    mail.Body = "Hallo Welt!\nIch wollte nur mal Guten Tag sagen und
        fragen, wie es so geht. Das war es auch schon.\nCiao";
    Smtplib.Send(mail);
}

public class CryptedMailMessage : MailMessage
{
    private string m_Body;
    private string m_Key;

    public CryptedMailMessage(string key)
    {
        m_Key = key;
    }

    public new string Body
    {
        get { return(m_Body); }
        set
        {
            string body = value;
```



```
m_Body = body;
body = Crypt(body, m_Key);
body = BreakText(body, 76);
base.Body = body;
}
}

private string Crypt(string data, string key)
{
    MemoryStream memorystream = new MemoryStream();

    DESCryptoServiceProvider des = new DESCryptoServiceProvider();
    des.Key = GetBytesFromString(key);
    des.IV = des.Key;

    CryptoStream cryptostream = new CryptoStream(memorystream,
        des.CreateEncryptor(), CryptoStreamMode.Write);
    StreamWriter writer = new StreamWriter(cryptostream);
    writer.Write(data);
    writer.Flush();
    cryptostream.FlushFinalBlock();

    memorystream.Seek(0, SeekOrigin.Begin);
    byte[] bytes = new byte[memorystream.Length];
    memorystream.Read(bytes, 0, bytes.Length);

    string crypt = Convert.ToBase64String(bytes);
    return(crypt);
}

private byte[] GetBytesFromString(string value)
{
    return(ASCIIEncoding.ASCII.GetBytes(value));
}

private string BreakText(string data, int linelength)
{
    string ret = string.Empty;
    while(data.Length > linelength)
    {
        ret += data.Substring(0, linelength) + "\n";
        data = data.Substring(linelength);
    }
    ret += data;
    return(ret);
}
}

</script>
```

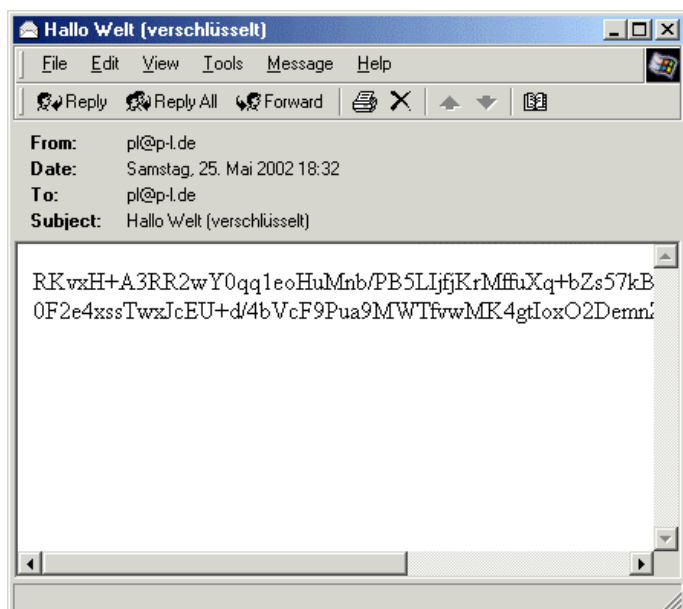
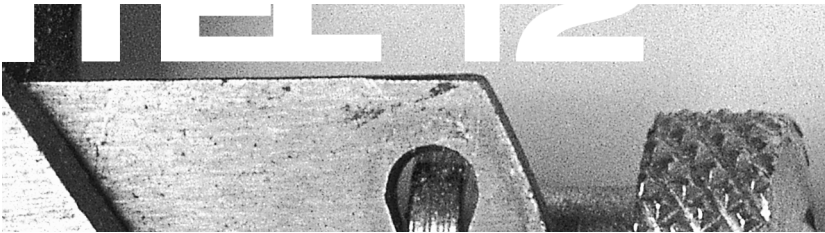


Abbildung 11.13 Die Email wurde erfolgreich verschlüsselt.

12

Grafik



Wie kann ich ...

12 Grafik

Bunte Bilder haben das Internet berühmt gemacht, und ohne sie würden vermutlich weder Sie noch ich uns heute so intensiv mit dem Word Wide Web beschäftigen. In diesem Kapitel dreht sich alles um die Grafiken, die Sie anzeigen und sogar selbst erzeugen können.

Einbindung

Die allgemeinen Klassen von GDI+ zur Bearbeitung von Grafiken werden bei .NET unterhalb des Namespaces `System.Drawing` angesammelt. Bevor Sie die Klassen aus den folgenden Rezepten nutzen können, müssen Sie in aller Regel den Namespace zunächst einbinden:

```
<% @Import Namespace="System.Drawing" %>
```

Weiter spezialisierte Klassen sind in den ungeordneten Namespaces `Drawing2D`, `Imaging` und `Text` hinterlegt:

```
<% @Import Namespace=" System.Drawing.Drawing2D" %>  
<% @Import Namespace=" System.Drawing.Imaging" %>  
<% @Import Namespace=" System.Drawing.Text" %>
```

Die in diesem Kapitel gezeigten Bilder stammen größtenteils aus Kalifornien, darunter sind bekannte Wahrzeichen aus Los Angeles und San Francisco.

12.1 ... ein Bild einladen und im Browser ausgeben?

Ein Pixel-Bild wird von der Klasse `Bitmap` repräsentiert. Um ein Bild von der Festplatte einzuladen, übergeben Sie den gewünschten Dateinamen dem Konstruktor der Klasse. Unterstützt werden alle gängigen Formate, darunter auch GIF, JPEG und PNG.

Um ein Bild an den Client zu senden, speichern Sie dieses im `OutputStream` der `Response`-Klasse ab. Zusätzlich sollten Sie der Eigenschaft `ContentType` den entsprechenden MIME-Typen zuweisen. Anschließend liefert die angeforderte Seite ein Bild an den Client zurück. Weitere Ausgaben innerhalb der Seite sind selbstverständlich *nicht* möglich.

Listing 12.1 Bitmap1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Request.PhysicalApplicationPath + "paramount.gif";
    Bitmap b = new Bitmap(file);
    Response.ContentType = "image/gif";
    b.Save(Response.OutputStream, ImageFormat.Gif);
}

</script>
```

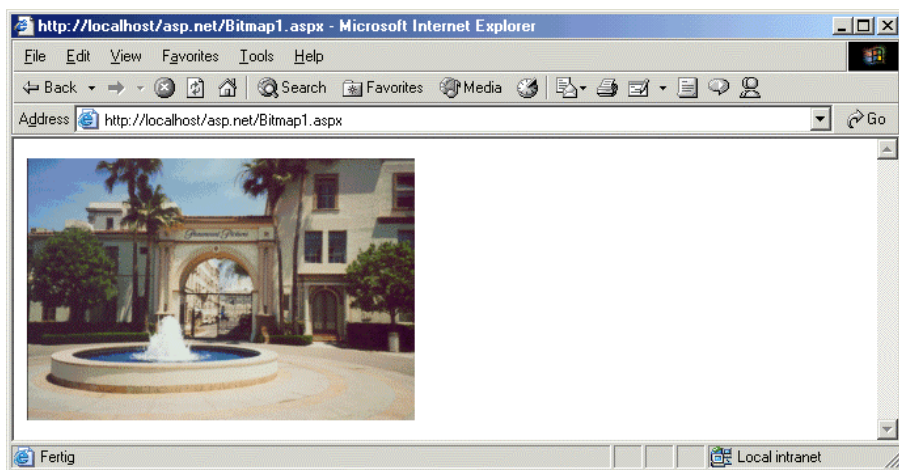


Abbildung 12.1 Das Bild wurde aus dem Server eingelesen und an den Client gesendet.

12.2 ... Bildformate konvertieren?

Ein im Speicher befindliches Bild können Sie in einem beliebigen unterstützten Format an den Client senden. Im Speicher befindet sich in jedem Fall nur das jeweilige Bitmap-Abbild, also eine Pixel-Grafik. Das folgende Listing entspricht dem vorherigen, überträgt das ursprüngliche GIF-Bild jedoch im Format JPEG.

Listing 12.2 Bitmap2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Request.PhysicalApplicationPath + "paramount.gif";
    Bitmap b = new Bitmap(file);
    Response.ContentType = "image/jpeg";
    b.Save(Response.OutputStream, ImageFormat.Jpeg);
}

</script>
```

12.3 ... die Größe und Auslösung eines Bildes ermitteln?

Einmal mit einem Bild instanziiert liefert die Klasse `Bitmap` eine Reihe von Informationen über die geladene Grafik. So können Sie zum Beispiel die Größe und die Auflösung abfragen.

Das Listing zeigt die Ausgabe einer zur Verfügung stehenden Information, darunter auch die Anzahl der Paletteneinträge sowie die darin hinterlegten Farben.

Listing 12.3 Bitmap3.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Request.PhysicalApplicationPath + "paramount.gif";
    Bitmap b = new Bitmap(file);

    Response.Write("<h2> Allgemeine Informationen </h2>");
    WriteLine("Dateiname:", file);
    WriteLine("Format:", b.RawFormat);
    WriteLine("Breite:", b.Size.Width);
    WriteLine("Höhe:", b.Size.Height);
    WriteLine("Hor. Auflösung:", b.HorizontalResolution);
    WriteLine("vert. Auflösung:", b.VerticalResolution);
}
```

```
Response.Write("<h2> Palette </h2>");
WriteLine("Paletteneinträge:", b.Palette.Entries.Length);

foreach(Color c in b.Palette.Entries)
{
    Label lbl = new Label();
    lbl.Text = c.Name + "<br>";
    lbl.ForeColor = Color.White;
    lbl.BackColor = c;
    this.Controls.Add(lbl);
}

void WriteLine(params object[] values)
{
    foreach(object value in values)
        Response.Write(value.ToString() + " ");
    Response.Write("<br>");
}

</script>
```

12.4 ... automatisch Thumbnails von Bildern erzeugen?

Die Klasse `Bitmap` verfügt über eine Methode `GetThumbnailImage` über die sich eine verkleinerte Vorschau des darunter liegenden Bildes erstellen lässt. Die gewünschten Maße werden als Parameter übergeben. Soll das Bild proportional verkleinert werden, müssen die Proportionen vorab manuell berechnet werden.

Das Listing zeigt die Erstellung eines Thumbnails. Um die genannte Methode wurde eine weitere `GetThumbnail` geschrieben, der das gewünschte Bild sowie Breite oder Höhe des gewünschten Vorschaubildes übergeben werden. Die Methode rechnet das jeweilige Gegenstück automatisch proportional aus.

Listing 12.4 Bitmap4.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Request.PhysicalApplicationPath +
        "vivendi-universal.gif";
    Bitmap b = new Bitmap(file);
    Bitmap thumb = GetThumbnail(b, 150, 0);
}
```



```

Response.ContentType = "image/jpeg";
thumb.Save(Response.OutputStream, ImageFormat.Jpeg);
}

Bitmap GetThumbnail(Bitmap b, int w, int h)
{
    if(w == 0 && h == 0)
    {
        w = b.Size.Width;
        h = b.Size.Height;
    }
    else if(w == 0)
        w = (int) (((float) h) / b.Size.Height * b.Size.Width);
    else if(h == 0)
        h = (int) (((float) w) / b.Size.Width * b.Size.Height);

    return((Bitmap) b.GetThumbnailImage(w, h, null, IntPtr.Zero));
}

</script>

```

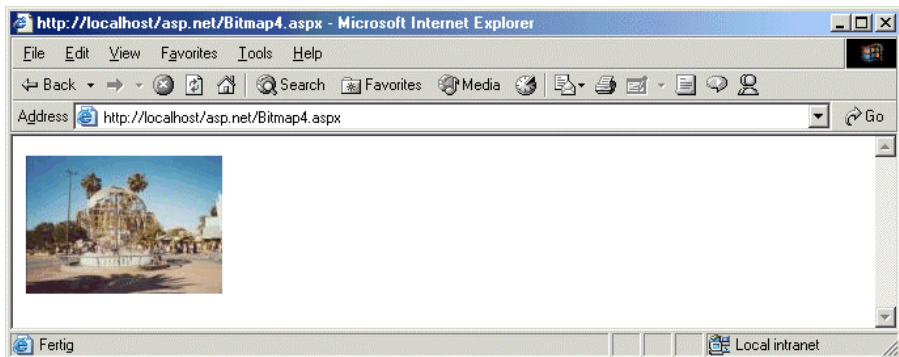


Abbildung 12.2 Die Universal-Studios ganz klein.

12.5 ... Thumbnails automatisch erstellen und zwischenspeichern?

Oftmals sollen dem Benutzer Vorschaubilder aller Grafiken in einem Verzeichnis angezeigt werden. So kann der Benutzer die für ihn relevante Abbildung schnell finden und per Klick in Originalgröße öffnen.

Eine derartige Realisierung besteht aus zwei Komponenten. Einerseits wird eine Seite benötigt, die alle Bilder anzeigt und Links zu den entsprechenden Originalen anbindet. Die zweite Seite muss eine verkleinerte Ansicht jeweils eines der Bilder liefern und wird über das `src`-Attribut eines `img`-Tags referenziert.

Nachfolgend sehen Sie die Übersichtsseite. Aus dem aktuellen Verzeichnis wird ein `string`-Array mit den Dateinamen aller GIF-Bilder ermittelt. Dieses Array wird einem `DataList`-Control als Datenquelle übergeben. Eine Vorlage sorgt für die Darstellung von Link und Bild.

Listing 12.5 ShowThumbnails.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string dir = Path.GetDirectoryName(Request.PhysicalApplicationPath);
    string[] files = Directory.GetFiles(dir, "*.gif");
    list.DataSource = files;
    list.DataBind();
}

</script>

<ASP:DataList id="list" runat="server"
RepeatColumns="2">

<ItemTemplate>
<a href="<%# Path.GetFileName(Container.DataItem.ToString()) %>"
target="_blank">
    
</a>
</ItemTemplate>

</ASP:DataList>
```

Die zweite Seite hat die Aufgabe, ein Vorschaubild der im Query-String übergebenen Datei zu erzeugen. Zur Optimierung wird das einmal erzeugte Thumbnail abgespeichert. Beim nächsten Aufruf wird das Bild nicht mehr dynamisch generiert, sondern schlichtweg von der Festplatte geöffnet. Gerade bei hochauflösenden und einer Vielzahl von Screenshots resultiert dieses Vorgehen in einer drastischen Geschwindigkeitsverbesserung.

Listing 12.6 GetThumbnail.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Bitmap thumb;
    string file = Request.PhysicalApplicationPath +
        Request.QueryString["file"];
    string thumbfile = Request.PhysicalApplicationPath +
        Path.GetFileNameWithoutExtension(file) + ".thm";

    if (File.Exists(thumbfile))
    {
        thumb = new Bitmap(thumbfile);
    }
    else
    {
        Bitmap b = new Bitmap(file);
        thumb = GetThumbnail(b, 150, 0);
        thumb.Save(thumbfile, ImageFormat.Jpeg);
    }

    Response.ContentType = "image/jpeg";
    thumb.Save(Response.OutputStream, ImageFormat.Jpeg);
    thumb.Dispose();
}

Bitmap GetThumbnail(Bitmap b, int w, int h)
{
    if (w == 0 && h == 0)
    {
        w = b.Size.Width;
        h = b.Size.Height;
    }
    else if (w == 0)
        w = (int) (((float) h) / b.Size.Height * b.Size.Width);
    else if (h == 0)
        h = (int) (((float) w) / b.Size.Width * b.Size.Height);

    return ((Bitmap) b.GetThumbnailImage(w, h, null, IntPtr.Zero));
}

</script>

```

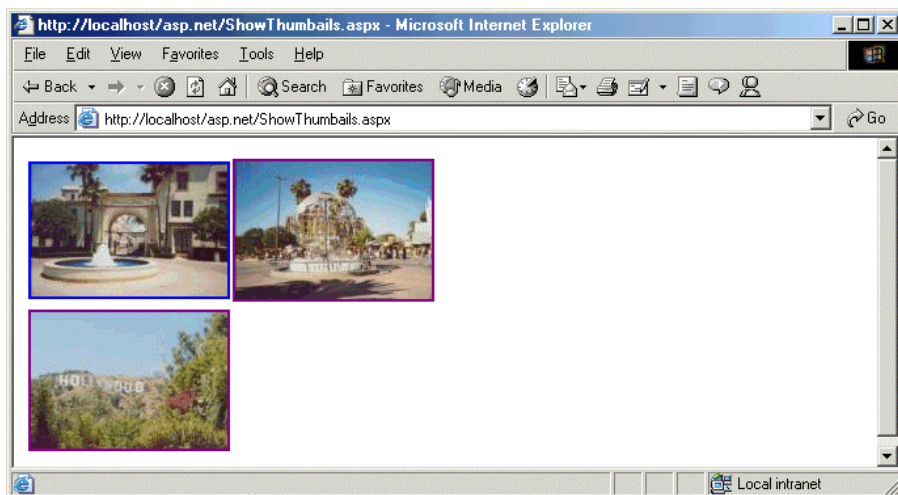


Abbildung 12.3 Alle GIF-Bilder im Verzeichnis übersichtlich angezeigt.



Achten Sie darauf, dass der Benutzer-Account „ASPNET“ Dateianlage- und Schreibrechte auf das entsprechende Verzeichnis besitzen muss, damit die Thumbnails abgespeichert werden können.

12.6 ... Bilder von Benutzern uploaden und ablegen?

Gerade viele Communities bieten den Upload von Dateien an. Ob es nun die Urlaubserinnerungen oder ein Porträt für das Benutzerprofil ist, Bilder bedeuten einen angenehmen und oftmals witzigen Zugewinn an Interaktivität.

Mit ASP.NET ist ein derartiges Feature schnell realisiert. Benötigt wird ein `HtmlInputFile-Control` zum Upload der Datei und die bereits beschriebene `Bitmap-Klasse`. Das ist im Grunde schon alles.

Listing 12.7 UploadBitmap1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    message.Text = string.Empty;
}

void Upload_Click(object sender, EventArgs e)
{
    HttpPostedFile pfile = tb_file.PostedFile;
    if(pfile == null || pfile.FileName.Length == 0)
```

```
message.Text = "Sie haben keine Datei ausgewählt ...";
else
{
    try
    {
        string localfile = Request.PhysicalApplicationPath +
            Path.GetFileName(pfile.FileName);

        Bitmap b = GetThumbnail(new Bitmap(pfile.InputStream), 200, 0);
        b.Save(localfile, ImageFormat.Jpeg);

        img.Src = Path.GetFileName(pfile.FileName);
        phimg.Visible = true;
    }
    catch
    {
        message.Text = "Die Datei ist ungültig ...";
    }
}
}
...

</script>

<form runat="server" enctype="multipart/form-data">

    <input type="file" id="tb_file" runat="server">
    <p> <ASP:Label id="message" runat="server"/> </p>
    <p> <ASP:Button runat="server" Text="Uploaden"
        OnClick="Upload_Click"/> </p>

    <ASP:Placeholder id="phimg" runat="server" Visible="false">
        <p> Ihr Bild sieht so aus: </p>
        <img id="img" runat="server"/>
    </ASP:Placeholder>

</form>
```

Das Bitmap wird nach dem Upload direkt mit dem InputStream des Eingabefeldes instanziiert, verkleinert und lokal abgespeichert und kann nun für jedermann im Browser angezeigt werden.

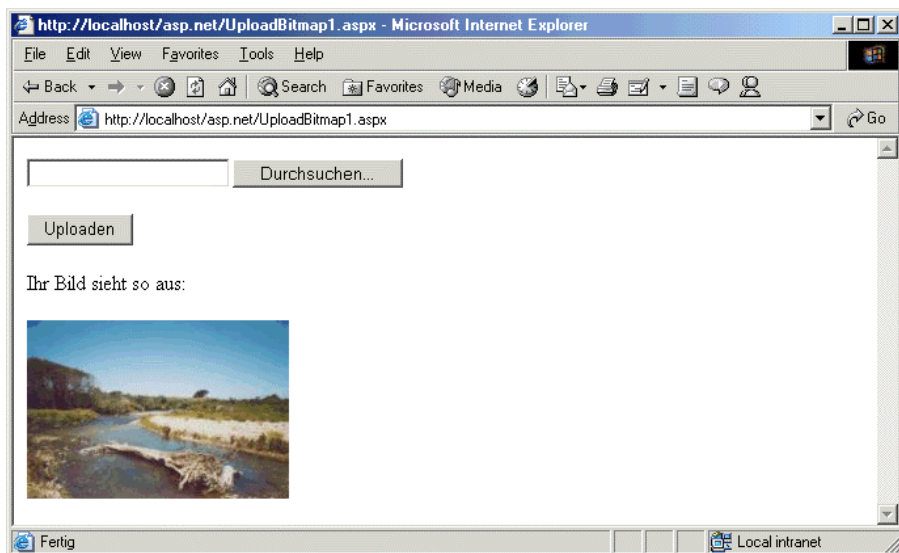


Abbildung 12.4 Eben noch auf Ihrer Festplatte und jetzt im Internet: Big Sur Regional Park

12.7 ... ein Bild rotieren?

Das Rotieren und Spiegeln von Bildern übernimmt die Methode `RotateFlip` der Klasse `Bitmap`. Hier können Sie einen Wert der Enumeration `RotateFlipType` übergeben, die eine Rotation in 90°-Schritten sowie X- und Y-Spiegelungen erlaubt. Das Listing zeigt ein Beispiel, dessen Ergebnis Sie in der Abbildung sehen (achten Sie auf den HOLLYWOOD-Text).

Listing 12.8 `RotateFlip1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Request.PhysicalApplicationPath + "hollywood-sign.gif";
    Bitmap b = new Bitmap(file);

    b.RotateFlip(RotateFlipType.Rotate270FlipX);

    Response.ContentType = "image/gif";
    b.Save(Response.OutputStream, ImageFormat.Gif);
}

</script>
```

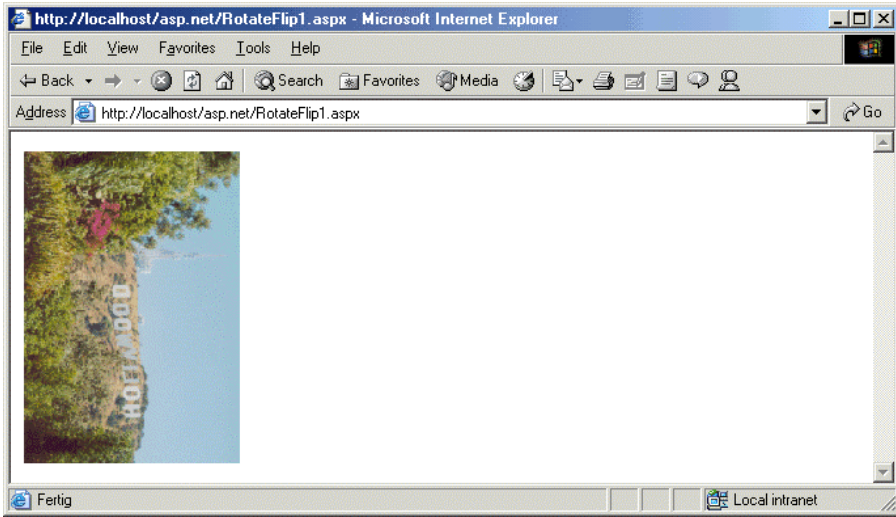


Abbildung 12.5 Das Bild wurde um 270° gedreht und um die x-Achse gespiegelt.

12.8 ... dynamisch Grafiken erstellen?

Statt vorhandene Grafik auszugeben, können Sie Bilder auch vollkommen dynamisch erstellen und direkt aus dem Arbeitsspeicher an den Client senden. Hier instanziiieren Sie die Klasse `Bitmap` nicht mit einem Pfad, sondern mit zwei int-Werten, die die Breite und Höhe des Bildes in Pixel angeben. Das Bild übergeben Sie der statischen Methode `Graphics.FromImage`. Sie erhalten nun eine Instanz der Klasse `Graphics`, die Ihnen zahlreiche Zeichenmöglichkeiten offeriert.

Das folgende Beispiel zeigt den einfachsten Einsatz der beiden Klassen. Es wird ein neues Bild mit den Maßen 200 x 100 Pixel angelegt. Die standardmäßig schwarze Hintergrundfarbe wird mittels `Graphics.Clear` auf geändert gesetzt. Anschließend wird das so erzeugte Bild an den Client übertragen.

Listing 12.9 CreateBitmap1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Bitmap b = new Bitmap(200, 100);
    Graphics g = Graphics.FromImage(b);

    g.Clear(Color.Red);

    g.Flush();
}
```

```
Response.ContentType = "image/gif";  
b.Save(Response.OutputStream, ImageFormat.Gif);  
}  
  
</script>
```

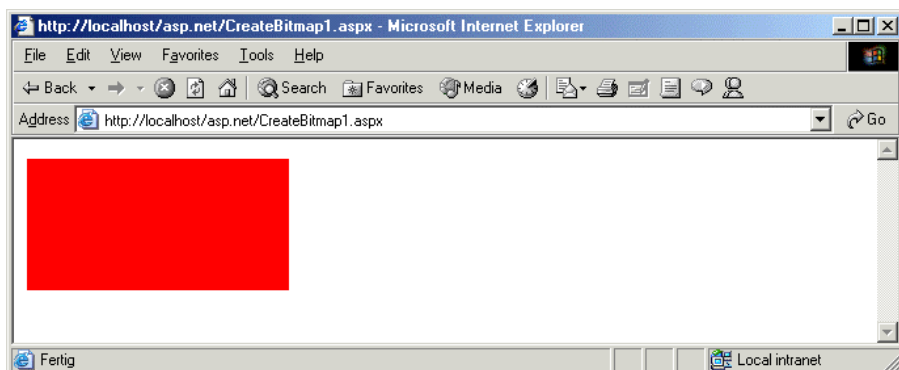


Abbildung 12.6 Ein rotes Rechteck – welche wahnsinnige Leistung

Natürlich können Sie mehr, als eine rote Fläche im Browser darstellen. Die Möglichkeiten sind wirklich immens. Sie können beispielsweise Linien, Kurven, Rechtecke, Kreise, Polygone und vieles mehr ausgeben. Nachfolgend ein kleines Beispiel mit einigen weiteren Grafikelementen, einem Kreis, zwei ausgefüllten Kreisen, einer Linie und einer Kurve.

Listing 12.10 CreateBitmap2.aspx

```
<script runat="server">  
  
void Page_Load(object sender, EventArgs e)  
{  
    Bitmap b = new Bitmap(200, 200);  
    Graphics g = Graphics.FromImage(b);  
  
    g.Clear(Color.White);  
  
    Pen pen = new Pen(Color.Red);  
    pen.Width = 3;  
  
    g.DrawEllipse(pen, 2, 2, 196, 196);  
  
    Point[] points = {new Point(40, 140), new Point(100, 170),  
                      new Point(160, 140)};  
    g.DrawCurve(pen, points);  
}
```



```
g.DrawLine(pen, 100, 70, 100, 130);

Brush brush = new SolidBrush(Color.Black);
g.FillEllipse(brush, 50, 60, 20, 20);
g.FillEllipse(brush, 130, 60, 20, 20);

g.Flush();
Response.ContentType = "image/gif";
b.Save(Response.OutputStream, ImageFormat.Gif);
}

</script>
```

Auch wenn das Beispiel ein wenig komplexer und auf jeden Fall fröhlicher ist als das vorherige, so wird es dennoch den Möglichkeiten der Klasse `Graphics` nicht gerecht. Sollten Sie sich für weitergehende Informationen interessieren, empfehle ich als Lektüre die .NET Framework SDK-Dokumentation. Hier werden alle Methoden der Klasse ausführlich und anhand diverser Beispiele aufgezeigt.

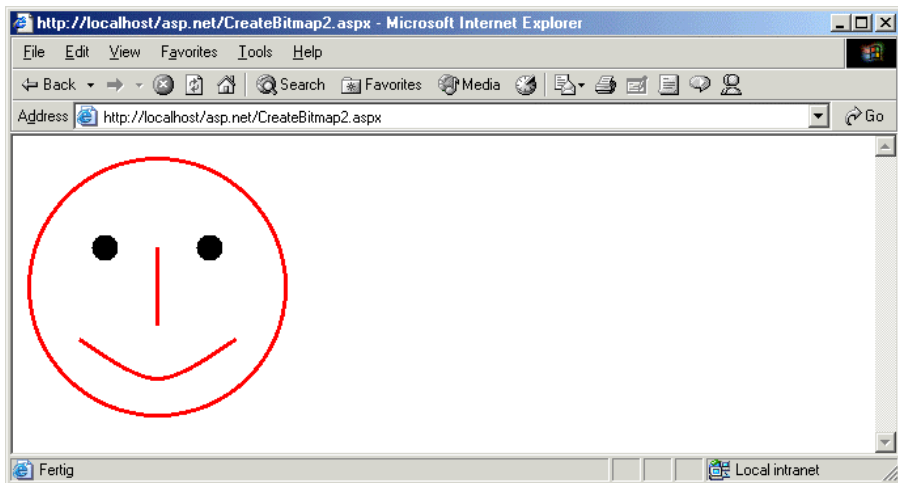


Abbildung 12.7 Punkt, Punkt, Komma, Strich, fertig ist das Mondgesicht!

12.9 ... grafischen Text ausgeben?

Die Ausgabe von grafischem Text scheint zunächst einmal nicht unbedingt wichtig zu sein, schließlich kann man den Text auch direkt in einer ASP.NET-Seite als solchen ausgeben. Ausnahmsweise ist es der Jurist, der sich die Darstellung als Bild wünscht. Warum?

Mit Hilfe von Scripts kann man nahezu jede Seite fernsteuern und automatisieren. Möchte man dies verhindern und sicherstellen, dass auch wirklich ein (menschliches) Wesen interagiert, kann man beispielsweise zur Eingabe eines Wortes auffordern, das als Grafik angezeigt wird. Das Script hat keine oder zumindest keine ernst zu nehmende Chance, aus dem Bild den einzugebenden Text zu gewinnen.

Das Listing zeigt eine derartige Abfrage. Die Formularseite enthält ein Eingabefeld für die Zeichenkette, einen Button sowie ein img-Tag. Beim ersten Laden wird aus einem Array ein zufälliges Wort gewählt und in einer Session-Variablen abgelegt. Die Zeichenkette wird von der zweiten Seite als Grafik ausgegeben, wenn diese über das img-Tag referenziert wird. Nur nach Eingabe des korrekten Wortes geht es weiter.

Listing 12.11 CheckString1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        string[] texts = {"Hallo", "Welt", "Filou", "Shari"};

        Random rnd = new Random();
        int index = rnd.Next(texts.Length);

        string text = texts[index];
        Session["text"] = text;

        ph2.Visible = false;
    }
}

void bt_click(object sender, EventArgs e)
{
    bool IsOK = (tb.Text == (string) Session["text"]);
    ph1.Visible = !IsOK;
    ph2.Visible = IsOK;

    if(!IsOK)
        lb.Text = "Das war leider nicht korrekt ...";
}

</script>

<form runat="server">

<h1> Bist Du menschlich? </h1>
```

```
<ASP:Placeholder id="ph1" runat="server">

<p> Bitte geben Sie den in der Grafik gezeigten Text ein: </p>

<p><b> <ASP:Label id="lb" runat="server"/> </b></p>

<ASP:TextBox id="tb" runat="server"/>
<ASP:Button runat="server" Text="OK" OnClick="bt_click"/>

<p>  </p>

</ASP:Placeholder>

<ASP:Placeholder id="ph2" runat="server">
<p> ... scheinbar ja! </p>
</ASP:Placeholder>

</form>
```

Die Ausgabe des Textes in der Session-Variablen erfolgt über die Methode `Graphics.DrawString`, der neben dem Text auch die zu verwendende Schriftart sowie ein Zeichenstift übergeben wird.

Listing 12.12 DrawString1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Bitmap b = new Bitmap(200, 50);
    Graphics g = Graphics.FromImage(b);

    g.Clear(Color.White);

    string text = (string) Session["text"];
    g.DrawString(text, new Font("Comic Sans MS", 25),
        new SolidBrush(Color.Red), 0, 0);

    g.Flush();
    Response.ContentType = "image/gif";
    b.Save(Response.OutputStream, ImageFormat.Gif);
}

</script>
```



Abbildung 12.8 Nicht so einfach zu überlisten ...

12.10 ... grafischen Text formatieren?

Statt den Text „nur“ auszugeben, können Sie auch zahlreiche Formatierungsmöglichkeiten einsetzen. Dies fängt bei der Wahl der Schriftattribute wie Fett und Unterstrichen an. Beides wird im nachfolgenden Listing gezeigt.

Listing 12.13 DrawString2.aspx

```
void Page_Load(object sender, EventArgs e)
{
    Bitmap b = new Bitmap(500, 200);
    Graphics g = Graphics.FromImage(b);
    g.Clear(Color.White);

    string text = ":-))))";
    Brush brush = new TextureBrush(
        new Bitmap(Server.MapPath("paramount.gif")));
    Font font = new Font("Comic Sans MS", 90, FontStyle.Bold |
        FontStyle.Italic);

    g.DrawString(text, font, brush, 0, 0);

    g.Flush();
    Response.ContentType = "image/gif";
    b.Save(Response.OutputStream, ImageFormat.Gif);
}
```

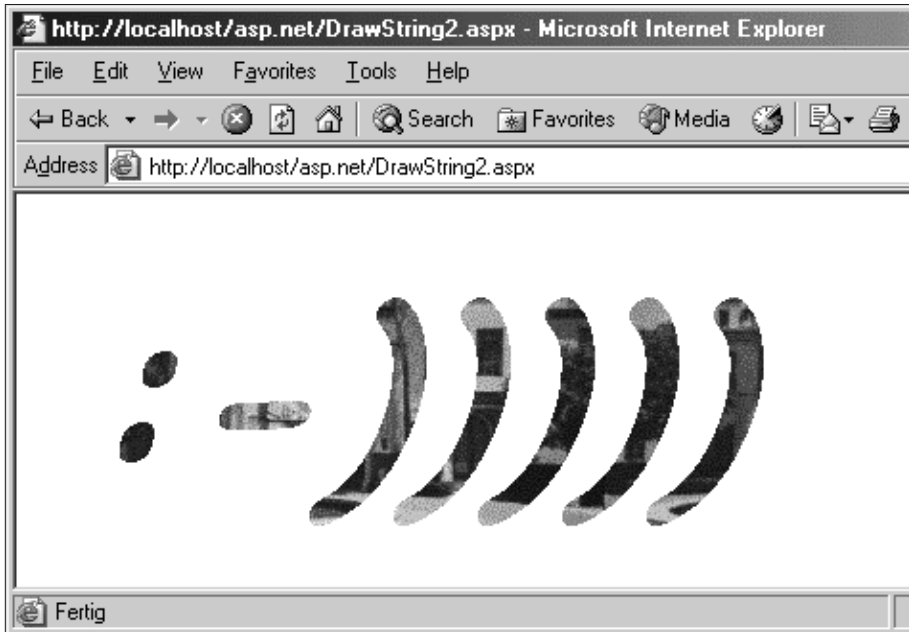


Abbildung 12.9 Die Schrift wird als Textur ausgegeben.

Wichtig ist auch die Wahl des Zeichenpinsels. Während im vorherigen Rezept eine öde Instanz der `SolidBrush`-Klasse zum Einsatz kam, zeigt die Abbildung nun die Ausgabe einer Grafik innerhalb der Schrift. Dies wurde über eine `TextureBrush` realisiert, der im Konstruktor das zu verwendende Bild übergeben wurde. Weitere Zeichenpinsel stehen beispielsweise zum Zeichnen von Farbverläufen bereit.

Listing 12.14 `DrawString3.aspx`

```
...  
  
string text = ":-)"))))";  
Brush brush = new LinearGradientBrush(  
    new Point(0, 0), new Point(500, 200), Color.Yellow, Color.Red);  
Font font = new Font("Comic Sans MS", 90, FontStyle.Bold |  
    FontStyle.Italic);  
  
g.DrawString(text, font, brush, 0, 0);  
  
...
```

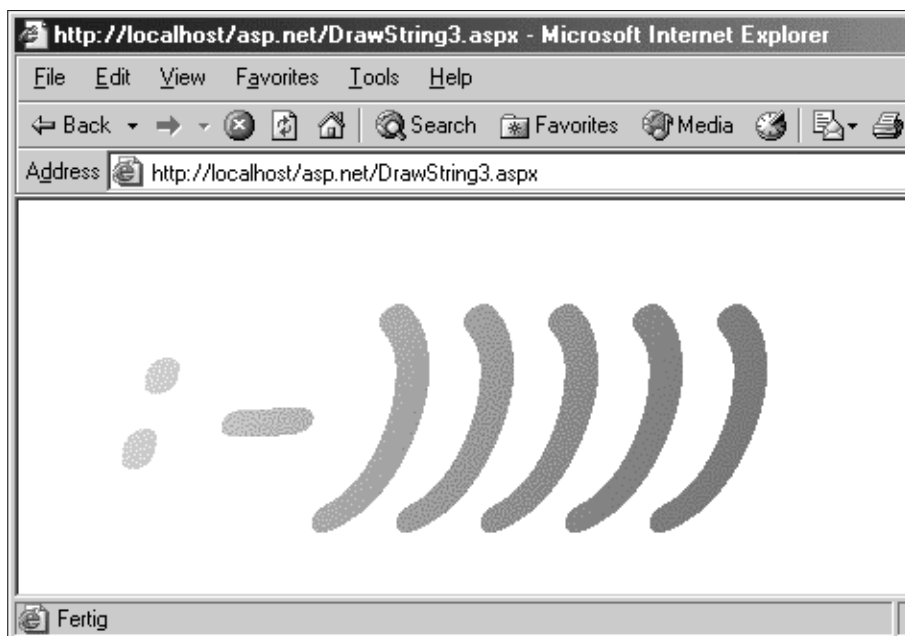


Abbildung 12.10 Der Text enthält einen Farbverlauf.

Auch Rotationen sind möglich. Hierzu müssen Sie vor dem Zeichnen der Methode `RotateTransform` den gewünschten Gradwinkel übergeben. Alle anschließend gezeichneten Objekte und somit auch der Text werden nun im angegebenen Winkel rotiert.

Listing 12.15 DrawString4.aspx

```
...  
  
string text = "Hallo Welt";  
g.RotateTransform(45f);  
g.DrawString(text, new Font("Arial", 30),  
    new SolidBrush(Color.Black), 30, 0);  
  
...
```

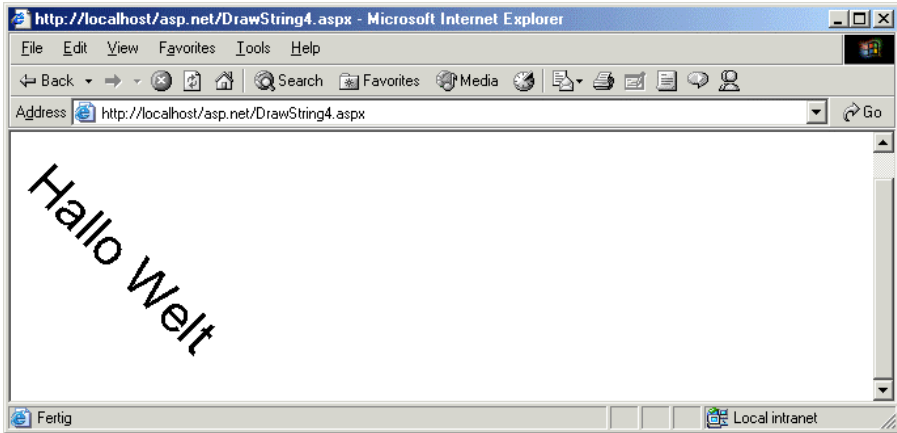


Abbildung 12.11 Der Text wurde um 45° rotiert.

Beim Aufruf der Methode `RotateTransform` wird nicht das Bild selbst rotiert, sondern dessen (Koordinaten-)Matrix. Das bisherige Erscheinungsbild wird daher nicht manipuliert, sondern es werden lediglich nachfolgende Ausgaben auf Basis der geänderten Matrix ausgegeben. Die Möglichkeiten sind sehr weitreichend. Sie können beispielsweise den Status des Objekts vor der Matrixänderung speichern und nach der Ausgabe des Textes wieder zurücksetzen. Innerhalb einer Schleife lässt sich so etwa Text in einem Kreis ausgeben.

Listing 12.16 DrawString5.aspx

```
...  
  
string text = "Hallo Welt";  
  
for(int i=0; i<360; i+=10)  
{  
    GraphicsState state = g.Save();  
    g.RotateTransform((float) i);  
    g.DrawString(text, new Font("Arial", 10),  
        new SolidBrush(Color.Black), 100, 0);  
    g.Restore(state);  
}  
  
...
```

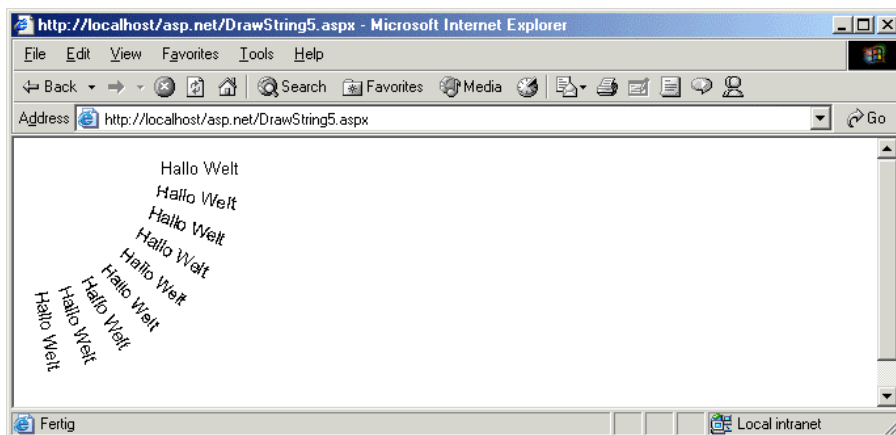


Abbildung 12.12 Der Text wird im Uhrzeigersinn rotiert.

12.11 ... ein zufälliges Bild anzeigen?

Bei Produktpräsentationen werden oft Bilder eines Produktes angezeigt. Stehen davon einige zur Verfügung, können sie bei jedem Aufruf einer Seite zufällig angezeigt werden. Dies zu realisieren ist mit Hilfe der Klasse `Random` und der statischen Methode `Directory.GetFiles` spielend einfach. Das folgende Listing fragt alle GIF-Bilder im aktuellen Verzeichnis ab und gibt ein zufällig ausgewähltes im Browser aus.

Listing 12.17 `RandomPic1.aspx`

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string dir = Server.MapPath("");
    string[] files = Directory.GetFiles(dir, "*.gif");
    Random rnd = new Random((int)DateTime.Now.Ticks);
    int picindex = rnd.Next(files.Length);
    Response.ContentType = "image/gif";
    Response.WriteFile(files[picindex]);
}

</script>
```

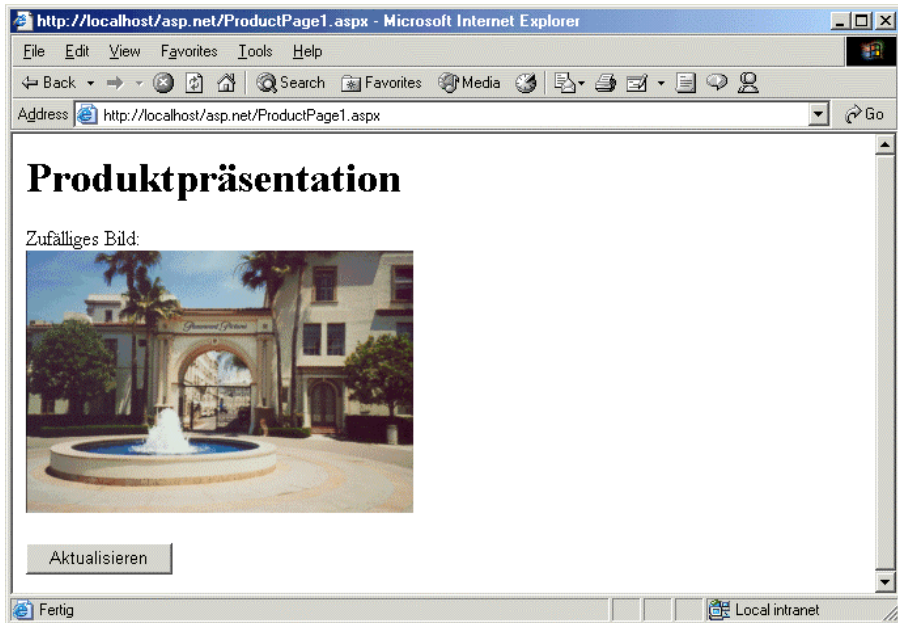



Abbildung 12.13 Das Bild wurde zufällig ausgewählt.

Um das Bild innerhalb einer HTML-Seite anzuzeigen, kann die Seite als Quelle für ein `img`-Tag eingebunden werden. Die Abbildung zeigt dies anhand einer kleinen Beispielseite.

Statt das vollständige Bild anzuzeigen, bietet es sich unter Umständen an, nur eine verkleinerte Version auszugeben. Möchte der Benutzer die Grafik in Originalgröße sehen, braucht er diese nur anzuklicken. Und genau dies ist das Problem. Um einen Link auf die Originalgrafik in die Produktseite aufnehmen zu können, muss die Logik zur zufälligen Ermittlung eines Bildes in diese Seite verlagert werden. Das Listing zeigt dies. Die Adresse des Bildes wird nun einem Image- und einem HyperLink-Control zugewiesen.

Listing 12.18 ProductPage2.aspx

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string dir = Server.MapPath("");
    string[] files = Directory.GetFiles(dir, "*.gif");
    Random rnd = new Random((int)DateTime.Now.Ticks);
    int picindex = rnd.Next(files.Length);
```

```
string filename = Path.GetFileName(files[picindex]);
image.ImageUrl = "ShowThumb2.aspx?file=" + filename;
link.NavigateUrl = filename;
}

</script>

<h1>Produktpräsentation</h1>

<form runat="server">

<p> Zufälliges Bild - Klick für Originalgröße: <br/>
  <ASP:HyperLink id="link" runat="server">
    <ASP:Image id="image" runat="server"/>
  </ASP:HyperLink>
</p>

<ASP:Button runat="server" text="Aktualisieren"/>

</form>
```

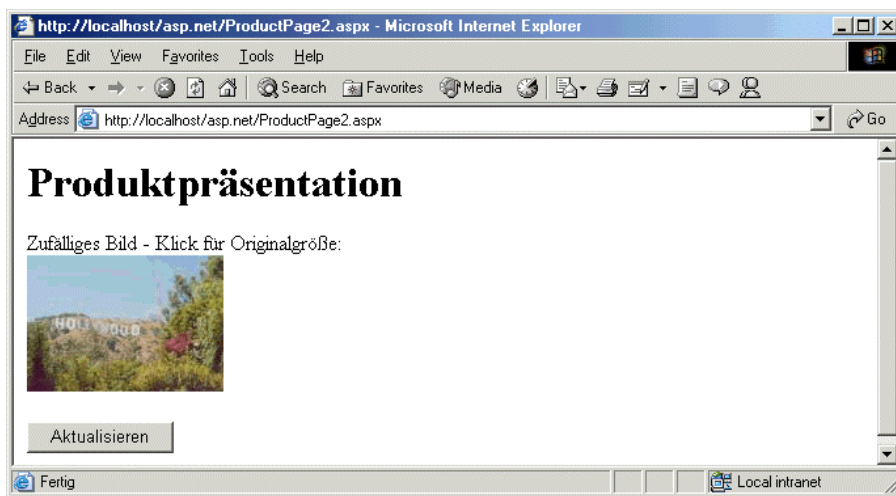


Abbildung 12.14 Das zufällige Bild wird verkleinert angezeigt, ein Klick öffnet das Original.

Die Anzeige des Bildes erfolgt über ein Image-Control. Als Quelle wird die Seite `ShowThumb2.aspx` angegeben, der über den Query-String die verkleinert anzuzeigende Grafik übergeben wird. Die Erzeugung dieses Thumbnails erfolgt wie im Rezept „... automatisch Thumbnails von Bildern erzeugen?“ beschrieben.

Listing 12.19 ShowThumb2.aspx

```
<% @Import Namespace="System.Drawing" %>
<% @Import Namespace="System.Drawing.Imaging" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string file = Server.MapPath(Request["file"]);
    Bitmap b = new Bitmap(file);
    Bitmap thumb = GetThumbnail(b, 150, 0);
    Response.ContentType = "image/jpeg";
    thumb.Save(Response.OutputStream, ImageFormat.Jpeg);
}

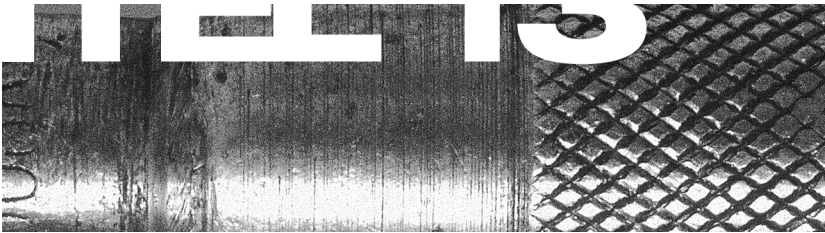
Bitmap GetThumbnail(Bitmap b, int w, int h)
{
    if(w == 0 && h == 0)
    {
        w = b.Size.Width;
        h = b.Size.Height;
    }
    else if(w == 0)
        w = (int) (((float) h) / b.Size.Height * b.Size.Width);
    else if(h == 0)
        h = (int) (((float) w) / b.Size.Width * b.Size.Height);

    return((Bitmap) b.GetThumbnailImage(w, h, null, IntPtr.Zero));
}

</script>
```

13

System und Netzwerk



Wie kann ich ...

13 System und Netzwerk

Was bei früheren ASP-Versionen tabu und schlichtweg nicht möglich war, wird jetzt zum Kinderspiel. Die Rede ist von einem mitunter tiefen Eingriff in das System. Dieses Kapitel stellt verschiedene Möglichkeiten vor, die Ihnen aus Ihrer Web-Applikation heraus Zugriff auf verschiedene Aspekte des Systems geben.

Das .NET Framework enthält nun auch zahlreiche Möglichkeiten, Netzwerkfunktionen aufzurufen. Sie können beispielsweise DNS-Abfragen durchführen oder gar auf Protokollebene Kontakt mit anderen Servern aufnehmen. Die notwendigen Klassen sind vornehmlich im Namespace `System.Net` abgelegt. Dieser sollte daher explizit eingebunden werden.

13.1 ... eine COM-Komponente in ASP.NET verwenden?

Prinzipiell erlaubt die Common Language Runtime ausschließlich Zugriff auf so genannten managed Code. Es können also nur solche Programme und Komponenten verwendet werden, die von der CLR selbst ausgeführt werden können. Diese Einschränkung gilt auch für ASP.NET, und Sie sollten bisherige COM-(Automation-)Komponenten nicht mehr direkt beispielsweise per `CreateObject` verwenden.

Selbstverständlich wurde eine Möglichkeit entwickelt, um bisherige Komponenten weiter verwenden zu können. COM Interop ist das Stichwort. Hierbei wird um eine bestehende Komponente ein „Mantel“ aus managed Code gelegt. Diesen Wrapper können Sie wie eine reguläre .NET Assembly verwenden. Intern werden jedoch alle Aufrufe an die entsprechende COM-Komponente weitergeleitet.

Erstellung einer Wrapper-Assembly

Zur Erstellung der Wrapper-Assembly stellt das Framework das Kommandozeilenprogramm `tlbimp.exe` zur Verfügung. Diesem wird als Parameter eine DLL übergeben:

```
tlbimp.exe <DLLName>
```

Das Programm fragt die Type Library der DLL ab und erzeugt daraus das entsprechende Klassengerüst als managed Code. Um beispielsweise einen Wrapper für die ADO-DLLs zu erstellen, verwenden Sie folgenden Aufruf:

```
Tlbimp.exe "C:\Programme\Gemeinsame Dateien\System\ADO\msado15.dll"
```

Dies erzeugt eine Assembly `adodb.dll`, die Sie über die `@Assembly`-Direktive in Ihre ASP.NET-Seite importieren können. Alternativ können Sie die Datei auch im `bin`-Verzeichnis Ihrer Web-Applikation ablegen. Der Dateiname entspricht übrigens nicht dem Namen der ursprünglichen Komponente, sondern deren `ProgID`, in diesem Fall also `ADODB`. Dieser Name wird auch dem Namespace gegeben, und so können Sie die `@Import`-Direktive verwenden, um diesen zu importieren. Der Aufruf der Komponente erfolgt anschließend wie bei .NET gewohnt. Das Listing zeigt's.

Listing 13.1 cominterop1.aspx

```
<% @Import Namespace="ADODB" %>

<script runat=server>

void Page_Load(object sender, EventArgs e)
{
    Recordset rs = new Recordset();
    // RS...
}

</script>
```



Beachten Sie bitte, dass die erstellte Wrapper-DLL nur den beschriebenen Mantel, aber nicht die eigentliche Funktionalität enthält. Möchten Sie diese beispielsweise auf einem anderen System nutzen, muss dort die ursprüngliche Komponente zwingend vorhanden und korrekt installiert/registriert sein.

Der einfache Weg mit Visual Studio .NET

Die Entwicklungsumgebung von Microsoft ist zwar nicht ganz kostenlos, bietet dafür aber jede Menge Komfort. Das gilt auch für die Einbindung von COM-Komponenten. Statt per Kommandozeilenprogramm manuell einen Wrapper zu erstellen, können Sie dies getrost der IDE überlassen.

- Um die bestehende Komponente einzubinden, klicken Sie im Solution Explorer rechts auf den Eintrag „Referenzen“ und anschließend „Referenz hinzufügen...“.
- Im nun erscheinenden Dialog wählen Sie die Lasche „COM“. Es werden nun alle im System registrierten COM-Komponenten angezeigt.
- Um wie im obigen Beispiel das alte ADO einzubinden, wählen Sie den Eintrag „Microsoft ActiveX Data Objects x.x Library“ aus, wählen „Select“ und bestätigen den Dialog mit „OK“.
- Ohne weiteres Zutun steht nun die Komponente zur Verfügung. Die Entwicklungsumgebung hat automatisch im Hintergrund die benötigte Wrapper-DLL erstellt und eingebunden. Einfacher geht's nimmer.

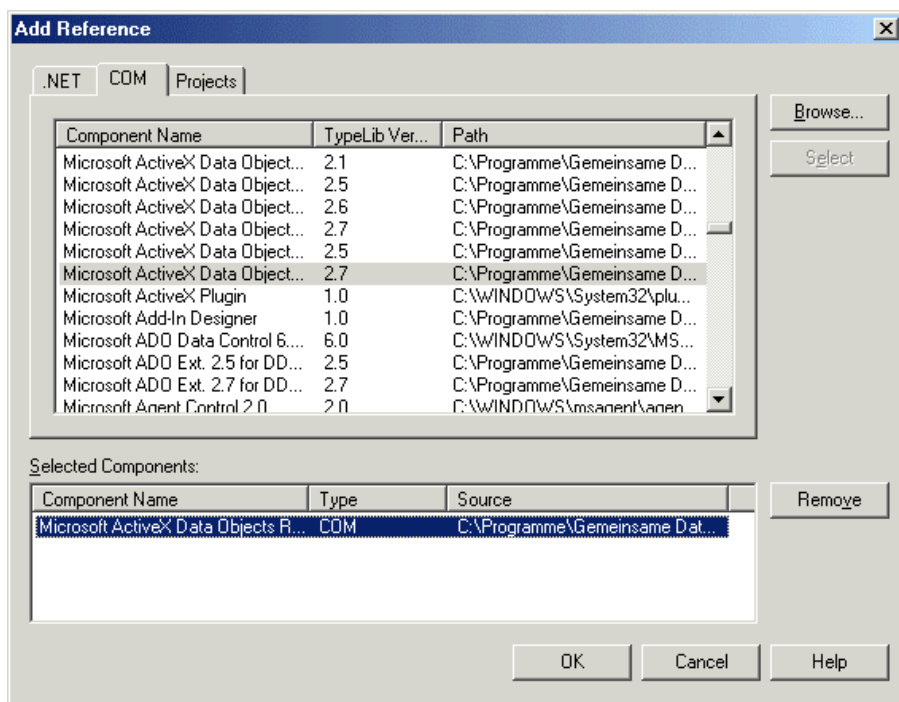


Abbildung 13.1 COM-Komponenten lassen sich ganz einfach einfügen.

13.2 ... eine Win32 API-Funktion aufrufen?

ASP.NET integriert sich vollständig in das .NET Framework und stellt dessen vollen Funktionsumfang zur Verfügung. Hierzu zählt auch, dass Sie aus einer Web-Applikation heraus Funktionen der Win32 API aufrufen können. Dies kann mitunter sehr nützlich sein.

In C# wird eine derartige Funktion als statische Methode implementiert. Diese muss zusätzlich mit dem Modifikator `extern` versehen werden. Der Dateiname der implementierenden DLL wird mit Hilfe des `DllImport`-Attributs angegeben. Ähnlich wie bei einer abstrakten Methode wird auch hier nur der Rumpf der Methode implementiert.

Das nachfolgende Beispiel zeigt den Aufruf einer Win32 API-Funktion. Es handelt sich um `LogonUser`, mit Hilfe dessen eine Kombination von Benutzername und Passwort gegen die Windows-interne Benutzerdatenbank geprüft werden kann.

Listing 13.2 win32api1.aspx

```
<% @Import Namespace="System.Runtime.InteropServices" %>

<script language="C#" runat=server>

[DllImport("advapi32.dll")]
public static extern bool LogonUser(string lpszUsername, string
    lpszDomain, string lpszPassword, int dwLogonType,
    int dwLogonProvider, out int phToken);

[DllImport("Kernel32.dll")]
public static extern int GetLastError();

void Submit_Click(object sender, EventArgs e)
{
    int usertoken;
    bool LoginOK = LogonUser(tb_username.Text, "", tb_password.Text,
        3, 0, out usertoken);

    if(LoginOK)
    {
        lb_info.Text = "Die Daten sind korrekt.";
    }
    else
    {
        int LastError = GetLastError();
        lb_info.Text = "Die Anmeldung konnte nicht bestätigt werden. Bitte
            versuchen Sie es noch einmal. (" + LastError + ")";
    }
}
```



```
</script>

<form runat="server">

<h1>Login</h1>

<p> Bitte geben Sie Ihre Benutzerdaten an:</p>

<p>
  Benutzername:
  <ASP:TextBox id="tb_username" runat="server"/>
</p>

<p>
  Benutzername:
  <ASP:TextBox id="tb_password" runat="server"
    TextMode="Password"/>
</p>

<p> <ASP:Label id="lb_info" runat="server"/> </p>

<ASP:Button id="Submit" runat="server"
  Text="Login"
  OnClick="Submit_Click"/>

</form>
```

Nach der Bestätigung der Eingaben werden die Benutzerdaten mit Hilfe der Win32 API-Funktion überprüft. Die boolesche Rückmeldung wird im Browser ausgegeben. Im Fehlerfall wird zusätzlich der zurückgelieferte Fehlercode angegeben. Sie können dieses System beispielsweise zur Verknüpfung von Forms und Windows Authentication verwenden, wenn Sie einen Bereich Ihrer Applikation schützen möchten.

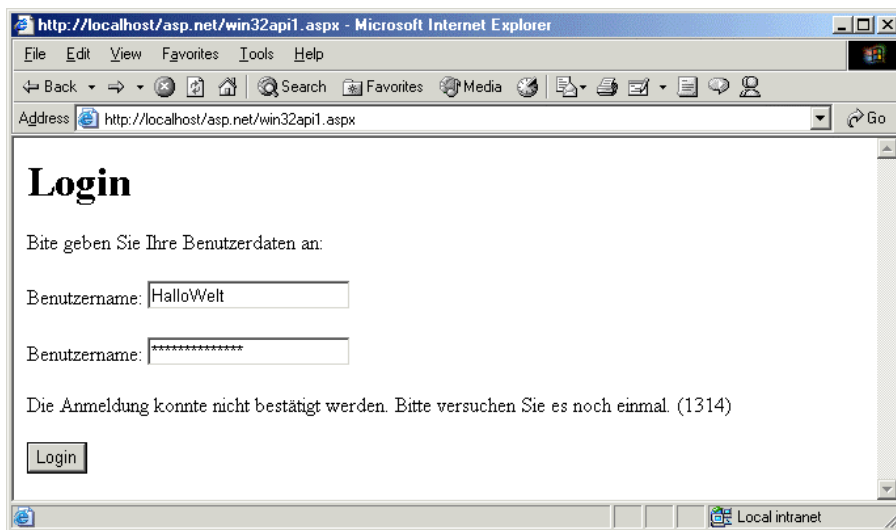


Abbildung 13.2 Die Benutzerdaten waren leider nicht korrekt.

Die Funktionen der Win32 API sind unmanaged Code, da sie nicht von der Common Language Runtime durchgeführt und somit überwacht werden können. Oftmals führen sie in einem derartigen Kontext Zeigeroperationen aus. Auch dies ist in ASP.NET problemlos möglich. Wie bei Windows-Applikationen auch müssen Sie den entsprechenden mit dem Schlüsselwort `unsafe` versehen. Dieses Schlüsselwort wird jedoch nur akzeptiert, wenn Sie den Compiler mit dem Kommandozeilenparameter `/unsafe` aufrufen. Da Sie den Compiler bei ASP.NET nicht direkt aufrufen, können Sie das Attribut `CompilerOptions` der `@Page`-Direktive hierzu verwenden.

Listing 13.3 win32api2.aspx

```
<% @Page Language="C#" Debug="true" CompilerOptions="/unsafe" %>
<% @Import Namespace="System.Runtime.InteropServices" %>

<script language="C#" runat=server>

[DllImport("advapi32.dll")]
public static extern bool LogonUser(string lpszUsername, string
    lpszDomain, string lpszPassword, int dwLogonType,
    int dwLogonProvider, out int phToken);

[DllImport("Kernel32.dll")]
public static extern int GetLastError();
```



Beachten Sie bitte, dass nicht auf jedem System die zur Ausführung von unmanaged Code notwendigen Rechte verfügbar sind. Insbesondere Webspaces-Provider schränken die Möglichkeiten aus gutem Grund ein. Zu groß ist die Gefahr, dass eine derartige Lücke missbraucht wird. Sollten Sie Schwierigkeiten haben, sprechen Sie doch gegebenenfalls einmal Ihren Provider an.

13.3 ... ein Programm auf dem Server starten?

Mitunter möchten Sie aus einem Server ein Programm starten. Bei älteren ASP-Versionen war dies nur unter Zuhilfenahme einer externen Komponente möglich. Bei ASP.NET steht Ihnen die Klasse `Process` aus dem Namespace `System.Diagnostics` zur Verfügung. Über die statische Methode `Start` können Sie ein Programm starten. Die zurückgelieferte Instanz der Klasse `Process` gibt Ihnen vielfältige Informationen über den neuen Prozess an die Hand und erlaubt unter anderem auch das vorzeitige Beenden des Programms. Vor der Verwendung muss der entsprechende Namespace eingebunden werden.

Das Listing zeigt den Start des Notepads. Es wird gewartet, bis das Programm vollständig gestartet ist. Anschließend wird der Name des Hauptmoduls („notepad.exe“) im Browserfenster ausgegeben und das Programm (unsachgemäß) explizit beendet.

Listing 13.4 Process1.aspx

```
<% @Import Namespace="System.Diagnostics" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Process proc = Process.Start("notepad.exe");
    proc.WaitForInputIdle();
    Response.Write(proc.MainModule.ModuleName);
    proc.Kill();
}

</script>
```

Ein Programm aus einer Web-Applikation heraus zu starten ist zwar prinzipiell aus technischer Sicht ganz nett, einen echten Praxiswert besitzt das gezeigte Beispiel jedoch nicht.

Interessant wird das Starten eines Prozesses, wenn anschließend mit diesem eine Kommunikation aufgebaut werden kann. Bei Windows-Programmen erfolgt eine derartige Kommunikation über die Standardeingabe- beziehungsweise -ausgabe-

Ports. Auch diese lassen sich mit Hilfe der `Process`-Klasse ansprechen. Hierzu werden wie bei .NET üblich Streams verwendet. Auf diese Weise können Sie Daten mit dem gestarteten Programm austauschen. Die Kommunikation kann, muss aber nicht interaktiv sein.

Das Listing zeigt eine einseitige Kommunikation mit dem bekannten Programm `ping.exe`. Im Eingabeformular kann der Benutzer eine IP-Adresse oder einen Hostnamen angeben. Ein Klick startet das Programm `ping.exe` und übergibt die Eingabe als Kommandozeilenparameter. Anschließend wird auf das Ende des Programms gewartet, und die erhaltenen Daten werden mit Hilfe eines `StreamReader` im Browserfenster ausgegeben.

Listing 13.5 Process2.aspx

```
<script runat="server">

void ping_click(object sender, EventArgs e)
{
    Process proc = new Process();
    proc.StartInfo.FileName = "ping";
    proc.StartInfo.Arguments = "-a " + tb_ip.Text;
    proc.StartInfo.UseShellExecute = false;
    proc.StartInfo.RedirectStandardOutput = true;
    proc.Start();
    proc.WaitForExit();
    string result = proc.StandardOutput.ReadToEnd();
    lb_result.Text = result.Replace("\r\n", "<br>");
}

</script>

<h1>Ping</h1>

<form runat="server">

    <p> IP oder Host:
        <ASP:TextBox id="tb_ip" runat="server" text="127.0.0.1"/>
    </p>

    <p> <ASP:Button runat="server" onclick="ping_click" text="Ping"/>
    </p>

    <h2>Ergebnis</h2>

    <pre> <ASP:Label id="lb_result" runat="server"/> </pre>

</form>
```

Die Abbildung zeigt das Ergebnis eines Pings auf den lokalen Rechner. Damit Sie die Ausgaben des Programms erhalten, müssen Sie den Start mit Hilfe der StartInfo-Eigenschaft genauer spezifizieren. So müssen die Eigenschaften `UseShellExecute` und `RedirectStandardOutput` explizit auf `false` beziehungsweise `true` gesetzt werden.

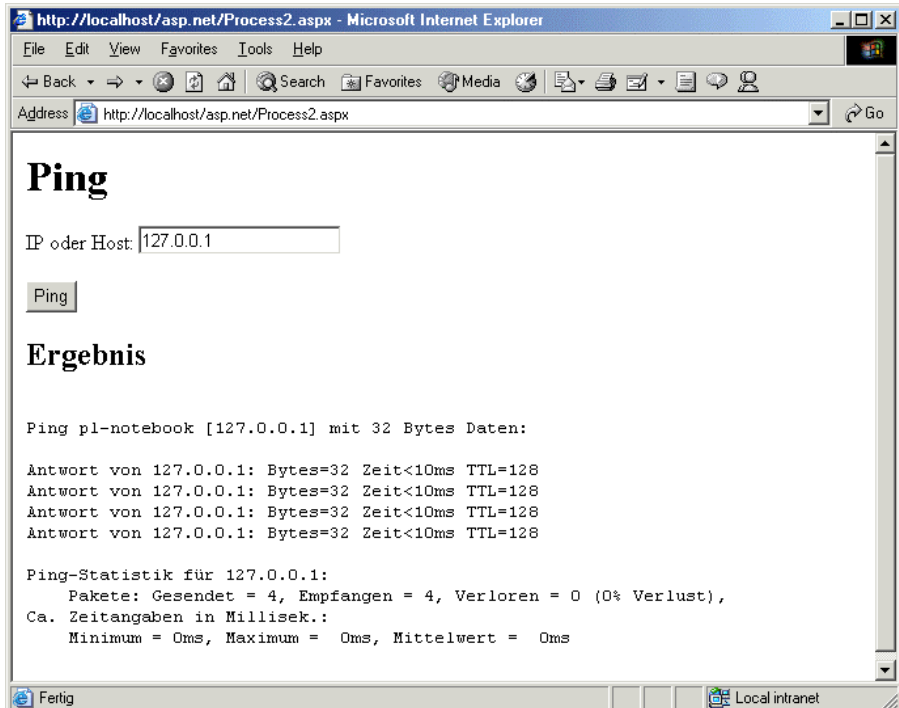


Abbildung 13.3 Der Ping auf den lokalen Rechner ist erwartungsgemäß schnell.

13.4 ... eine Übersicht der Server-Prozesse anzeigen?

Mit der Klasse `Process` lässt sich mehr anfangen, als ein Programm auf dem Server zu starten. Die statische Methode `GetProcesses` liefert beispielsweise ein `Process`-Array mit allen gestarteten Prozessen. Sie können das Array als Datenquelle für ein `DataGrid`-Control verwenden. Auf diese Weise lässt sich bequem eine Liste der geöffneten Prozesse anzeigen.

Das Listing zeigt einen derartigen Ansatz. Um zusätzliche Informationen wie Prozessname, Startzeit, aktuelle CPU-Zeit und so weiter anzeigen zu können, wird das ItemDataBound-Ereignis verwendet. Hier können die Spalten des Controls individuell für jede Zeile „nachbearbeitet“ werden.

Listing 13.6 ProcessList1.aspx

```
...

void InitDataSource()
{
    Process[] Processes = Process.GetProcesses();
    Array.Sort(Processes, new ProcessComparer());
    ProcessList.DataSource = Processes;
    Page.DataBind();
}

void ProcessList_Bound(Object sender, DataGridItemEventArgs e)
{
    Process P = (Process) e.Item.DataItem;
    if(P != null)
    {
        e.Item.Cells[1].Text = GetLocalProcessName(P.ProcessName);

        TimeSpan TPT = P.TotalProcessorTime;
        e.Item.Cells[3].Text = TPT.Hours + ":" + TPT.Minutes + ":" +
            TPT.Seconds;

        e.Item.Cells[5].Text = P.Threads.Count.ToString();

        if(P.Responding)
        {
            e.Item.BackColor = Color.White;
            e.Item.Cells[6].Text = "OK";
        }
        else
        {
            e.Item.BackColor = Color.Red;
            e.Item.Cells[5].Text = "Nicht OK";
        }
    }
}

...
```

Das DataGrid-Control verfügt über einen zusätzlichen Button zum Beenden des jeweiligen Prozesses. Zur Identifizierung wird die eindeutige Prozess-ID benutzt. Mit Hilfe der statischen `Process.GetProcessById` wird der Prozess abgefragt. Es

wird nun versucht, das Hauptfenster des Prozesses explizit zu schließen. Gelingt dies nicht, kann der Prozess mittels der Methode `Kill` auf die harte Tour beendet werden.

```
void ProcessList_KillProcess(Object sender,
    DataGridCommandEventArgs e)
{
    Process P = Process.GetProcessById(int.Parse(e.Item.Cells[0].Text));
    string ProcessName = e.Item.Cells[1].Text;
    if (P.CloseMainWindow())
    {
        StatusLabel.Text = "\"" + ProcessName + "\" wurde geschlossen!";
    }
    else
    {
        try
        {
            P.Kill();
            StatusLabel.Text = "\"" + ProcessName + "\" musste
                gewaltsam beendet werden!";
        }
        catch (Exception ex)
        {
            StatusLabel.Text = "Gewaltsames Beenden von \"" + ProcessName + "\"
                nicht möglich, " + ex.Message;
        }
    }
    InitDataSource();
}
```

Die Abbildung zeigt die Prozessliste im Einsatz. Über den Button „Aktualisieren“ kann die Liste jederzeit wieder auf den aktuellen Stand gebracht werden. Sie sehen auch, wie viele Handles jedes Programm verbraucht und wie viele Threads es verwendet.

Prozessname	Startzeit	CPU Zeit	Handles	Threads	Status	Beenden
aspnet_wp	14.05.2002 19:54:15	0:0:4	405	17	OK	Beenden
csrss	14.05.2002 18:48:58	0:0:6	370	11	OK	Beenden
dexplore	14.05.2002 18:50:15	0:0:10	245	5	OK	Beenden
dllhost	14.05.2002 19:35:53	0:0:0	127	8	OK	Beenden
Explorer	14.05.2002 18:49:23	0:0:21	345	15	OK	Beenden
hidserv	14.05.2002 18:49:07	0:0:0	67	4	OK	Beenden
iexplore	14.05.2002 18:50:14	0:0:11	239	10	OK	Beenden
inetinfo	14.05.2002 18:49:14	0:0:1	397	23	OK	Beenden

Abbildung 13.4 Das DataGrid-Control zeigt alle Prozesse auf dem System an.



Der von der ASP.NET-Engine verwendete Benutzer-Account „ASPNET“ verfügt nur über sehr eingeschränkte Rechte. Je nach Konfiguration kann daher die Abfrage der geöffneten Prozesse mit einer Win32Exception abgebrochen werden. In diesem Fall sollten Sie die Rechte des Benutzers auf das Windows-Verzeichnis erweitern oder den alternativen SYSTEM-Benutzer verwenden.

Um den Benutzer zu ändern, müssen Sie die globale Konfigurationsdatei `machine.config` editieren. Diese befindet sich im folgenden Verzeichnis:

```
<WINDIR>\Microsoft.NET\Framework\<Version>\Config\
```

Im Abschnitt `<processModel>` tragen Sie unter „userName“ statt des bisherigen „machine“ den neuen Wert „system“ ein. Nach einem Neustart des IIS-Dienstes oder auch nur des ASP.NET-Prozesses (`aspnet_wp.exe`) wird nun der SYSTEM-Account benutzt. Die Prozessliste sollte nun korrekt angezeigt werden.

13.5 ... eine Übersicht der Server-Services anzeigen?

Dienste oder auch Services sind ein wichtiger Bestandteil der Windows Infrastruktur. Es handelt sich um Programme, die im Hintergrund und ohne direkte Interaktion unabhängig von einem speziellen Benutzer-Account laufen. Ein naheliegendes Beispiel für einen solchen Dienst sind die Internet Information Services.

.NET bietet die Möglichkeit, auf die installierten Dienste zuzugreifen. Ein solcher Dienst wird dabei über die Klasse `ServiceController` aus dem Namespace `System.ServiceProcess` repräsentiert. Bevor Sie die Klasse nutzen können, müssen Sie die Assembly erst explizit referenzieren. Dies geschieht über die Konfigurationsdatei `web.config`:

Listing 13.7 web.config

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="System.ServiceProcess, Version=1.0.3300.0,
          Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Nun steht der Namespace zur Verfügung und kann eingebunden werden. Der genannte `ServiceController` liefert über die statische Methode `GetServices` ein Array mit allen eingerichteten Diensten. Das Array wird im folgenden Beispiel als Datenquelle für ein `DataGrid`-Control verwendet.

Listing 13.8 GetServices1.aspx

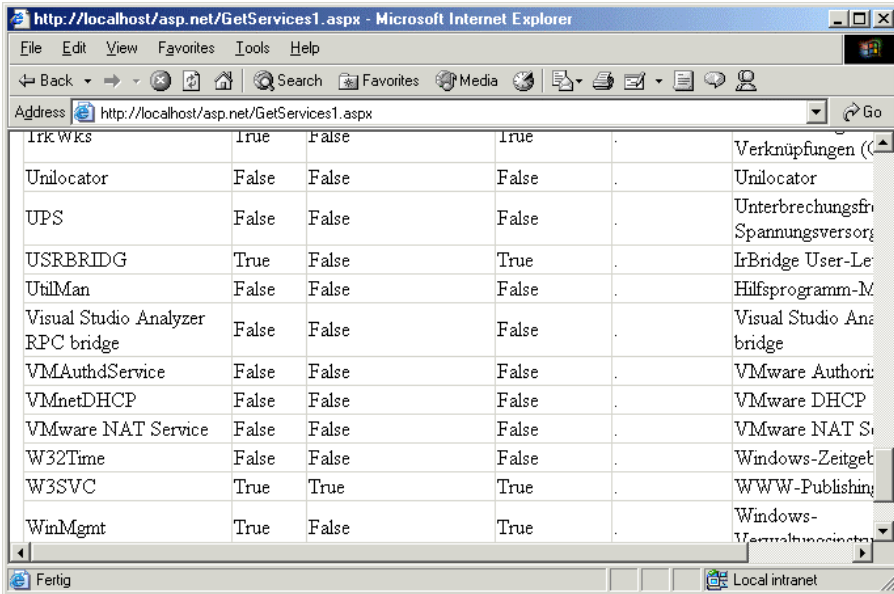
```
<% @Import Namespace="System.ServiceProcess" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
  ServiceController[] services = ServiceController.GetServices();
  dg.DataSource = services;
  DataBind();
}

</script>

<ASP:DataGrid id="dg" runat="server"/>
```



Internal Name	Public Name	Status	Internal ID	Public Name
IRK Wks	True	False	True	Verknüpfungen (<
Unlocator	False	False	False	Unlocator
UPS	False	False	False	Unterbrechungsfr
USBRIDG	True	False	True	Spannungsversorg
UtilMan	False	False	False	IrBridge User-Le
Visual Studio Analyzer	False	False	False	Hilfsprogramm-M
RPC bridge	False	False	False	Visual Studio Ana
VMAuthdService	False	False	False	bridge
VMnetDHCP	False	False	False	VMware Authori
VMware NAT Service	False	False	False	VMware DHCP
W32Time	False	False	False	VMware NAT S
W3SVC	True	True	True	Windows-Zeitget
WinMgmt	True	False	True	WWW-Publishing
				Windows-
				Verwaltungsinst

Abbildung 13.5 Die eingerichteten Dienste im Überblick

Gebunden an ein DataGridView-Control gibt dieses Detailinformationen zu den einzelnen Diensten aus. Neben dem öffentlichen Namen ist dies auch die interne ID. Die Abbildung zeigt die Übersicht im Browser-Fenster.

Alternativ können Sie auch direkt einen speziellen Dienst abfragen, indem Sie den gewünschten Namen dem Konstruktor der Klasse `ServiceController` übergeben. Das folgende Beispiel zeigt dies. Der abgefragte Dienst „W3SVC“ entspricht dem WWW-Server, also dem Web-Server der Internet Information Services. Der angezeigte Status „Running“ verwundert daher nicht wirklich.

Listing 13.9 GetServices2.aspx

```
<script runat="server">

ServiceController service;

void Page_Load(object sender, EventArgs e)
{
    service = new ServiceController("W3SVC");
    DataBind();
}

</script>

<h1> <%= service.DisplayName %> </h1>
```

```

<p>
  Name: <%# service.ServiceName %> <br>
  Type: <%# service.ServiceType %> <br>
  Status: <%# service.Status %>
</p>

```

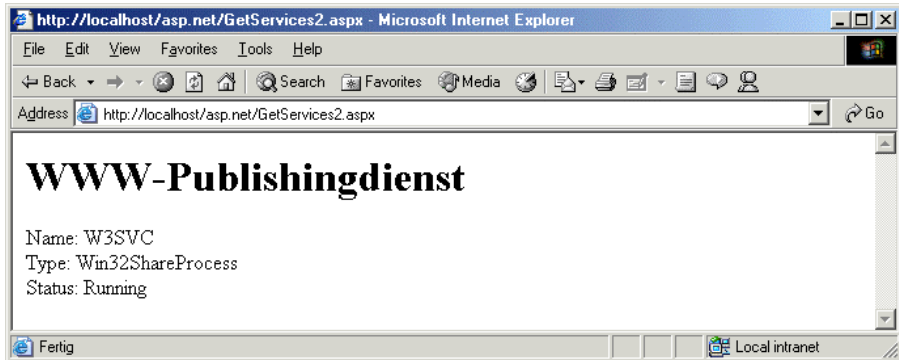


Abbildung 13.6 Der WWW-Service der IIS

13.6 ... einen Service starten oder beenden?

Im vorherigen Rezept haben Sie gelesen, wie Sie eine Liste der installierten Windows-Dienste abfragen und Detailinformationen von diesen ermitteln können. Über die Methoden der Klasse `ServiceController` ist es zudem möglich, Dienste neu zu starten, zu beenden und zu pausieren.

Das folgende Beispiel zeigt die Verwendung der Methoden anhand eines `DataGrid-Controls` mit einer Übersicht der verfügbaren Dienste. Zu jedem wird der Name sowie der aktuelle Status angezeigt. Dieser wird von der gleichnamigen Eigenschaft als Wert der Enumeration `ServiceControllerStatus` geliefert. Anhand dieses Wertes wird zudem innerhalb des `ItemCreated`-Ereignisses festgelegt, welcher der als `ButtonColumn` realisierten Buttons angezeigt werden soll.

Listing 13.10 Services1.aspx

```

<% @Import Namespace="System.ServiceProcess" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)

```

```
{
    dg.DataSource = ServiceController.GetServices();
    DataBind();
}
}

void dg_ItemCreated(object sender, DataGridItemEventArgs e)
{
    if(e.Item.DataItem != null)
    {
        ServiceController service = (ServiceController) e.Item.DataItem;
        ServiceControllerStatus status = service.Status;

        e.Item.Cells[2].Visible = (status ==
            ServiceControllerStatus.Stopped);
        e.Item.Cells[3].Visible = (status ==
            ServiceControllerStatus.Running);
        e.Item.Cells[4].Visible = (status ==
            ServiceControllerStatus.Running);
        e.Item.Cells[5].Visible = (status ==
            ServiceControllerStatus.Paused);
    }
}

void dg_ItemCommand(object sender, DataGridCommandEventArgs e)
{
    string servicename = (string) dg.DataKeys[e.Item.ItemIndex];
    ServiceController service = new ServiceController(servicename);
    ServiceControllerStatus status = service.Status;

    switch(e.CommandName)
    {
        case "start":
            service.Start();
            status = ServiceControllerStatus.Running;
            break;

        case "stop":
            service.Stop();
            status = ServiceControllerStatus.Stopped;
            break;

        case "pause":
            service.Pause();
            status = ServiceControllerStatus.Paused;
            break;

        case "continue":
            service.Continue();
    }
}
```

```
        status = ServiceControllerStatus.Running;
        break;
    }

    try
    {
        service.WaitForStatus(status, new TimeSpan(0, 0, 15));
    }
    catch {}

    dg.DataSource = ServiceController.GetServices();
    DataBind();
}

</script>

<form runat="server">

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        DataKeyField="ServiceName"
        OnItemCreated="dg_ItemCreated"
        OnItemCommand="dg_ItemCommand">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Name"
                DataField="DisplayName"/>

            <ASP:BoundColumn
                HeaderText="Status"
                DataField="Status"/>

            <ASP:ButtonColumn
                Text="Starten"
                CommandName="start"/>

            <ASP:ButtonColumn
                Text="Stoppen"
                CommandName="stop"/>

            <ASP:ButtonColumn
                Text="Pausieren"
                CommandName="pause"/>

            <ASP:ButtonColumn
                Text="Fortsetzen"
                CommandName="continue"/>
```

```
</Columns>

</ASP:DataGrid>

</form>
```

Die Buttons werden über das Ereignis `ItemCommand` des `DataGrid`-Controls behandelt. Hier entscheidet sich auf Basis des festgelegten `CommandArgument`, welche Aktion durchgeführt werden soll. Es stehen die vier Methoden `Start`, `Stop`, `Pause` und `Continue` zur Verfügung. Die notwendige Instanz der Klasse `ServiceController` wird über den Namen des Dienstes erzeugt, der als `DataKey` hinterlegt wurde.

Der abschließende Aufruf der Methode `WaitForStatus` soll sicherstellen, dass die gewünschte Aktion ausgeführt wird, bevor die Seite erneut im Browser angezeigt wird. Als `Timeout` sind 15 Sekunden angegeben. Wird der gewünschte Status innerhalb dieser Zeitspanne nicht erreicht, wird die daraus resultierende Ausnahme ignoriert und die Seite regulär angezeigt.

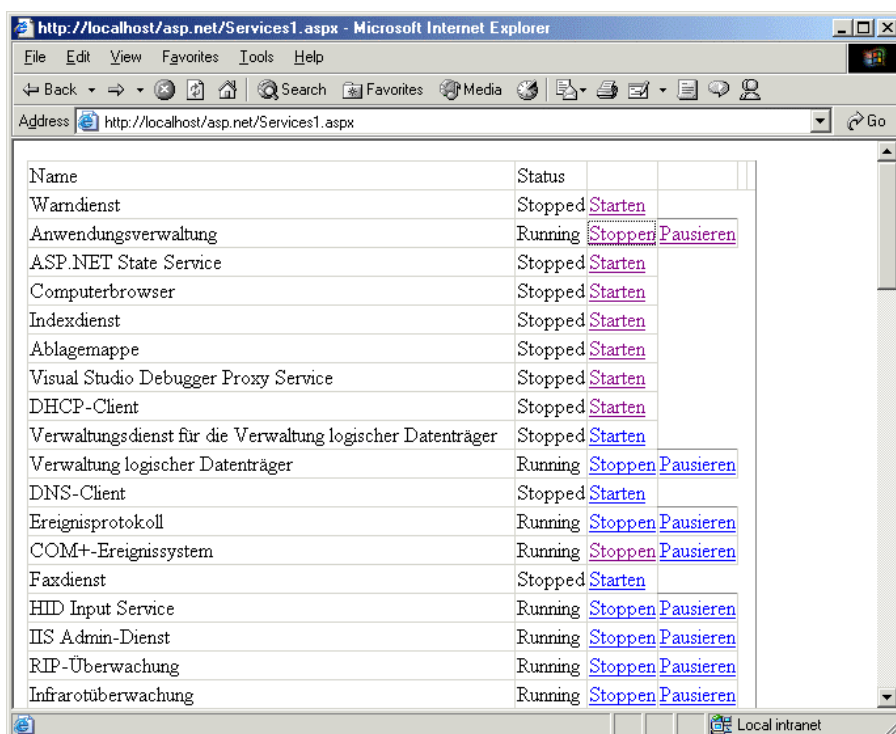


Abbildung 13.7 Jeder Dienst lässt sich individuell steuern.

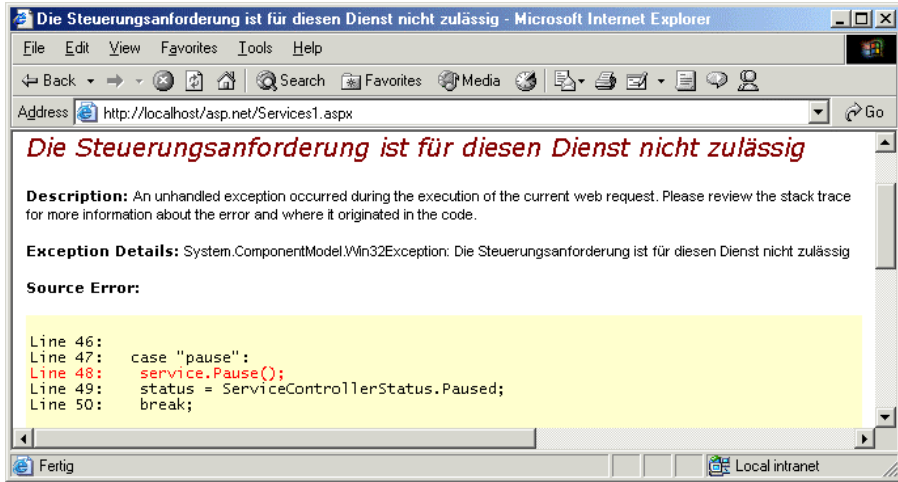


Abbildung 13.8 Der Benutzer-Account „ASPNET“ verfügt nicht über die notwendigen Rechte.

Die erste Abbildung zeigt das DataGrid-Control im Browser-Fenster. Über die angezeigten Buttons kann der Laufzeitstatus der einzelnen Dienste geändert werden. In aller Regel resultiert ein Button-Klick jedoch in einer Laufzeitmeldung, die Sie in der zweiten Abbildung sehen. Der Grund hierfür liegt in den eingeschränkten Rechten des Benutzers „ASPNET“. Dieser darf nicht aktiv auf einen Dienst zugreifen, sondern die relevanten Informationen ausschließlich lesend abfragen.

Um diesen Missstand zu beheben, haben Sie zwei alternative Möglichkeiten. Zum einen können und sollten Sie Windows Authentication benutzen, um sich mit einem Administrator-Account anzumelden. Sofern Sie Impersonation benutzen, erfolgt der Zugriff auf die Dienste mit diesem Account und ist dadurch problemlos möglich. Ein solcher Schutz ist für eine derartig wichtige und sensible Funktionalität ohnehin nur zu empfehlen.

Alternativ und *nicht* zu empfehlen ist eine Änderung des von ASP.NET verwendeten Benutzers. Hierzu müssen Sie die Konfigurationsdatei `machine.config` im folgenden Verzeichnis anpassen:

```
<WINDIR>\Microsoft.NET\Framework\<Version>\Config\
```

Weisen Sie im Abschnitt `processModel` den Attributen `userName` und `password` den gewünschten Benutzer-Account zu.

```

<processModel
...
  userName="Administrator"
  password="sagichnicht"
...

```

Wie bereits erwähnt ist diese Variante nicht zu empfehlen, da sie sich auf alle Seiten und alle Web-Applikationen auswirkt und die – sinnvollerweise – restriktiv vergebenen Benutzerrechte untergräbt.

13.7 ... das Event-Log im Browser anzeigen lassen?

Die Ereignisprotokolle der Server-Betriebssysteme aus dem Hause Microsoft bieten Administratoren eine wichtige Informationsquelle. Insbesondere nicht-visuelle Programme wie Dienste nutzen diese Protokolle, um Informationen, aber auch Fehler oder Sicherheitshinweise aufzulisten. Administratoren sollten die Protokolle daher regelmäßig sichten, um eventuelle Probleme schnell oder gar im Voraus zu erkennen.

Auch aus ASP.NET heraus ist ein Zugriff auf die Ereignisprotokolle möglich. Ein Protokoll wird dabei über die Klasse `EventLog` aus dem Namespace `System.Diagnostics` repräsentiert. Über die statische Methode `GetEventLogs` können Sie ein Array der Klasse mit allen eingerichteten Protokollen abfragen. Das Listing zeigt dies.

Listing 13.11 EventLog1.aspx

```
<% @Import Namespace="System.Diagnostics" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    EventLog[] eventlogs = EventLog.GetEventLogs();
    dg.DataSource = eventlogs;
    DataBind();
}

</script>

<ASP:DataGrid id="dg" runat="server"/>
```

Die Abbildung zeigt, dass auf dem Rechner die drei vom System vorgegebenen Protokolle „Application“, „Security“ und „System“ eingerichtet sind.

Haben Sie sich für eines der Ereignisprotokolle entschieden, können Sie sich die hinterlegten Einträge anschauen. Die Eigenschaft `Entries` liefert eine `EventLogEntryCollection`, die ihrerseits den Zugriff auf die einzelnen Instanzen der Klasse `EventLogEntry` bietet.

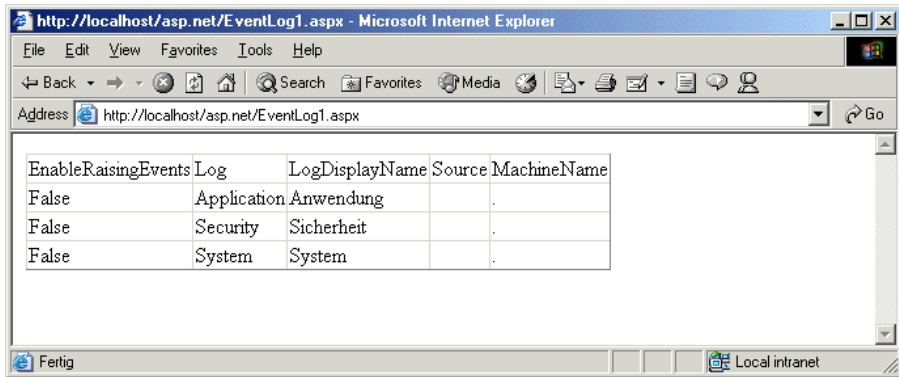


Abbildung 13.9 Die eingerichteten Ereignisprotokolle

Das zweite Beispiel erlaubt die Auswahl des gewünschten Protokolls über ein DropDownList-Control sowie die Anzeige aller darin enthaltenen Einträge in einem DataGrid-Control.

Listing 13.12 EventLog2.aspx

```
<% @Import Namespace="System.Diagnostics" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        EventLog[] eventlogs = EventLog.GetEventLogs();
        ddl.DataSource = eventlogs;
        ddl.DataBind();

        ddl.SelectedIndex = 0;
        ddl_SelectedIndexChanged(ddl, EventArgs.Empty);
    }
}

void ddl_SelectedIndexChanged(object sender, EventArgs e)
{
    string logname = ddl.SelectedItem.Value;
    EventLog eventlog = new EventLog(logname);
    dg.DataSource = eventlog.Entries;
    dg.DataBind();
}

</script>
```

```
<form runat="server">

<p>
  <ASP:DropDownList id="ddl" runat="server"
    DataValueField="Log"
    DataTextField="LogDisplayName"
    AutoPostBack="true"
    OnSelectedIndexChanged="ddl_SelectedIndexChanged"/>
</p>

<ASP:DataGrid id="dg" runat="server"/>

</form>
```

Die Abbildung zeigt das chronologisch aufgelistete Anwendungsprotokoll. Für den Administrator ist dabei insbesondere der Inhalt der Eigenschaften *Message* sowie *Source* wichtig, die den Protokolltext sowie die dahinter stehende Applikation angeben.

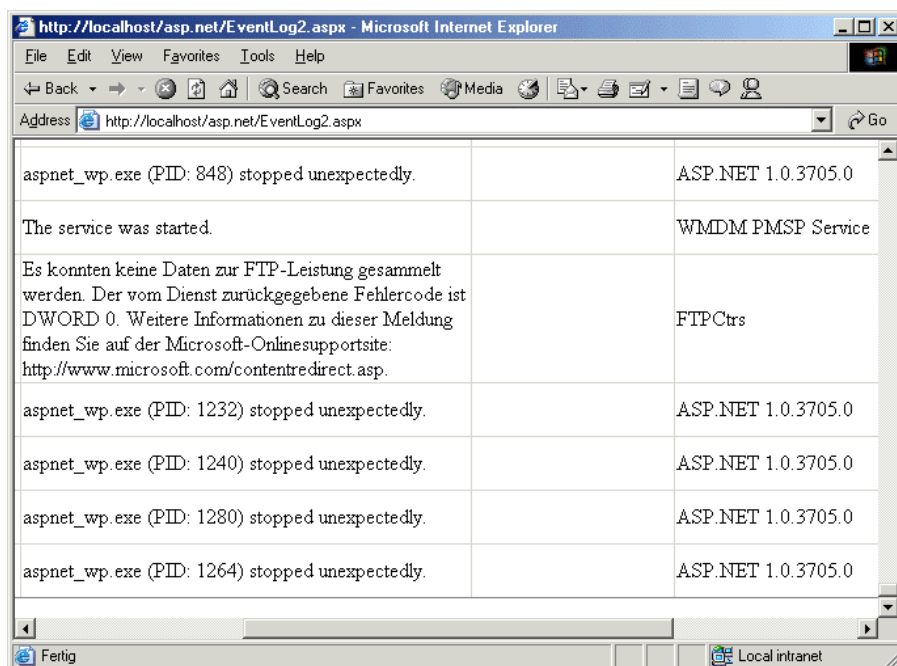


Abbildung 13.10 Das Anwendungsprotokoll

Eine weitere wichtige Eigenschaft ist *EntryType*. Diese liefert die Art des Protokolleintrags. Damit der Administrator auf einen Blick entscheiden kann, ob ein Eintrag von Relevanz ist oder nicht, verwendet die „offizielle“ Ereignisanzeige

entsprechende Symbole. Das erweiterte Beispiel im folgenden Listing macht dies ebenso und sortiert die Einträge zudem umgekehrt chronologisch. Die aktuellste Meldung ist somit gleich ganz oben sichtbar.

Listing 13.13 EventLog3.aspx

```
<% @Import Namespace="System.Diagnostics" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        EventLog[] eventlogs = EventLog.GetEventLogs();
        ddl.DataSource = eventlogs;
        ddl.DataBind();

        ddl.SelectedIndex = 0;
        ddl_SelectedIndexChanged(ddl, EventArgs.Empty);
    }
}

void ddl_SelectedIndexChanged(object sender, EventArgs e)
{
    string logname = ddl.SelectedItem.Value;
    EventLog eventlog = new EventLog(logname);

    EventLogEntry[] entries = new EventLogEntry[eventlog.Entries.Count];
    eventlog.Entries.CopyTo(entries, 0);
    Array.Sort(entries, new EventLogEntryComparer());

    dg.DataSource = entries;
    dg.DataBind();
}

public class EventLogEntryComparer : IComparer
{
    public int Compare(object x, object y)
    {
        EventLogEntry ele1 = (EventLogEntry) x;
        EventLogEntry ele2 = (EventLogEntry) y;

        return(ele1.Index.CompareTo(ele2.Index) * -1);
    }
}

void dg_ItemCreated(object sender, DataGridItemEventArgs e)
```

```
{
    EventLogEntry entry = (EventLogEntry) e.Item.DataItem;

    if(entry != null)
    {
        Image img = new Image();

        switch(entry.EntryType)
        {
            case EventLogEntryType.Error:
                img.ImageUrl = "error.gif";
                break;

            case EventLogEntryType.Warning:
                img.ImageUrl = "exclamation.gif";
                break;

            default:
                img.ImageUrl = "information.gif";
                break;
        }

        e.Item.Cells[0].Controls.Add(img);
    }
}

</script>

<form runat="server">

    <p>
        <ASP:DropDownList id="ddl" runat="server"
            DataValueField="Log"
            DataTextField="LogDisplayName"
            AutoPostBack="true"
            OnSelectedIndexChanged="ddl_SelectedIndexChanged"/>
    </p>

    <ASP:DataGrid id="dg" runat="server"
        AutoGenerateColumns="false"
        OnItemCreated="dg_ItemCreated">

        <Columns>

            <ASP:BoundColumn
                HeaderText="Typ"/>

            <ASP:BoundColumn
                HeaderText="Datum"
```

```

        DataField="TimeGenerated"/>

        <ASP:BoundColumn
            HeaderText="Quelle"
            DataField="Source"/>

        <ASP:BoundColumn
            HeaderText="Beschreibung"
            DataField="Message"/>

    </Columns>

</ASP:DataGrid>

</form>

```

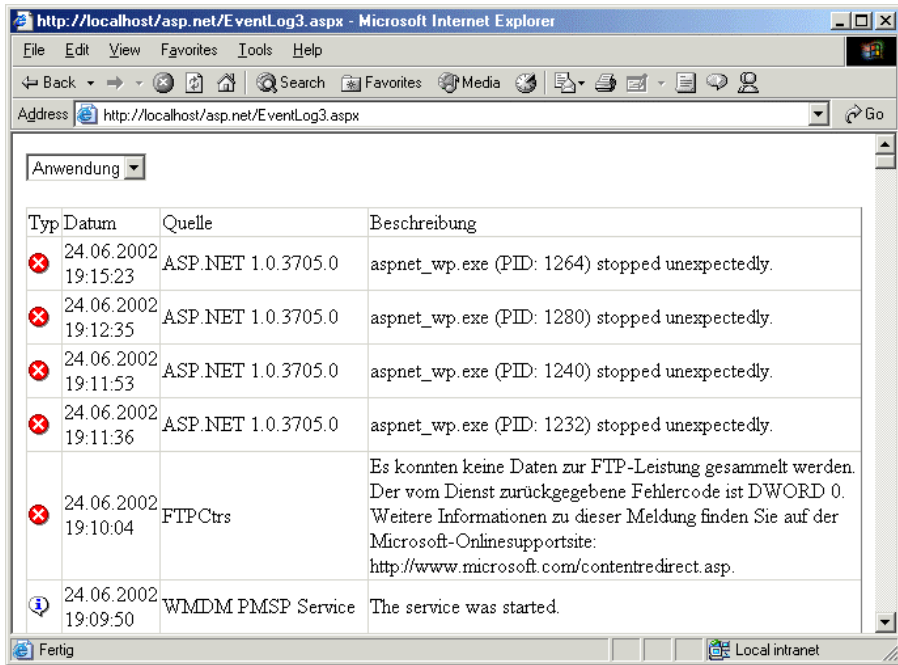


Abbildung 13.11 Die neuesten Einträge werden nun oben angezeigt.

Da die Klasse `EventLogEntryCollection` keine eigene Möglichkeit zur umgekehrt chronologischen Sortierung der Einträge bietet, musste ich ein wenig in die Trickkiste greifen. Zunächst wird der Inhalt der Collection in ein neu angelegtes `EventLogEntry-Array` kopiert. Dieses wird mit Hilfe der statischen Methode

`Array.Sort` sortiert. Neben dem Array muss die Instanz einer eigens geschriebenen Comparer-Klasse übergeben werden. Diese muss die Schnittstelle `IComparer` und somit die Methode `Compare` unterstützen. Die Abbildung zeigt das geänderte Ergebnis.

13.8 ... Ereignisse im Event-Log protokollieren?

Das vorangegangene Rezept hat den lesenden Zugriff auf die Windows-Ereignisprotokolle gezeigt. Sie können aber auch eigene Ereignisse protokollieren und somit für die Nachwelt (den Administrator) sichern. Die Klasse `EventLog` bietet eine vielfach überladene, statische Methode `WriteEntry`, der die notwendigen Informationen übergeben werden. Das Listing zeigt die Anlage eines Eintrags im Anwendungsprotokoll.

Listing 13.14 EventLog4.aspx

```
<% @Import Namespace="System.Diagnostics" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    EventLog.WriteEntry("Web-Application \"test\"", "Hier ist die Hölle
        los, ich bitte um Hilfe!", EventLogEntryType.Warning);
}

</script>
```

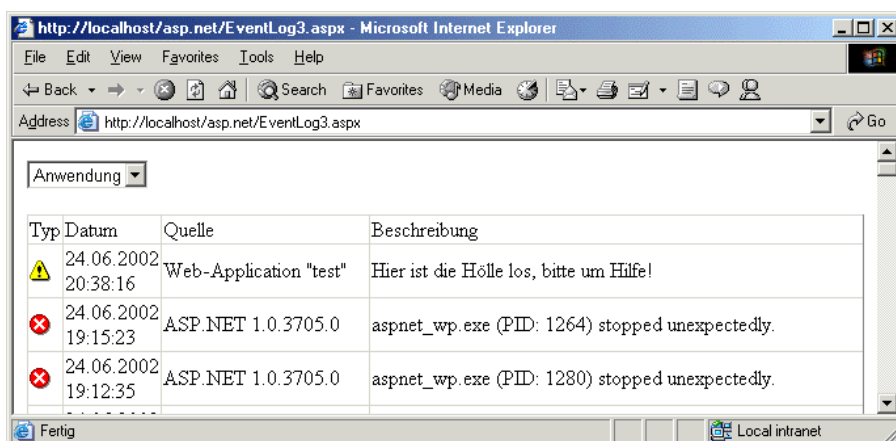


Abbildung 13.12 Der neue Eintrag wurde korrekt angelegt.

13.9 ... den Computernamen des Servers ermitteln?

Jeder Windows-Rechner verfügt über einen eigenen NETBIOS-Namen, der in den Netzwerkeinstellungen vergeben wird. Insbesondere bei Web Farms kann dieser Name zur Identifizierung des aktuellen Servers verwendet werden.

Der vergebene Name wird von der statischen Eigenschaft `MachineName` der Klasse `Environment` als Zeichenkette geliefert.

Listing 13.15 `MachineName1.aspx`

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("Ihr Rechner hört auf den Namen: <br>");
    Response.Write(Environment.MachineName);
}

</script>
```

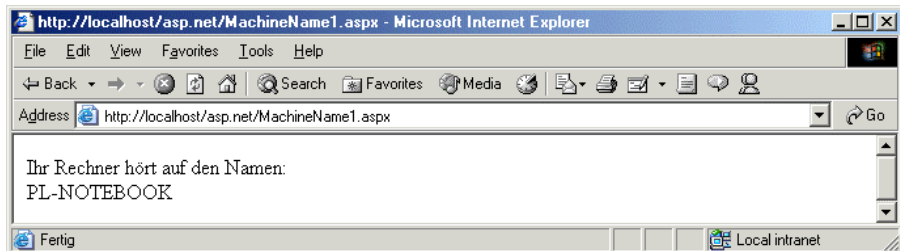


Abbildung 13.13 Der Computer-Name des Servers

13.10 ... das Windows-Verzeichnis herausfinden?

Das Windows-Verzeichnis kann über die Umgebungsvariable „windir“ abgefragt werden. Den Inhalt von derartigen Variablen liefert die statische Methode `Environment.GetEnvironmentVariable` in Form einer Zeichenkette zurück. Das Listing zeigt die Ausgabe des Windows-Verzeichnisses.

Listing 13.16 windir1.aspx

```
void Page_Load(object sender, EventArgs e)
{
    Response.Write(Environment.GetEnvironmentVariable("windir"));
}
```

Neben dem Windows-Verzeichnis stellt das Betriebssystem eine ganze Reihe weiterer Spezialverzeichnisse zur Verfügung. Hierzu gehören beispielsweise das Systemverzeichnis, der Programme-Ordner oder die „Eigenen Dateien“ des Benutzers.

Alle wichtigen Verzeichnisse lassen sich mittels der ebenfalls statischen Methode `Environment.GetFolderPath` abfragen. Übergeben wird ein Wert der verschachtelt implementierten Enumeration `Environment.SpecialFolder`. Im Listing sehen Sie die Ausgabe sämtlicher verfügbaren Spezialverzeichnisse.

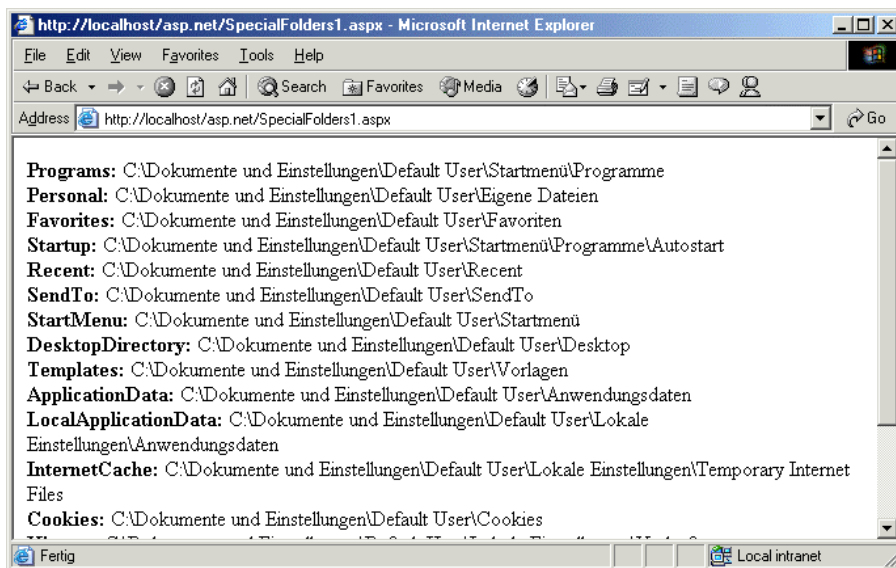


Abbildung 13.14 Die Verzeichnisse werden meist auch vom Betriebssystem genutzt.

Listing 13.17 SpecialFolders1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Type t = typeof(Environment.SpecialFolder);
    string[] names = Enum.GetNames(t);
    foreach(string name in names)
    {
```



```
Environment.SpecialFolder folder = (Environment.SpecialFolder)
    Enum.Parse(t, name);
Response.Write("<b>" + name + ":</b> ");
Response.Write(Environment.GetFolderPath(folder) + "<br>");
}
}
</script>
```

13.11 ... ermitteln, wie lange der Server schon läuft?

Web-Server laufen im Idealfall ohne Unterbrechung durch. In der Realität sieht dies meist anders aus. Mit ASP.NET soll das besser werden. Ein Indiz, ob dies tatsächlich so ist, liefert die Laufzeit. Wie lange ist der letzte Bootvorgang her?

Die statische Eigenschaft `Environment.TickCount` liefert in Form eines 64 Bit long-Wertes die Anzahl der Millisekunden seit dem Systemstart. In Verbindung mit der Struktur `TimeSpan` lässt sich so sehr übersichtlich die so genannte Uptime ermitteln. Die statische Methode `TimeSpan.FromMilliseconds` liefert eine Instanz der Struktur. Über die Eigenschaften `Days`, `Hours`, `Minutes` und `Seconds` können Sie die einzelnen Zeitelemente abfragen und im Browser ausgeben.

Listing 13.18 TickCount1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    TimeSpan ts = TimeSpan.FromMilliseconds(Environment.TickCount);
    Response.Write("Der Server läuft seit...<br>");
    Response.Write(ts.Days.ToString() + " Tagen, ");
    Response.Write(ts.Hours.ToString() + " Stunden, ");
    Response.Write(ts.Minutes.ToString() + " Minuten und ");
    Response.Write(ts.Seconds.ToString() + " Sekunden.<br>");
}

</script>
```

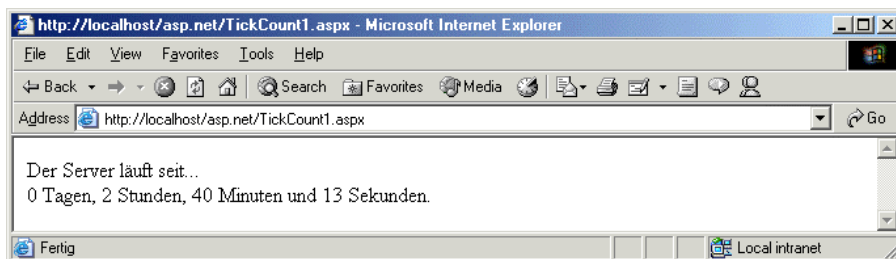


Abbildung 13.15 Würde es sich um einen Web-Server handeln, wäre das eine schlechte Zeit.



Beachten Sie bitte unbedingt, dass die genannte Eigenschaft `TickCount` eine Anzahl von Millisekunden liefert. Die Eigenschaft wurde in Anlehnung an die dahinter stehende Win32 API-Funktion `GetTickCount` benannt. Es existiert parallel eine Eigenschaft `Ticks`, die die Anzahl von 100-Nanosekunden seit dem 01.01.0001, 0:00 Uhr liefert. 100-Nanosekunden ergeben dabei einen so genannten Tick. Es handelt sich um eine vollkommen andere Einheit, die jedoch vom Namen deutlich kollidiert. Dies kann insbesondere zu Problemen führen, wenn Sie den Konstruktor der Struktur `TimeSpan` oder deren statische Methode `FromTicks` verwenden wollen. Beide funktionieren nur mit `Ticks`, nicht aber mit Millisekunden.



Der höchste positive Wert für den Datentyp `long` liegt bei 9.223.372.036.854.775.807. Ist dieser Wert erreicht, wird der `TickCount`-Zähler auf 0 gesetzt. Rechnerisch ist dies nach gut 47 Tagen ohne Neustart des Systems der Fall. Ob ein Windows-System diesen Wert jedoch jemals erreicht hat?

13.12 ... eine Umgebungsvariable abfragen?

Mitunter kann es sinnvoll sein, den Wert einer Umgebungsvariablen abzufragen. Hier sind beispielsweise Informationen wie Umgebungspfade sowie das temporäre Verzeichnis angegeben. Insbesondere bei Novell-Netzwerken existiert meist auch eine Umgebungsvariable für den Netzwerk-Benutzernamen.

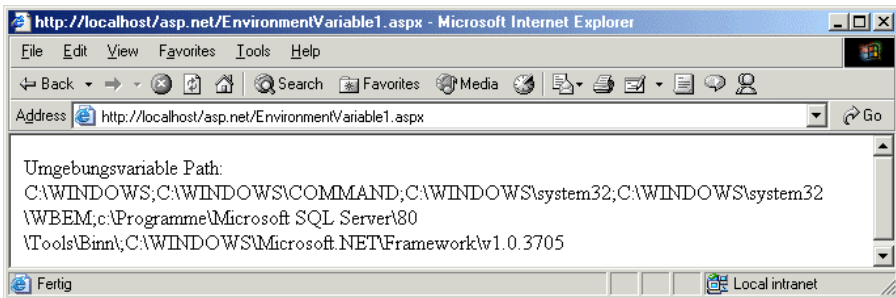
Der Zugriff auf alle vorhandenen Umgebungsvariablen erfolgt über die statische Methode `Environment.GetEnvironmentVariable`. Übergeben wird der Name der gewünschten Variablen; deren Inhalt wird als Zeichenkette zurückgeliefert. Das Listing zeigt dies anhand der bekannten Umgebungsvariable „`PATH`“.

Listing 13.19 EnvironmentVariable1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("Umgebungsvariable Path: ");
    string path = Environment.GetEnvironmentVariable("PATH");
    Response.Write(path);
}

</script>
```

**Abbildung 13.16** Der Inhalt der abgefragten Umgebungsvariablen im Browserfenster.

Alternativ lassen sich auch alle hinterlegten Umgebungsvariablen abfragen. Hierzu wird die ebenfalls statisch implementierte Methode `Environment.GetEnvironmentVariables` verwendet. Zurückgeliefert wird eine Klasse, die die Schnittstelle `IDictionary` unterstützt. Das Listing zeigt den Zugriff auf alle Variablen mit Hilfe des Dictionary-Enumerators und der davon gelieferten Klasse `DictionaryEntry`. Die Eigenschaft `Key` liefert den Namen der Variablen, die Eigenschaft `Value` deren Inhalt. Die Abbildung zeigt das Ergebnis im Browser.

Listing 13.20 EnvironmentVariable2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<h1>Alle Umgebungsvariablen</h1>");

    foreach(DictionaryEntry de in Environment.GetEnvironmentVariables())
    {
        Response.Write("<p><b>");
        Response.Write(de.Key);
        Response.Write("</b> = ");
        Response.Write(de.Value);
    }
}
```

```
Response.Write("</p>");  
}  
}  
  
</script>
```

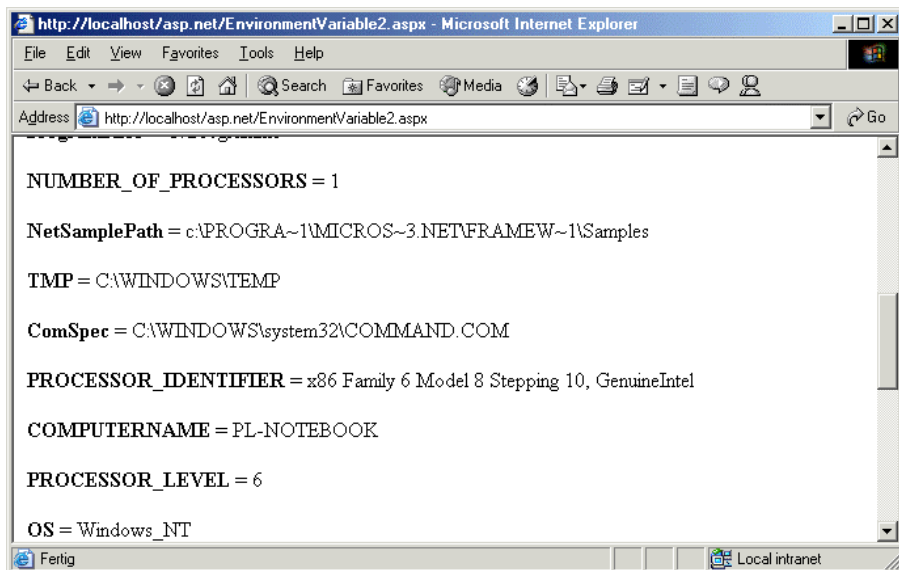


Abbildung 13.17 Alle hinterlegten Umgebungsvariablen werden im Browser ausgegeben.

13.13 ... die Version des Betriebssystems abfragen?

Neben der Version der Common Language Runtime ist auch das verwendete Betriebssystem durchaus interessant. Zurzeit werden Windows 2000 und Windows XP unterstützt. Bald gesellt sich auch die neue .NET Server-Familie hinzu.

Die derzeit eingesetzte Version können Sie über die statische Eigenschaft `Environment.OSVersion` erfragen. Sie erhalten eine Instanz der Klasse `OperatingSystem`, die über die beiden Eigenschaften `Platform` und `Version` Auskunft über die Art und die Version des Systems gibt. Es handelt sich um eine Enumeration beziehungsweise eine Instanz der Klasse `Version`. Beide lassen sich mittels der Methode `ToString` in lesbare Daten umwandeln. Das Listing zeigt dies.

Listing 13.21 OSVersion1.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<h1>Version des Betriebssystems</h1>");

    OperatingSystem os = Environment.OSVersion;
    Response.Write("Plattform: " + os.Platform.ToString() + "<br>");
    Response.Write("Version: " + os.Version.ToString() + "<br>");
}

</script>

```

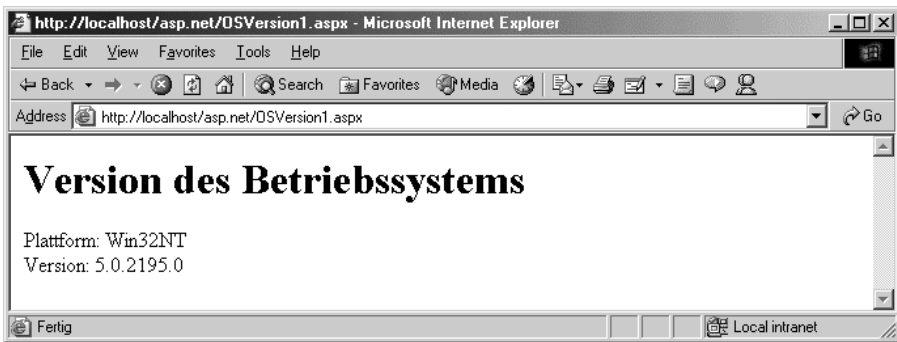


Abbildung 13.18 Auf meinem Notebook ist Windows 2000 (NT 5.0) installiert.

13.14 ... die Version der Common Language Runtime (CLR) abfragen?

Die Common Language Runtime ist das Herzstück von .NET. Das erste offizielle und ausgelieferte Release trägt die Version 1.0.3705. Weitere Versionen werden mit Sicherheit folgen. Um zu erfahren, welche derzeit auf dem Web Server installiert ist, verwenden Sie die statische Eigenschaft `Environment.Version`. Zurückgeliefert wird eine Instanz der Klasse `Version`, die über vier Eigenschaften Zugriff auf die einzelnen Elemente der Version bietet. Das Listing zeigt deren Ausgabe.

Listing 13.22 CLRVersion1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("<h1>Version der CLR</h1>");

    Version version = Environment.Version;
    Response.Write("Major: " + version.Major.ToString() + "<br>");
    Response.Write("Minor: " + version.Minor.ToString() + "<br>");
    Response.Write("Revision: " + version.Revision.ToString() + "<br>");
    Response.Write("Build: " + version.Build.ToString() + "<br>");
}

</script>
```

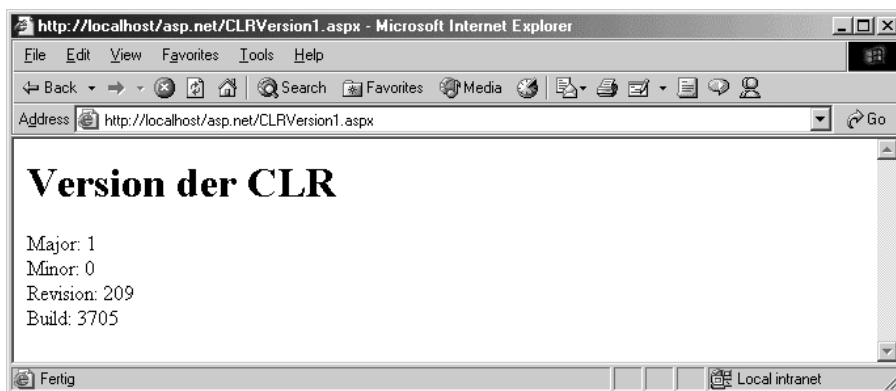


Abbildung 13.19 Auf meinem System ist das erste Release der CLR installiert.

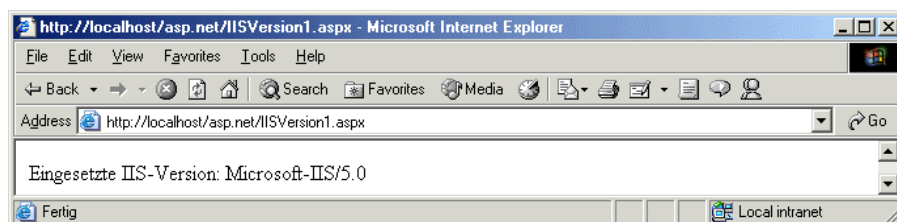
13.15 ... die Version der verwendeten Internet Information Services abfragen?

Im Grunde lässt sich diese Frage über das von Ihnen eingesetzte Betriebssystem beantworten. ASP.NET lässt sich ab Windows 2000 mit IIS Version 5.0 einsetzen. Darüber hinaus gilt die Tabelle.

Tabelle 13.1 Die IIS-Version ist abhängig vom Betriebssystem

Betriebssystem	IIS Version
Windows 2000	IIS Version 5.0
Windows XP	IIS Version 5.1
.NET Server	IIS Version 6.0

Möchten Sie die eingesetzte Version schwarz auf weiß abfragen, dann lassen Sie sich einfach die Server-Variable „Server_Software“ ausgeben.

**Abbildung 13.20** ... folgerichtig verwende ich Windows 2000 (Professional)

13.16 ... die IIS vor Viren und Eindringlingen schützen?

Die Internet Information Services sind in den vergangenen Monaten immer wieder in die Fachpresse geraten. Grund hierfür waren immer neue Sicherheitslücken, die durch immer neue Hot-Fixes des Herstellers beseitigt wurden. Die meisten dieser Patches haben ein Problem gelöst und andere überhaupt erst möglich gemacht. Wieder andere Probleme wurden zwar erkannt, die entsprechenden Patches jedoch von vielen Betreibern nicht eingespielt und so kurzerhand zur extremen Verbreitung von Viren benutzt.

Ich kenne das Problem aus eigener Erfahrung. Es ist schon recht schwierig, bei all den vielen Service Packs, Hot-Fixes und Konfigurationsmöglichkeiten den Überblick zu behalten. Nachdem Microsoft bereits längere Zeit ein Kommandozeilenprogramm zur Überprüfung auf neue Hot-Fixes zur Verfügung gestellt hat, ist mittlerweile eine grafische Analyse-Software zum kostenlosen Download verfügbar.

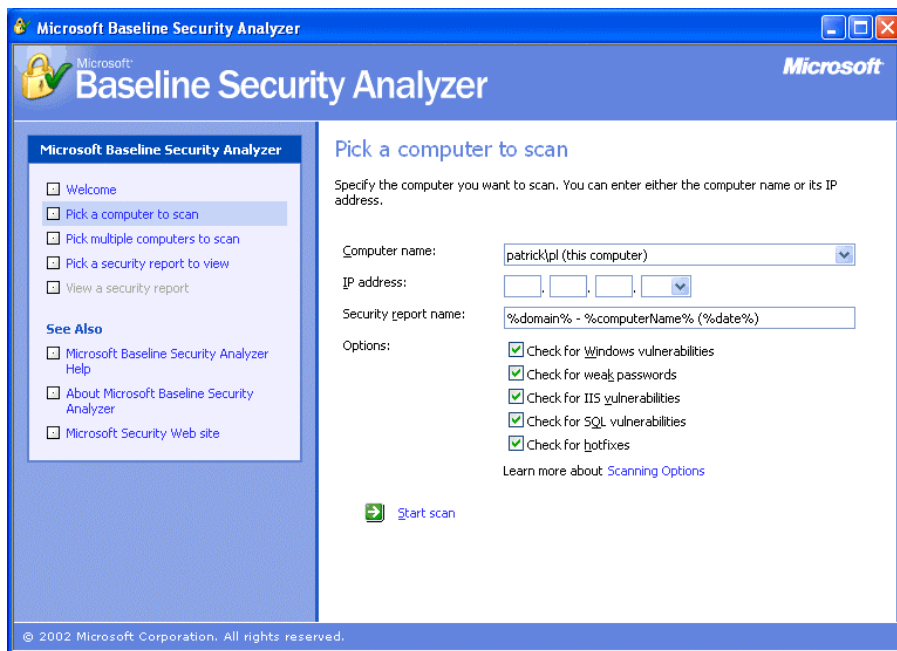


Abbildung 13.21 Die Software erlaubt auch das Scannen von Remote-Rechnern.

Der Microsoft Baseline Security Analyser überprüft das lokale System, einen Remote-Rechner oder ganze IP-Adressräume. Dabei werden ganz unterschiedliche und vielfältige Informationen abgefragt und analysiert. Hierzu zählen beispielsweise die installierten Service Packs und Hot-Fixes. Eine aktuelle Liste wird beim Start des Scans heruntergeladen, so dass das Programm immer auf dem letzten Stand ist. Auch sicherheitsrelevante Informationen werden überprüft. Existieren etwa noch die Beispielsverzeichnisse der IIS, werden Sie zum Löschen aufgefordert.

Im Anschluss an den mehrminütigen Scanvorgang wird eine übersichtliche Liste mit allen überprüften Elementen angezeigt. Ein Icon informiert über den Status und gibt bekannt, ob der Bereich korrekt ist oder eine Benutzerinteraktion notwendig ist. Sollen beispielsweise neue Hot-Fixes eingespielt werden, so listet diese eine Detailliste auf und ermöglicht den direkten Download.

Der Microsoft Baseline Security Analyser nimmt Ihnen das Mitdenken und die Verantwortung für den Server nicht ab, bietet aber dennoch eine übersichtliche Möglichkeit, wichtige Service Packs und Hot-Fixes sowie häufige Sicherheitsprobleme zu erkennen und zu beseitigen.

Der Scanvorgang sollte regelmäßig durchlaufen und alle vorgeschlagenen, sinnvollen Änderungen durchgeführt werden. Hierzu zählt insbesondere auch die Verwendung des IIS Lockdown Tools, das die IIS auf die wenigsten, tatsächlich benötigten Leistungen reduziert.

Sie erhalten den Baseline Security Analyzer kostenlos von Microsoft über die folgende Adresse:

<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/Tools/MBSAhome.asp>

Ich empfehle in diesem Zusammenhang auch, die Security-Mailing-Liste von Microsoft zu abonnieren. Aktuelle Sicherheitslücken und entsprechende Hot-Fixes werden hier umgehend publiziert. Die Liste stellt insofern eine wichtige Informationsquelle für alle Administratoren dar.

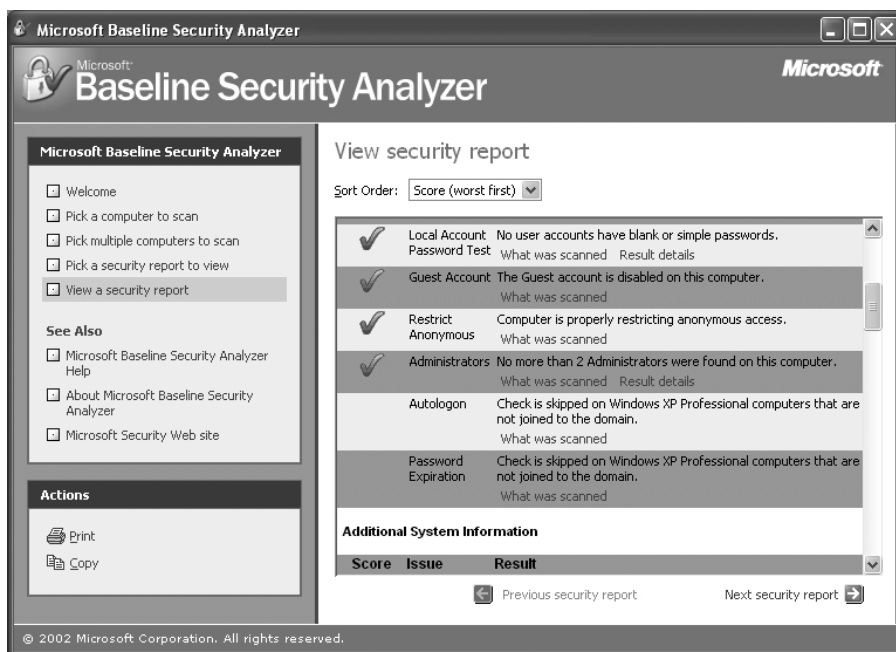


Abbildung 13.22 Das Ergebnis des Scans wird übersichtlich angezeigt.

13.17 ... festlegen, wann die ASP.NET-Engine neu gestartet wird?

Die ASP.NET-Engine verwaltet mehrere Prozesse, die die einzelnen Client-Anfragen parallel bearbeiten und beantworten. Ein solcher Thread wird Worker Process genannt. In der globalen Konfigurationsdatei `machine.config` können Sie

angeben, nach welcher Zeitspanne ein solcher Prozess beendet und dessen Aufträge an einen neuen Prozess übergeben werden sollen. Auf diese Weise kann je nach Anwendung die Stabilität des Servers erhöht werden. Der Standardwert ist jedoch „Infinite“, der Prozess wird also unendlich benutzt und nicht neu gestartet.

```
<processModel
  timeout="00:10:00" <!-- 10 Minuten -->
  ...
```

Über eine weitere Einstellung kann die Zeitspanne angegeben werden, nach der der Arbeitsprozess im Leerlauf komplett beendet wird. Auch hier ist als Standardwert „Infinite“ hinterlegt.

```
<processModel
  idleTimeout="00:10:00" <!-- 10 Minuten -->
  ...
```



Beachten Sie bitte, dass die Konfigurationssektion `<processModel>` nur bei Verwendung der Internet Information Services Version 5.0 ausgewertet wird. Ab Version 6.0 wird die Konfiguration direkt in der zur Verfügung gestellten Management Console der IIS vorgenommen.

Die Konfiguration des Prozessmodells lässt sich weiter verfeinern. Hierzu stehen weitere Einstellungsmöglichkeiten zur Verfügung. Eine ausführliche Beschreibung mit den jeweils möglichen Werten finden Sie in der .NET Framework SDK-Dokumentation.

13.18 ... die ASP.NET-Engine (neu) registrieren?

ASP.NET wird als ISAPI Extension installiert, also als Erweiterung der Internet Information Services (IIS). Mitunter ist eine Neuinstallation oder Konfiguration notwendig. Über die entsprechende Management Console (MMC) können Sie die Einstellungen manuell vornehmen. Sehr viel einfacher geht dies jedoch mit Hilfe des Kommandozeilenprogramms `aspnet_regiis.exe`, das mit dem Framework ausgeliefert wird. Sie finden das Programm im folgenden Verzeichnis:

```
<WINDIR>\Microsoft.NET\Framework\<Version>\
```

Die Steuerung erfolgt über eine ganze Reihe von unterschiedlichen Parametern. So können Sie eine (Neu-)Installation beispielsweise mittels `-i` durchführen:

```
aspnet_regiis -i
```

Anschließend sollten noch die benötigten Script-Dateien zum Beispiel für die clientseitige Validierung für Eingabeformulare kopiert werden. Dies geschieht über den Parameter `-c`.

```
aspnet_regiis -c
```

Benötigen Sie ASP.NET nicht mehr, so können Sie dieses auch deinstallieren. Hier stehen zwei analoge Parameter `-u` und `-e` für die ISAPI Extension sowie die Scripts zur Verfügung.

```
aspnet_regiis -u
aspnet_regiis -e
```



Beachten Sie bitte, dass lediglich die Konfiguration der IIS zurückgesetzt wird, die Engine oder das Framework selbst aber nicht gelöscht werden. Dies können Sie wie gewohnt über den Software-Dialog der Systemsteuerung vornehmen.

Das Tool kennt eine Reihe weiterer Parameter. Eine Übersicht können Sie durch einfachen Aufruf des Programms in der Eingabeaufforderung `cmd.exe` vornehmen.

13.19 ... ASP.NET auf einem anderen Web-Server als den IIS verwenden?

Microsoft unterstützt derzeit ausschließlich die Integration von ASP.NET in die Internet Information Services ab Version 5.0, also Windows 2000 und Windows XP. Auch die Verwendung der Version 6.0 unter dem .NET Server wird selbstverständlich möglich sein. Andere Systeme oder Server werden *nicht* unterstützt, und laut Aussagen des Herstellers gibt es derzeit auch keine entsprechenden Planungen.

Durch die offene Struktur des .NET Frameworks ist die Implementierung eines individuellen Hosts für ASP.NET jedoch relativ einfach mit Bordmitteln zu bewerkstelligen. Von daher wird es nur eine Frage der Zeit sein, bis es freie Implementierungen für andere Web-Server wie Apache von Drittherstellern oder gar Privatpersonen geben wird.

Da das .NET Framework ausschließlich für Windows-basierte Systeme verfügbar ist, wird die Implementierung für weitere Web-Server wohl zunächst ebenfalls auf diese Welt eingeschränkt bleiben. Das Mono-Projekt arbeitet bereits seit einiger Zeit an einer Portierung des Frameworks auf Linux. Sollte dieser Vorgang

abgeschlossen sein, darf man sicher auch eine ASP.NET-Unterstützung für in diesem Umfeld verbreitete Web-Server wie den original Linux-Apache erwarten. Informationen über den aktuellen Stand des Projekts finden Sie unter folgender Adresse:

<http://www.go-mono.com>

13.20 ... die IP-Adresse des Benutzers ermitteln?

Eine echte Authentifizierung eines eigentlich anonymen Internet-Benutzers ist für den Betreiber schlichtweg unmöglich. Um zumindest in der Theorie eine Handhabe zu bekommen, speichern gerade Anbieter eines Internet-Shops die IP-Adresse des Besuchers samt Zeitstempel ab. Auf diese Weise ist theoretisch mit Hilfe der Protokolldateien des jeweiligen Zugangsproviders eine Identifizierung möglich. Zugang zu diesen Daten erhalten jedoch nur Ermittlungsbehörden im Zuge eines Strafverfahrens. Dennoch, die Speicherung kann nicht schaden.

Die Ermittlung der IP-Adresse ist relativ einfach. Sie wird als Zeichenkette von der Eigenschaft `Request.UserHostAddress` zurückgeliefert. Das Listing zeigt die Ausgabe mitsamt eines Zeitstempels.

Listing 13.23 ip1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Response.Write("Big Brother is watching you!<br>");
    Response.Write("IP: " + Request.UserHostAddress + "<br>");
    Response.Write("Time: " + DateTime.Now.ToString() + "<br>");
}

</script>
```

Die Abbildung zeigt die Ausgabe im Browserfenster. Typischerweise würden die Angaben in einer Datenbank abgespeichert werden. Die gezeigte IP-Adresse 12.7.0.0.1 ist natürlich ein schlechtes Beispiel, da es sich um die lokale Adresse handelt, die auf jedem Rechner analog verwendet wird.

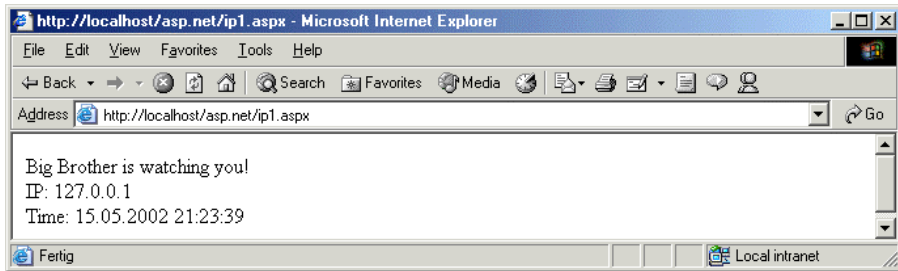


Abbildung 13.23 Zumindest theoretisch können Sie den Benutzer hiermit identifizieren.

13.21 ... einen Host-Namen auflösen?

IP-Adressen können eine ganze Menge verraten. Insbesondere, wenn diese per DNS aufgelöst werden. Sofern vorhanden kann so der zugewiesene Hostname ermittelt werden. Auf diese Weise können Sie oftmals erfahren, über welchen Provider sich ein Benutzer eingewählt hat, T-Namen wie ...t-dialin.net sind dabei überproportional vertreten.

Ist die IP-Adresse des Benutzers vorhanden (vergleiche Rezept „... die IP-Adresse des Benutzers ermitteln?“), reicht ein Aufruf der statischen Methode `Resolve` der Klasse `Dns`, um den zugehörigen Hostnamen aufzulösen. Zurückgeliefert wird eine Instanz der Klasse `IPHostEntry` mit den gewünschten Informationen. Das Listing gibt das Ergebnis im Browser aus.

Listing 13.24 dns1.aspx

```
<% @Import Namespace="System.Net" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string ip = Request.UserHostAddress;
    IPHostEntry host = Dns.Resolve(ip);
    Response.Write("Ihr Host: " + host.HostName + "<br>");
}

</script>
```

Rufen Sie das Beispiel lokal auf, wird der Windows-Name des Rechners ausgegeben. In meinem Fall erhalte ich „pl-notebook“ im Browserfenster.

Auch umgekehrt funktioniert die Methode `Resolve`, um zu einem Hostnamen die zugehörige IP-Adresse zu ermitteln. Hierzu übergeben Sie einfach den gewünschten Namen. Die Adresse wird in Form eines `IPAddress`-Arrays über die Eigenschaft `AddressList` geliefert. Es sollte mindestens eine IP vorhanden sein, so dass das Array explizit auf das erste Element hin abgefragt werden kann.

Listing 13.25 dns2.aspx

```
<% @Import Namespace="System.Net" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string hostname = "localhost";
    IPEndPoint host = Dns.Resolve(hostname);
    Response.Write("Ihre IP: " + host.AddressList[0].ToString() +
        "<br>");
}

</script>
```

Der Name „localhost“ ist für den lokalen Rechner reserviert, und so führt das Listing zur Ausgabe der lokalen IP-Adresse 127.0.0.1.

13.22 ... eine Datei serverseitig von einem anderen Server herunterladen?

Manchmal ist es notwendig, eine Datei von einem fremden Server herunterzuladen. Mit Hilfe der Klasse `WebClient` aus dem Namespace `System.Net` ist dies sehr einfach zu realisieren. Sie können die angegebene Datei entweder direkt lokal abspeichern lassen und einen `Stream` auf die zurückgelieferten Daten abfragen.

Die Listings zeigen die beiden unterschiedlichen Varianten. Im ersten wird eine Bilddatei vom (lokalen) Server heruntergeladen, unter einem temporären Namen abgespeichert und mittels `WriteFile` an den Client übertragen.

Listing 13.26 DownloadFile1.aspx

```
<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
```

```

WebClient webclient = new WebClient();

string remoteFile = "http://localhost/asp.net/paramount.gif";
string localFile = Path.GetTempFileName();
webclient.DownloadFile(remoteFile, localFile);

Response.ContentType = "image/gif";
Response.WriteFile(localFile);
}
</script>

```

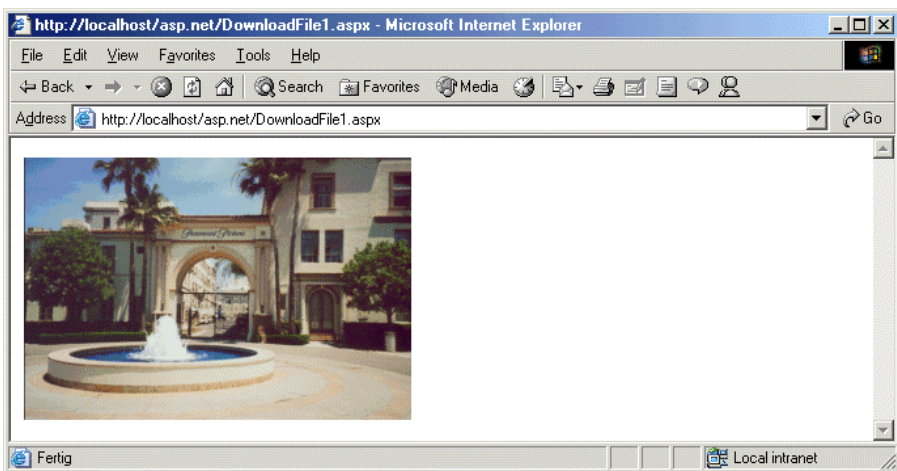


Abbildung 13.24 Das Bild wurde dynamisch vom Server heruntergeladen.

Alternativ haben Sie die Möglichkeit, eine Stream-Instanz mit den heruntergeladenen Daten zu erhalten. Auf diese Weise sparen Sie die sonst notwendigen Dateioperationen. Der Stream wird von der Methode `OpenRead` zurückgeliefert. Zur Übertragung der Daten an den Client werden diese innerhalb einer Schleife aus dem Stream ausgelesen und an den `OutputStream` der Response-Klasse übergeben.

Listing 13.27 DownloadFile2.aspx

```

<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    WebClient webclient = new WebClient();

```

```
string remoteFile = "http://localhost/asp.net/paramount.gif";

Stream stream = webclient.OpenRead(remoteFile);

Response.ContentType = "image/gif";

byte[] bytes = new byte[4096];
int count;
do
{
    count = stream.Read(bytes, 0, 4096);
    if(count > 0) Response.OutputStream.Write(bytes, 0, count);
} while(count > 0);
}

</script>
```

Alternativ zu der Klasse `WebClient` können Sie auch die Klasse `WebRequest` zum Download einer Datei verwenden. Die abstrakte Basisklasse bietet eine statische Methode `Create`, über die eine Instanz der abgeleiteten Klasse `HttpWebRequest` für die übergebene Adresse angefordert werden kann. Hierüber lässt sich analog zum vorherigen Ansatz ein `Stream` abfragen.

Listing 13.28 DownloadFile3.aspx

```
<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string remoteFile = "http://localhost/asp.net/paramount.gif";

    HttpWebRequest request =
        (HttpWebRequest) WebRequest.Create(remoteFile);
    HttpWebResponse response = (HttpWebResponse) request.GetResponse();

    Stream stream = response.GetResponseStream();

    Response.ContentType = "image/gif";

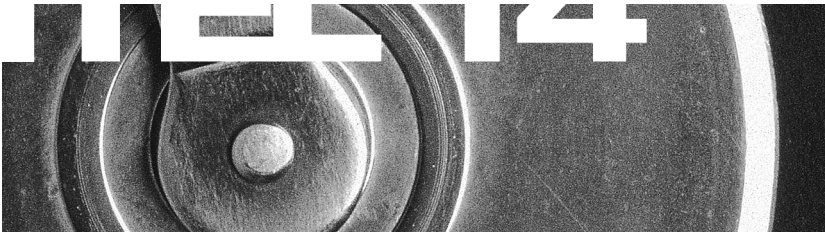
    byte[] bytes = new byte[4096];
    int count;
    do
    {
        count = stream.Read(bytes, 0, 4096);
```


614 _____ 13.22 ... eine Datei serverseitig von einem anderen Server herunterladen?

```
    if(count > 0) Response.OutputStream.Write(bytes, 0, count);  
  } while(count > 0);  
}  
  
</script>
```

14

XML Web Services



Wie kann ich ...

14 XML Web Services

Web Services sind der Schlachtruf, mit dem Microsoft in den Kampf um die absolute Vorherrschaft in der Enterprise-Entwicklung zieht. Die Technik dahinter ist jedoch mitnichten neu oder gar eine reine Erfindung aus Redmont. Hinter Web Services steht SOAP, also XML mit HTTP als Transportmedium.

Wenn nicht die Technik, was macht Web Services bei .NET denn wirklich aus? Es ist die Einfachheit, mit der sich die Dienste entwickeln und vor allem auch nutzen lassen. Einfach eine Datei mit der Endung `asmx` anlegen und auf der anderen Seite eine Web-Referenz hinzufügen. Wenn das nicht einfach ist, was dann?

14.1 ... einen Web Service erstellen?

Wie einfach die Erstellung eines XML Web Services wirklich ist, soll dieses Rezept zeigen. Der geplante Web Service dient der Überprüfung einer Email-Adresse auf Basis eines regulären Ausdrucks. Hierzu soll eine öffentliche `CheckRegex` die zu prüfende Adresse entgegennehmen und deren syntaktische Korrektheit als booleschen Wert zurückliefern.

Listing 14.1 CheckMail1.asmx

```
<% @WebService Class="CheckMail" Language="C#" Debug="true" %>

using System;
using System.Web.Services;
using System.Text.RegularExpressions;

[WebService(
    Name="CheckMail",
    Description="Ueberpruefung einer Email-Adresse",
    Namespace="http://www.asp-buch.de")
]
public class CheckMail : WebService
{
    [WebMethod(Description="Ueberprüft eine Adresse
        mittels Regular Expressions")]
```

```
public bool CheckRegex(string email)
{
    Regex r = new Regex(@"^[w\.-]+@([w\.-]+\.)*
        [w\.-]{2,63}\.[a-zA-Z]{2,4}$");
    return(r.IsMatch(email));
}
```

Die Datei hat die Endung `asmx`, wobei das „m“ für „Method“, also für eine öffentliche Web-Methode, steht. Das Listing zeigt die verschiedenen Elemente eines Web Services. Die `@WebService`-Direktive legt zunächst analog zu `@Page` grundlegende Einstellungen wie die verwendete Sprache und den Debug-Modus fest. Zudem muss der Name einer nachfolgenden Klasse angegeben werden.

Diese Klasse wird anders als bei Dateien mit der Endung `aspx` in regulärer Quellcode-Manier notiert und muss sich von der Basis `WebService` ableiten. Über das `WebService`-Attribut können zusätzliche Meta-Informationen angegeben werden.

Die öffentlich zur Verfügung stehenden Methoden müssen einerseits als `public` markiert sein und andererseits mit dem `WebMethod`-Attribut versehen werden. Optional kann hier eine Beschreibung angegeben werden. Die Methode selbst wird wie gewohnt implementiert.

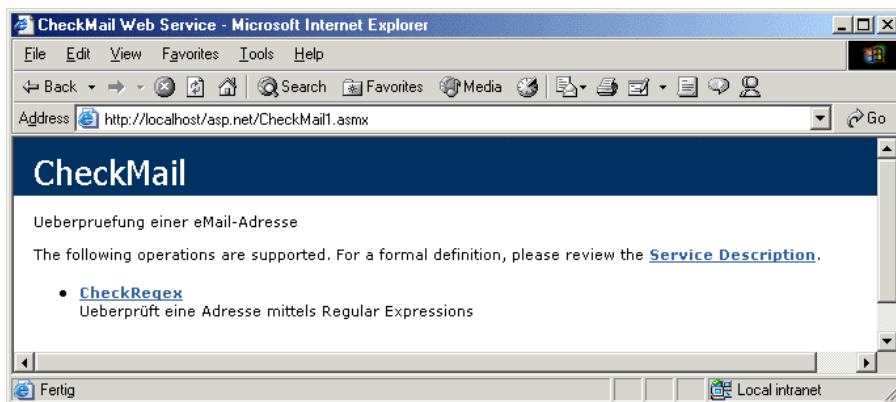


Abbildung 14.1 Die Übersichtsseite wird automatisch erzeugt.

Nach der Anlage eines Web Services als `asmx`-Datei können Sie die entsprechende Adresse im Browser aufrufen. ASP.NET erstellt automatisch eine Hilfeseite, die Ihnen eine Übersicht über den Dienst und alle angebotenen Methoden liefert. Die Abbildung zeigt eine solche Seite für den zuvor gezeigten Web Service `CheckMail`. Die Hilfeseite erlaubt Ihnen auch, einfache Web Services direkt auszuprobieren. Folgen Sie dazu einfach dem Link hinter dem gewünschten Methodennamen. Sie erhalten nun eine Übersicht der Methode. Sofern es sich bei den Parametern um

einfache Datentypen wie Zeichenketten, numerischen Werte und so weiter handelt, stehen Ihnen zur Eingabe Textfelder zur Verfügung. Die zweite Abbildung zeigt dies.

Im Beispiel der Web-Methode `CheckRegex` können Sie für den Parameter `email` eine selbige angeben. Ein Klick auf `Invoke` übergibt den Wert an den Web Service und zeigt das zurückgelieferte XML-Ergebnis in einem zweiten Fenster. Deutlich erkennbar ist der boolesche Rückgabewert, der, sofern Sie eine syntaktisch korrekte Email-Adresse eingegeben haben, `true` sein sollte.

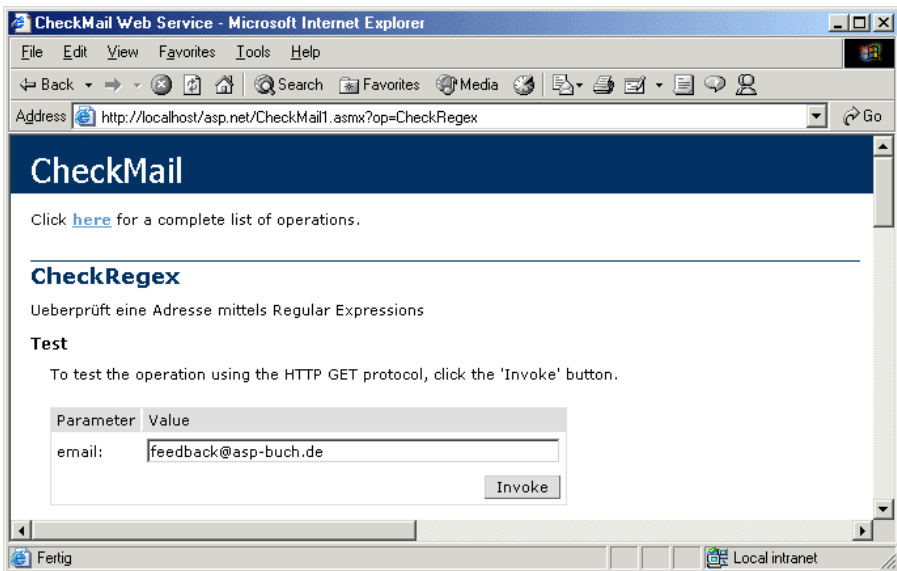


Abbildung 14.2 Sie können einfache Web Services direkt ausprobieren.

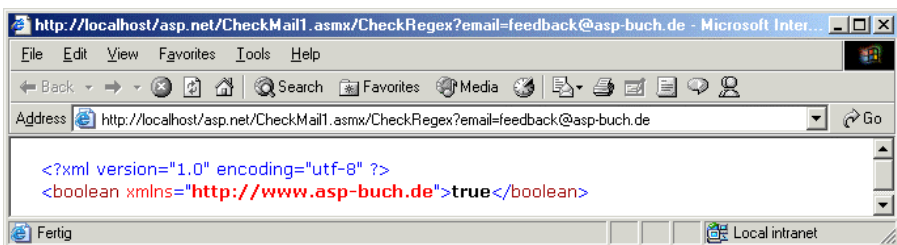


Abbildung 14.3 Das Ergebnis eines Web Services wird als XML-Stream geliefert.

14.2 ... einen Web Service mit Visual Studio .NET konsumieren?

Das Einbinden eines XML Web Services in einer Visual Studio .NET Solution ist denkbar einfach. Wer die Entwicklungsumgebung für teuer Geld erworben hat, bekommt also durchaus etwas geboten.

- Nachdem Sie eine neue Web-Applikation angelegt haben, wählen Sie den Menübefehl Project > Add Web Reference... an.
- Es wird nun ein kleiner Browser angezeigt. In der Adressleiste können Sie die Ihnen bekannte Adresse des Web Services angeben und die Eingabe bestätigen. Ich verwende nachfolgend das Beispiel CheckMail aus dem vorherigen Rezept.
- Der Web Service wird kontaktiert und es werden parallel die HTML-Ansicht sowie das beschreibende WSDL-Dokument geladen.
- Ist die Wahl korrekt, können Sie eine Referenz auf den gewählten Dienst über den Button „Add Reference“ einfügen.

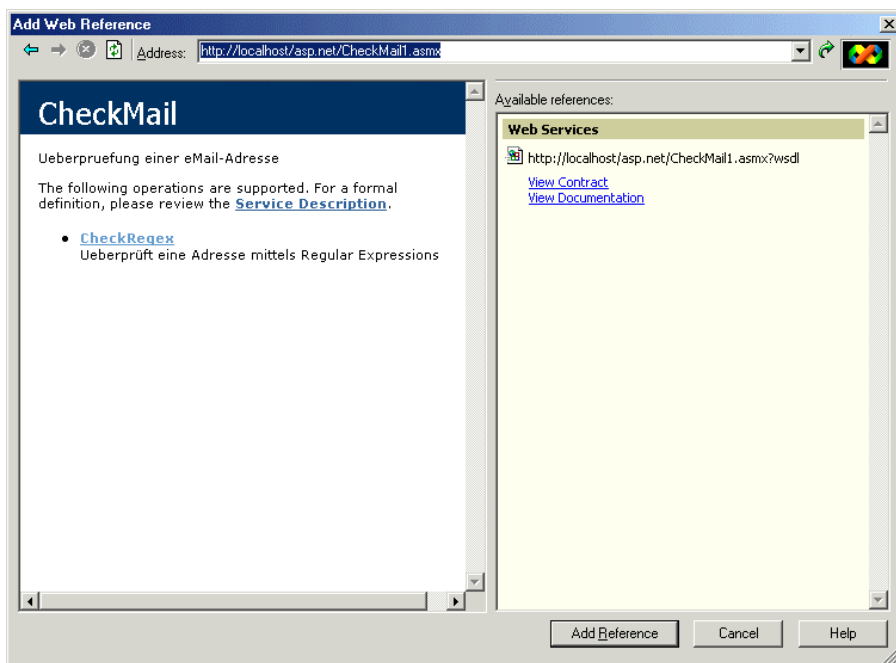


Abbildung 14.4 Die Web-Referenz lässt sich ganz einfach einfügen.

Ist der XML Web Service eingefügt, können Sie diesen sogleich benutzen. Im Beispiel habe ich ein kleines Web-Formular entworfen. Dieses enthält eine TextBox, ein Label und einen Button. Wird eine Email-Adresse eingegeben und der Button

betätigt, wird in der entsprechenden Ereignisbehandlung der Web Service instanziiert und wie eine lokale Klasse verwendet. Der boolesche Rückgabewert entscheidet über die positive oder negative Nachricht, die den Benutzer über das Label-Control erreicht.

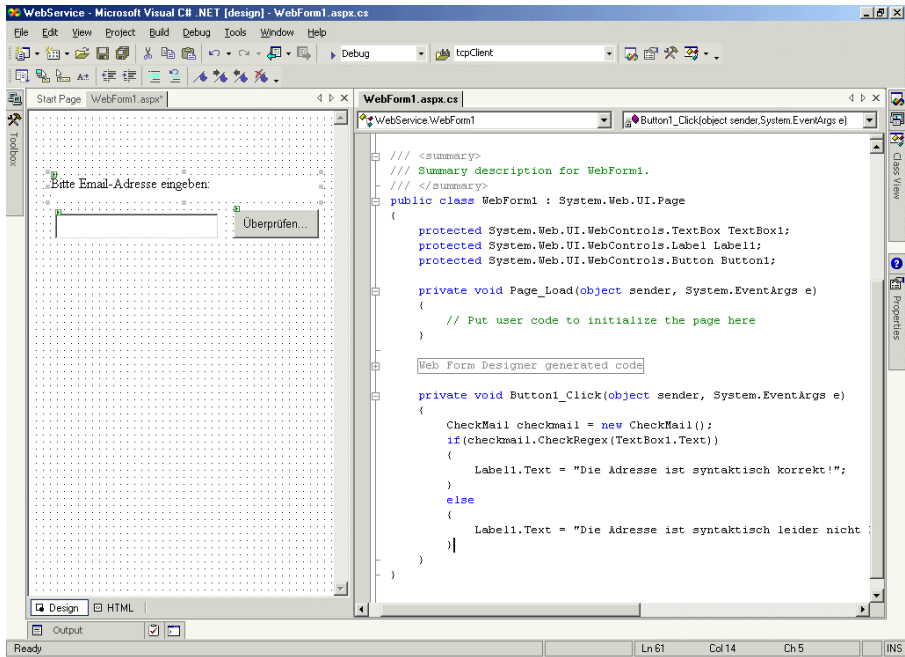


Abbildung 14.5 Der Web Service lässt sich wie eine lokale Klasse verwenden.

Listing 14.2 WebForm1.aspx.cs

```
private void Button1_Click(object sender, System.EventArgs e)
{
    CheckMail checkmail = new CheckMail();
    if(checkmail.CheckRegex(TextBox1.Text))
    {
        Label1.Text = "Die Adresse ist syntaktisch korrekt!";
    }
    else
    {
        Label1.Text = "Die Adresse ist syntaktisch leider nicht korrekt!";
    }
}
```

Die Abbildung zeigt das Ergebnis des neuen Projekts im Browserfenster. Ein einfacher Buttonklick genügt tatsächlich, um den Web Service aufzurufen. Dabei ist es ganz unerheblich, ob dieser auf dem lokalen System oder gar auf einem Internet-Server auf der anderen Seite der Erde läuft. Oh Du schöne neue Entwicklerwelt ;-)

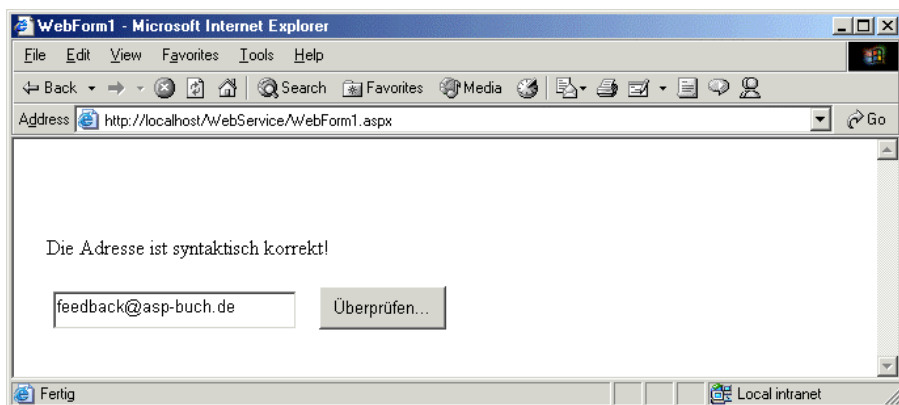


Abbildung 14.6 Der Web Service meint, die Adresse sei korrekt.

Selbstverständlich können Sie XML Web Services auch in „regulären“ Windows-Applikationen nutzen. Nachdem Sie ein entsprechendes Projekt in der Entwicklungsumgebung angelegt haben, können Sie ganz analog zu einer Web-Applikation eine Referenz auf den Web Service einfügen.

Entsprechende Controls vorausgesetzt, können Sie den für die Web-Applikation erstellten Quellcode sogar eins zu eins in die Desktop-Version übernehmen. Die Abbildung zeigt die Verwendung des Web Services in einer kleiner Windows-Anwendung.

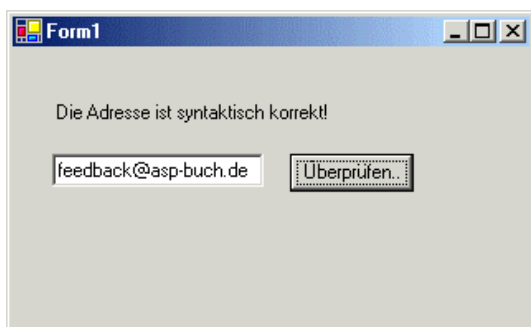


Abbildung 14.7 Web Services machen auch in Windows-Applikationen eine gute Figur.



Sollte Ihnen die Adresse des Web Services nicht (mehr) bekannt oder Sie noch auf der Suche nach einem passenden Dienst sein, so hilft Ihnen vielleicht eines der UDDI-Verzeichnisse weiter. Hier sind viele Anbieter und deren Dienste gelistet. Mehr dazu finden Sie im Rezept „... einen Web Service im UDDI-Verzeichnis finden?“.

14.3 ... einen Web Service ohne Visual Studio .NET konsumieren?

Selbstverständlich können Sie Web Services auch ohne eine Lizenz der Entwicklungsumgebung Visual Studio .NET einsetzen. Das kostenlose Framework enthält alle notwendigen Tools. Die ansonsten von der IDE übernommene Arbeit muss allerdings manuell durchgeführt werden, was jedoch beim Gedanken an die gesparten Euro zu verschmerzen ist.

Das .NET Framework stellt mit dem Kommandozeilenprogramm `wsdl.exe` ein Tool zur Verfügung, mit dem sich die benötigte Proxy-Klasse für einen Web Service erstellen lässt. Diese Klasse kann anschließend lokal verwendet werden und leitet die Aufrufe intern an den Dienst weiter.

Das Programm erwartet zumindest die Übergabe einer URL, unter der der Web Service zu finden ist. Optional kann über den Parameter `/l` eine (Programmier-)Sprache angegeben werden, in der die Proxy-Klasse erstellt wird. Ein möglicher Aufruf des Tools sieht beispielsweise so aus:

```
wsdl /l:cs http://localhost/asp.net/checkmail1.asmx
```

Das Ergebnis ist eine neue C# Quellcode-Datei mit der lokalen Proxy-Klasse für den Web Service. Selbstverständlich wäre auch eine beliebige andere .NET-Sprache wie Visual Basic .NET möglich. Das Listing zeigt die anhand des Beispiels aus dem Rezept „... einen Web Service erstellen?“ automatisch erzeugte Datei.

Listing 14.3 CheckMail.cs

```
//-----
// <autogenerated>
//   This code was generated by a tool.
//   Runtime Version: 1.0.3705.209
//
//   Changes to this file may cause incorrect behavior and will be
//   lost if the code is regenerated.
// </autogenerated>
//-----
// This source code was auto-generated by wsdl, Version=1.0.3705.209.
```

```
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;

/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="CheckMailSoap",
    Namespace="http://www.asp-buch.de")]
public class CheckMail :
    System.Web.Services.Protocols.SoapHttpClientProtocol {

    /// <remarks/>
    public CheckMail() {
        this.Url = "http://localhost/asp.net/checkmail1.asmx";
    }

    /// <remarks/>
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute
        ("http://www.asp-buch.de/CheckRegex",
        RequestNamespace="http://www.asp-buch.de",
        ResponseNamespace="http://www.asp-buch.de",
        Use=System.Web.Services.Description.SoapBindingUse.Literal,
        ParameterStyle=
            System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public bool CheckRegex(string email) {
        object[] results = this.Invoke("CheckRegex", new object[] {
            email});
        return ((bool)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult BeginCheckRegex(string email,
        System.AsyncCallback callback, object asyncState) {
        return this.BeginInvoke("CheckRegex", new object[] {
            email}, callback, asyncState);
    }

    /// <remarks/>
    public bool EndCheckRegex(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((bool)(results[0]));
    }
}
```

Die automatisch erstellte Klasse sieht dank der Kommentare und diverser Attribute ein wenig wüst aus. Der Schein trügt. Die Klasse leitet sich von der Basis `SoapHttpClientProtocol` ab. Diese wurde um drei Methoden ergänzt. Es handelt sich um die lokale Variante von `CheckRegex` sowie zwei analoge Methoden zum asynchronen Aufruf.

Um den Dienst nun innerhalb einer ASP.NET-Seite nutzen zu können, können Sie die Quellcode-Datei entweder kompilieren und im `bin`-Verzeichnis ablegen oder aber mittels der `@Assembly`-Direktive direkt einbinden. Das Listing zeigt Letzteres. Der Web Service wird hier in Verbindung mit einer `TextBox` verwendet. Ein Klick auf den ebenfalls eingefügten Button genügt, um die eingegebene Email-Adresse syntaktisch durch den Web Service überprüfen zu lassen.

Listing 14.4 CheckMail1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Assembly src="CheckMail.cs" %>

<script runat="server">

private void Button1_Click(object sender, System.EventArgs e)
{
    CheckMail checkmail = new CheckMail();
    if(checkmail.CheckRegex(TextBox1.Text))
    {
        Label1.Text = "Die Adresse ist syntaktisch korrekt!";
    }
    else
    {
        Label1.Text = "Die Adresse ist syntaktisch leider nicht korrekt!";
    }
}

</script>

<form runat="server">

    <p> <ASP:Label id="Label1" runat="server"
        text="Bitte Email-Adresse eingeben:"/> </p>

    <p> <ASP:TextBox id="TextBox1" runat="server"/>
        <ASP:Button runat="server" text="Überprüfen ..."
            onclick="Button1_Click"/>
    </p>

</form>
```

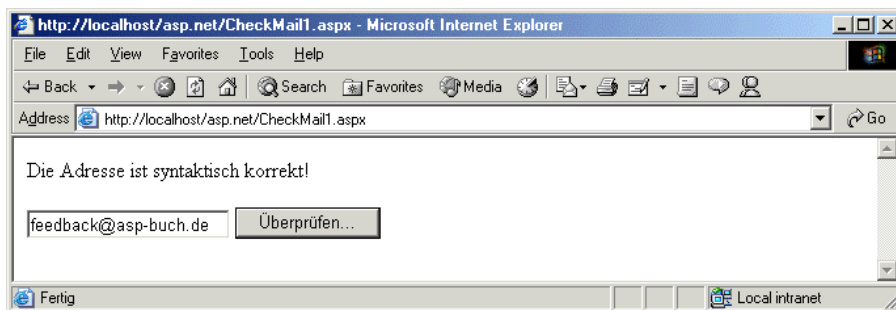


Abbildung 14.8 Der Web Service wird über die erstellte Proxy-Klasse angesprochen.

Die Abbildung zeigt das Ergebnis der Seite im Browserfenster. Ganz offensichtlich ist die Verwendung eines Web Services auch ohne teure Entwicklungsumgebung ziemlich einfach zu handhaben.

14.4 ... einen Web Service ohne Round Trip per JavaScript abfragen?

Web Services lassen sich bequem innerhalb von .NET-Projekten wie Web- und Windows-Applikationen einsetzen. Doch es gibt auch andere Möglichkeiten die Dienste zu nutzen. Eine davon ist das Web Service Behavior von Microsoft. Sie können dies in Verbindung mit dem Internet Explorer innerhalb von einer beliebigen Web-Seite benutzen und den Dienst mit clientseitigem JavaScript aufrufen.

Das Behaviour funktioniert also auf Seite des Clients komplett ohne ASP.NET beziehungsweise ohne .NET Framework. Auch kann komplett auf einen Round Trip verzichtet werden, da der Web Service direkt vom Client aus mittels JavaScript angesprochen wird.

Das Listing zeigt eine HTML-Datei, die das Behaviour verwendet, um über den Dienst aus dem Rezept „... einen Web Service erstellen?“ eine eingegebene Email-Adresse syntaktisch zu überprüfen. Ein Klick auf den Button löst eine clientseitige Ereignisbehandlung aus. Hier wird der Dienst über das Behaviour angesprochen.

Das Behaviour wird im Internet Explorer als Erweiterung eines bestehenden Tags registriert. Sie erkennen dies am div-Tag am Ende des Listings. Über die Cascading Style Sheet-Anweisung wird die URL der externen htc-Datei angegeben. Hier ist der notwendige Quellcode zum Aufrufen von Web Services enthalten. Die angebotenen Methoden werden auf das Tag abgebildet und stehen über dieses Element innerhalb der HTML-Seite zur Verfügung.

Der Service wird mittels der Methode `useService` angebunden. Nun werden neue Aufrufoptionen erstellt und hier die zu übergebenden Parameter sowie der Name der aufrufenden Web-Methode zugewiesen. Diese kann anschließend direkt angesprochen werden. Der Rückgabewert kann nun dazu verwendet werden, eine positive oder negative Meldung auszugeben.

Listing 14.5 CheckMail.htm

```
<html>
<head>

<script language="JavaScript">

function CheckRegex(email)
{
    service.useService
        ("http://localhost/asp.net/checkmail1.asmx?WSDL", "CheckMail");

    callOpt = service.createCallOptions();
    callOpt.async = false;
    callOpt.params = new Array();
    callOpt.params.email = email;
    callOpt.funcName = "CheckRegex";

    result = service.CheckMail.callService(this, callOpt);

    if(result.value)
        alert("Die Adresse ist syntaktisch korrekt!");
    else
        alert("Die Adresse ist syntaktisch leider nicht korrekt!");
}

</script>

</head>
<body>

<form method="post">

    <p> Bitte Email-Adresse eingeben: </p>

    <p> <input type="text" name="email"/>
        <input type="submit" value="Ueberpruefen ..."
            onClick="CheckRegex(document.all.email.value);
            return(false);"/>
    </p>

    <div id="service" style="behavior:url(websevice.htc)"/>
```

```
</form>

</body>
</html>
```

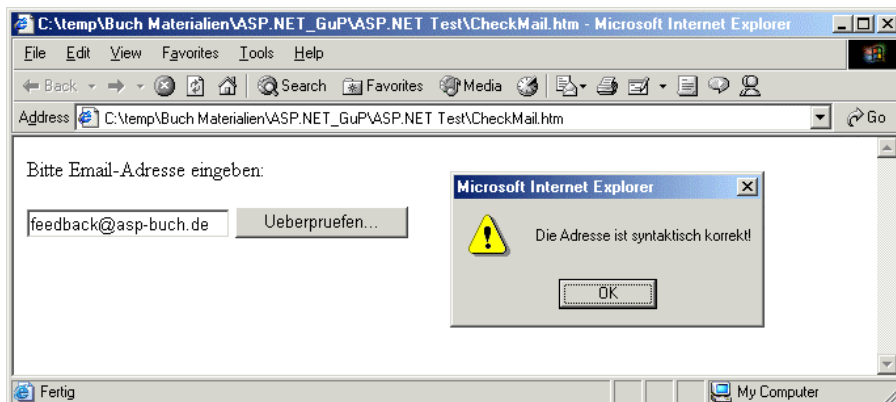


Abbildung 14.9 Der Web Service wird direkt per JavaScript aufgerufen.

14.5 ... einen Web Service mit Code Behind erzeugen?

Was für ASP.NET-Seiten gilt, gilt auch für Web Services. Dort wie hier können Sie Quellcode auf einfache Weise mittels Code Behind auslagern. Erstellen Sie dazu zunächst eine Quellcode-Datei mit der notwendigen Ableitung der Basisklasse `WebService`. Hier werden die öffentlichen Methoden des Dienstes implementiert.

Listing 14.6 CheckMail1.cs

```
using System;
using System.Web.Services;
using System.Text.RegularExpressions;

[WebService(
    Name="CheckMail",
    Description="Ueberpruefung einer Email-Adresse",
    Namespace="http://www.asp-buch.de")
]
public class CheckMail : WebService
{
    [WebMethod(Description="Ueberprüft eine Adresse mittels
        Regular Expressions")]
}
```

```

public bool CheckRegex(string email)
{
    Regex r = new Regex(@"^[\\w\\.\\-]+@[\\w\\.\\-]+\\.([\\w\\.\\-]+\\.)*[\\w\\.\\-]{2,63}\\.[a-zA-Z]{2,4}$");
    return(r.IsMatch(email));
}
}

```

Im zweiten Schritt kompilieren Sie die Datei mittels des sprachabhängigen Kompilers und kopieren die so erzeugte DLL in das bin-Verzeichnis Ihrer Web-Applikation.

```
csc /t:library /r:System.dll,System.Web.dll checkmail1.cs
```

Statt in der aufzurufenden asmx-Datei die Implementierung des Dienstes zu hinterlegen, benötigen Sie hier nur noch die @WebService-Direktive:

```
<% @WebService Class="CheckMail" Language="C#" Debug="true" %>
```

Mit sehr wenigen Schritten können Sie so den Quelltext eines Web Services auslagern. Selbstverständlich können Sie auf diese Weise auch zentrale Web Service-Vorlagen definieren, die Sie je nach Bedarf um die notwendigen Elemente erweitern.

14.6 ... Session-Daten in einem Web Service verwenden?

Web Services sind darauf ausgelegt, verbindungslos und ohne jegliche Zustandsinformationen zu arbeiten. Versucht man daher, auf das Session-Objekt zuzugreifen, erlebt man eine Überraschung, und zwar in Form einer `NullReferenceException`. Das Objekt steht in diesem Kontext nicht zur Verfügung.

Das folgende Beispiel-Listing zeigt das Verhalten auf. Eine Methode `SetValue` soll eine Zeichenkette im Session-Speicher ablegen. Die zweite Web-Methode `GetValue` soll den abgelegten Wert wieder zurückliefern. Führt man eine der beiden Methoden aus, kommt es – wie beschrieben – zu einer Ausnahme.

Listing 14.7 Session1.asmx

```

<% @WebService Class="SetGetValue" Language="C#" Debug="true" %>

using System;
using System.Web.Services;
using System.Text.RegularExpressions;

```

```
public class SetGetValue : WebService
{
    [WebMethod]
    public void SetValue(string value)
    {
        Session["value"] = value;
    }

    [WebMethod]
    public string GetValue()
    {
        return((string) Session["value"]);
    }
}
```

Augenscheinlich ist die Verwendung des Session-Objekts innerhalb von Web Services nicht möglich. Oder etwa doch? Ja, Sie müssen dies nur explizit für jede öffentliche Methode aktivieren. Hierzu stellt das WebMethod-Attribut eine Eigenschaft `EnableSession` zur Verfügung, die – standardmäßig `false` – zunächst eingeschaltet werden muss. Das Listing zeigt das entsprechend abgeänderte Beispiel. Nun werden die Daten korrekt auf dem Server zwischengespeichert.

Listing 14.8 Session2.asmx

```
<% @WebService Class="SetGetValue" Language="C#" Debug=true %>

using System;
using System.Web.Services;
using System.Text.RegularExpressions;

public class SetGetValue : WebService
{
    [WebMethod(EnableSession=true)]
    public void SetValue(string value)
    {
        Session["value"] = value;
    }

    [WebMethod(EnableSession=true)]
    public string GetValue()
    {
        return((string) Session["value"]);
    }
}
```


Das folgende Listing zeigt eine ASP.NET-Seite, die den vorherigen Dienst nutzt. Zunächst wird ein neuer Wert zugewiesen, und anschließend wird dieser wieder abgefragt. In der Zwischenzeit wurde der Textinhalt in einer Session-Variablen auf dem Server gespeichert.

Listing 14.9 Session2.aspx

```
<% @Assembly Src="SetGetValue.cs" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    SetGetValue sgv = new SetGetValue();

    string value1 = "Hallo Welt";
    Response.Write(value1 + " ---> SetValue<br>");
    sgv.SetValue(value1);

    string value2 = sgv.GetValue();
    Response.Write("GetValue ---> " + value2 + "<br>");
}

</script>
```

14.7 ... Sessions ohne Cookie verwenden?

Ganz wie die „regulären“ Sessions bei ASP.NET-Seiten funktioniert das System auch bei Web Services. Zur Identifikation eines Benutzers wird beim ersten Aufruf ein Cookie mit einer eindeutigen Session-ID vom Server an den Client gesendet. Bei weiteren Anfragen wird diese ID vom Client zurück an den Server übertragen und ermöglicht somit eine Identifikation.

ASP.NET bietet die alternative Möglichkeit, Sessions ohne Cookies zu realisieren. Dabei wird beim ersten Aufruf einer Seite ein Redirect auf ein virtuelles Verzeichnis durchgeführt, über das die ID gespeichert wird.

Doch ist die cookie-lose Speicherung von Session-Informationen auch bei Web Services möglich? Ein kleiner Test beantwortet die Frage. Hierfür wird zunächst eine entsprechend konfigurierte web.config-Datei benötigt.

Listing 14.10 web.config

```
<configuration>
  <system.web>
    <sessionState cookieless="true"/>
  </system.web>
</configuration>
```

Versuchen Sie nun, das Beispiel aus dem vorherigen Rezept „... Session-Daten in einem Web Service verwenden?“ auszuführen. Die Abbildung zeigt das ernüchternde Ergebnis. Der aufrufende Client folgt dem Redirect des Servers nicht und kann die zurückgelieferten Daten daher nicht auswerten. Die Verwendung des Services schlägt fehl.

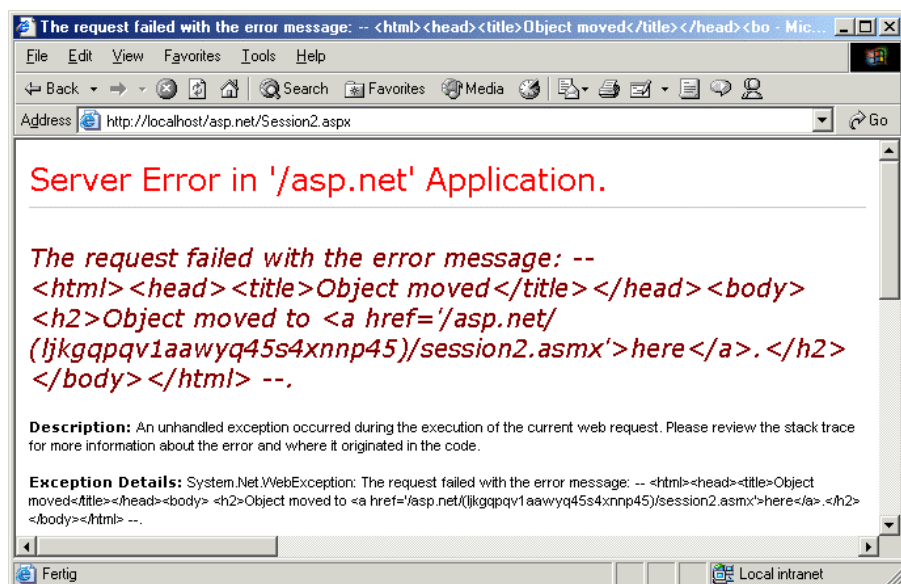


Abbildung 14.10 Die cookie-lose Session verträgt sich nicht mit dem Client.

Die lokale Proxy-Klasse leitet sich von der Basis `SoapHttpClientProtocol` ab. Diese bietet eine Eigenschaft `AllowAutoRedirect` an, die standardmäßig deaktiviert ist. Setzt man die Option auf `true`, so folgt der Client nunmehr den vom Server zurückgemeldeten Redirect-Anweisungen (HTTP-Statuscode 401). Man könnte annehmen, dies wäre die Lösung für das Problem der cookie-losen Session, doch dies ist mitnichten der Fall. Leider verarbeitet der Client auch anschließend die des virtuellen Ordners mit der Session-ID nicht korrekt.

Mangels korrekter Unterstützung bleibt Ihnen nichts anderes übrig, als Cookies zu verwenden, falls Sie den Session-Scope innerhalb eines Web Services nutzen möchten.

14.8 ... einen Counter als Web Service realisieren?

Die Entwicklung eines Zählers als Web Service ist sehr einfach. Sie benötigen lediglich eine öffentliche Web-Methode, die den Zähler inkrementiert und den neuen Wert zurückliefert. Jede Abfrage des aktuellen Zählstandes impliziert damit die Inkrementierung und wird somit gezählt.

Das folgende Listing zeigt die Implementierung eines einfachen Counters als Web Service. Die Methode `GetCount` liefert einen `int`-Wert mit dem aktuellen Stand. Jeder Aufruf der Methode inkrementiert den mit `Application-Scope` abgelegten Zähler.

Listing 14.11 Counter1.asmx

```
<% @WebService Class="Counter" Language="C#" Debug=true %>

using System;
using System.Web.Services;

public class Counter : WebService
{
    [WebMethod]
    public int GetCount()
    {
        if(Application["counter"] == null)
            Application["counter"] = 1;
        else
            Application["counter"] = ((int) Application["counter"]) + 1;

        return((int) Application["counter"]);
    }
}
```

Der vorgestellte Counter bringt zwei Nachteile. Zunächst einmal wird der enthaltene Wert mit `Application-Scope` gespeichert. Fällt der Server aus oder wird die Applikation aus einem anderen Grund neu gestartet, so geht der Wert unwiderruflich verloren. Abhilfe schafft hier die einfach zu implementierende Anbindung an eine Datenbank.

Die zweite Unschönheit des Dienstes ist die Mehrfachzählung. Wird die aufrufende Seite vom Benutzer im Browser aktualisiert, wird auch der Counter inkrementiert. Auf diese Weise lässt sich die Zählung recht einfach manipulieren. Abhilfe schafft hier die Speicherung der Adressen der letzten Besucher. Das zweite Listing zeigt dies anhand einer Queue. Hier werden die Adressen der letzten Besucher abgespeichert. Nur wenn der aktuelle Besucher nicht in der Liste enthalten ist, wird der Zähler inkrementiert. Ansonsten wird der bestehende Wert zurückgeliefert.

Listing 14.12 Counter2.asmx

```
<% @WebService Class="Counter" Language="C#" Debug=true %>

using System;
using System.Collections;
using System.Web.Services;

public class Counter : WebService
{
    const int IPCount = 10;

    [WebMethod]
    public int GetCount(string ip)
    {
        Queue IPList = (Queue) Application["iplist"];
        if(IPList == null)
        {
            IPList = new Queue();
            Application["iplist"] = IPList;
        }

        if(Application["counter"] == null)
            Application["counter"] = 0;

        if(!IPList.Contains(ip))
        {
            if(IPList.Count == IPCount) IPList.Dequeue();
            IPList.Enqueue(ip);
            Application["counter"] = ((int) Application["counter"]) + 1;
        }
        return((int) Application["counter"]);
    }
}
```



Im Listing können Sie erkennen, dass die Adresse des Besuchers als Parameter der Web-Methode übergeben wird. Vielleicht fragen Sie sich, warum hier nicht einfach die Daten aus dem Request-Objekt verwendet werden können. Dies ist nicht möglich, da hier die Daten des Web Service-Clients, nicht aber die Daten des darauf zugreifenden Benutzers vorgehalten werden.

Was fehlt ist eine ASP.NET-Seite, die den Counter einsetzt, um die Besucherabrufe zu zählen. Nachdem die lokale Proxy-Klasse mittels `wsdl.exe` erstellt wurde, kann der Counter benutzt werden. Das Listing zeigt dies.

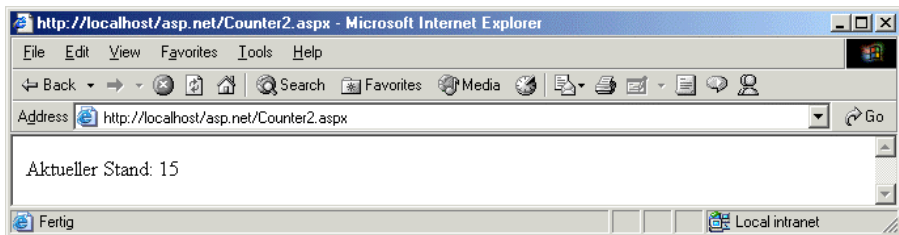
Listing 14.13 Counter2.aspx

```
<% @Assembly Src="Counter.cs" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Counter counter = new Counter();
    int count = counter.GetCount(Request.UserHostAddress);
    Response.Write("Aktueller Stand: " + count.ToString());
}

</script>
```

**Abbildung 14.11** Die Seitenaufrufe werden über den Web Service gezählt.

Die beiden Beispiele demonstrieren die Realisierung eines einfachen Counter Web Services. Es sind zahlreiche Verbesserungen und Erweiterungen denkbar. So können Sie beispielsweise eine Datenbank zugrunde legen, in der die Zugriffe gezählt werden. Über einen zusätzlichen ID-Parameter können Sie mit Hilfe des Web Services unterschiedliche Counter realisieren, die über die ID identifiziert werden.

Sehr sinnvoll ist zudem, den Dienst vor unbefugtem Zugriff zu schützen. Hierzu können Sie den Dienst beispielsweise mit einem Passwort versehen, wie im Rezept „... einen mit Passwort geschützten Web Service erstellen?“ beschrieben. Auch könnten Sie eine Grafik zurückliefern, die in die aufrufende Seite eingebunden wird. Informationen hierzu finden Sie im Rezept „... Bilder von einem Web Service generieren lassen?“.

14.9 ... einen mit Passwort geschützten Web Service erstellen?

Oftmals soll ein Web Service nur einem eingeschränkten Benutzerkreis und nicht gar der Öffentlichkeit zur Verfügung stehen. Wie üblich kann hier die Identifizierung mittels so genannter Credentials, also Benutzername und Passwort, erfolgen. Doch wie sollen diese Daten übergeben werden? Hierzu stehen zwei unterschiedliche Varianten zur Auswahl.

Übergabe der Credentials als Parameter

Sie können die Benutzerdaten als zusätzliche Parameter jeder öffentlichen Web-Methode übergeben. Vor der eigentlichen Ausführung der Methode werden die Daten überprüft. Sind diese nicht korrekt, wird eine `SecurityException` geworfen. Das Listing zeigt dies anhand des Counters aus dem Rezept „... einen Counter als Web Service realisieren?“.

Listing 14.14 Counter3.asmx

```
<% @WebService Class="Counter" Language="C#" Debug=true %>

using System;
using System.Collections;
using System.Security;
using System.Web.Services;

public class Counter : WebService
{
    const int IPCount = 10;

    [WebMethod]
    public int GetCount(string username, string pw, string ip)
    {
        if((username != "paddel") || (pw != "geheim"))
            throw(new SecurityException("Incorrent username or password"));

        Queue IPList = (Queue) Application["iplist"];
        ...
    }
}
```

Das System funktioniert zwar, jedoch müssen die Daten bei jeder Web-Methode übergeben und dort jeweils überprüft werden. Das ist nicht unbedingt die entwicklerfreundlichste Implementierung, insbesondere wenn die Klasse über mehrere Methoden verfügt.

Verwendung von Windows Authentication

Eine Alternative bietet die Verwendung von Windows Authentication. Dabei wird die Identifizierung auf Basis der Windows Benutzer-Accounts den Internet Information Services überlassen.

Das System wird wie bei der Sicherung von regulären ASP.NET-Seiten eingerichtet. Zunächst muss die verwendete Windows Authentication in der Konfigurationsdatei `web.config` aktiviert werden.

Listing 14.15 web.config

```
<configuration>
  <system.web>
    <authentication mode="Windows">
    </authentication>
  </system.web>
</configuration>
```

Im zweiten Schritt muss die Windows Authentication zusätzlich in der IIS-Konfiguration eingeschaltet werden. Dies geschieht über den Dialog „Authentifizierungsmethoden“ in den Eigenschaften der gewünschten Web-Applikation. Die anonyme Anmeldung muss deaktiviert und die Standardauthentifizierung aktiviert werden.

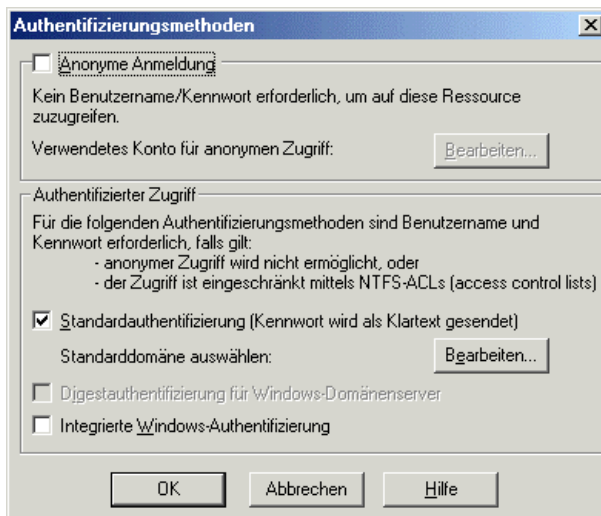


Abbildung 14.12 Der Web Service wird per Windows Authentication geschützt.

Ist die Konfiguration abgeschlossen, kann der Dienst nur noch mit gültigen Benutzerdaten aufgerufen werden. Ein Test im Browser bestätigt dies. Der Dienst selbst enthält dabei keinerlei (zusätzliche) Sicherung, und die Benutzerdaten müssen auch nicht per Parameter übergeben werden. Doch wie werden die Credentials stattdessen programmatisch bei der Verwendung des Web Services übergeben? Das nachfolgende Rezept erklärt's.

14.10 ... auf einen mit Passwort geschützten Web Service zugreifen?

Die gängige Sicherung von Web Services mit Benutzername und Passwort wird per Windows Authentication realisiert. Die Einrichtung erfolgt analog zur Sicherung von regulären Seiten, wie im vorherigen Rezept beschrieben.

Die ersten Schwierigkeiten bei der Nutzung eines so geschützten Dienstes ergeben sich bereits bei der Erstellung der lokalen Proxy-Klasse. Diese wird mit einem Fehler abgebrochen; der Dienst kann nicht abgefragt und die Klasse daher nicht erstellt werden.

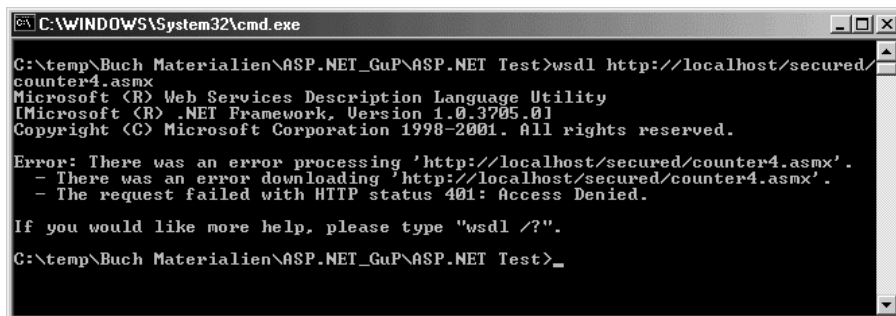


Abbildung 14.13 Das Tool wsdl.exe kann den Web Service nicht abfragen.

Die Ursache für den Fehlschlag ist offensichtlich. Der Dienst ist gesichert, und auch die Erstellung der Proxy-Klasse ist nur in Verbindung mit gültigen Benutzerdaten möglich. Diese können dem Programm über die Parameter `/u` und `/p` mitgeteilt werden. Der Aufruf sieht demnach zum Beispiel wie folgt aus:

```
wsdl /u:testaccount /p:test http://localhost/secured/counter4.asmx
```

Ist die Proxy-Klasse erstellt, können Sie den Web Service verwenden. Da die Daten *nicht* fest in der Klasse abgelegt werden, müssen diese auch beim Aufruf des Dienstes explizit übergeben werden. Hierzu wird eine Klasse `NetworkCredential` aus

dem zuvor importierten Namespace `System.Net` instanziiert. Über entsprechende Eigenschaften oder aber den Konstruktor der Klasse werden Benutzername und Passwort zugewiesen. Anschließend werden die Credentials der gleichnamigen Eigenschaft der Proxy-Klasse übergeben. Nun kann der Dienst aufgerufen werden.

Listing 14.16 Counter4.aspx

```
<% @Assembly Src="Counter.cs" %>
<% @Import Namespace="System.Net" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    NetworkCredential credentials = new NetworkCredential();
    credentials.UserName = "testaccount";
    credentials.Password = "test";

    Counter counter = new Counter();
    counter.Credentials = credentials;
    counter.PreAuthenticate = true;

    int count = counter.GetCount(Request.UserHostAddress);
    Response.Write("Aktueller Stand: " + count.ToString());
}

</script>
```



Sofern Sie einen mit Windows Authentication geschützten Web Service aufrufen, sollten Sie neben der Übergabe der Credentials die Eigenschaft `PreAuthenticate` der von der Basis `SoapHttpClientProtocol` abgeleiteten Proxy-Klasse auf `true` setzen. Dies stellt sicher, dass die Daten direkt übergeben werden und nicht erst nach einer (internen) 401-Fehlermeldung des Servers. Dies spart Zeit und beschleunigt den Aufruf des Dienstes.



Beachten Sie bitte, dass bei der Standard-Windows Authentication die Benutzerdaten wie üblich im Klartext übermittelt werden. Belauscht jemand die Verbindung, so können die Daten ausgelesen und möglicherweise missbraucht werden. Um dies zu verhindern, können Sie den Web Service mittels HTTPS aufrufen. Wie dies geht, erfahren Sie im nachfolgenden Rezept „... einen Web Service per HTTPS/SSL aufrufen?“.

14.11 ... einen Web Service per HTTPS/SSL aufrufen?

Um die beim Aufruf eines Web Services übertragenen Daten wie zum Beispiel die benötigten Credentials zu sichern, können Sie den Dienst über HTTPS aufrufen. Die Daten werden dann per SSL verschlüsselt übertragen. Sie benötigen dazu auf dem Server lediglich ein korrekt installiertes und gültiges SSL-Zertifikat. Um dieses nutzen zu können, geben Sie bei der Erstellung der Proxy-Klasse das Protokoll mit „https“ an.

```
wsdl /u:testaccount /p:test https://localhost/secured/counter4.asmx
```



Beachten Sie bitte, dass die verschlüsselte Kommunikation zwischen dem Web Service und dessen Konsumenten eine größere Bandbreite benötigt als die unverschlüsselte Variante. Der Aufruf eines Dienstes per SSL dauert daher in der Regel länger.

14.12 ... komplexe Datentypen in einem Web Service verwenden?

Neben Zeichenketten, numerischen Werten und anderen einfachen Datentypen können auch komplexe Daten über Web Services transferiert werden.

DataSet

Ein gutes Beispiel hierfür ist die Klasse `DataSet`. Diese kann inklusive aller enthaltenen Daten und Meta-Informationen als XML-Stream abgebildet werden. Sollen umfangreiche Daten übertragen werden, liefert ein Web Service daher oftmals eine `DataSet`-Instanz zurück. Diese kann auf der Client-Seite wie gewohnt verwendet und manipuliert werden. Sollen Änderungen zurückgeschrieben werden, ist natürlich auch eine Rückübertragung an den Server möglich, sofern eine entsprechende Web-Methode angeboten wird.

Im folgenden Listing eines Web Services wird eine neue `DataSet`-Instanz mit Daten aus einer Datenbank gefüllt. Es handelt sich um die Tabelle „Books“ der mitgelieferten Beispieldatenbank. Das `DataSet` wird als Rückgabewert an den Aufrufer zurückgesendet.

Listing 14.17 DataSet1.asmx

```
<% @WebService Class="GetBooksService" Language="C#" Debug=true %>

using System;
using System.Data;
using System.Data.OleDb;
using System.Web.Services;

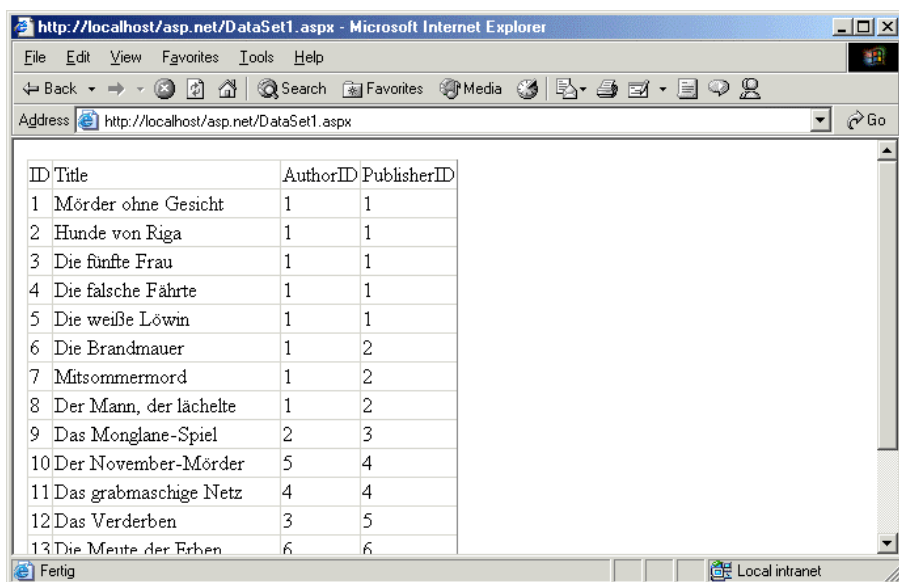
public class GetBooksService : WebService
{
    [WebMethod]
    public DataSet GetBooks()
    {
        OleDbConnection conn = new
            OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\books.mdb");
        conn.Open();

        string SQL = "SELECT * FROM BOOKS";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);

        OleDbDataAdapter adapter = new OleDbDataAdapter();
        adapter.SelectCommand = cmd;

        DataSet dataset = new DataSet();
        adapter.Fill(dataset, "books");
        conn.Close();

        return(dataset);
    }
}
```



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/asp.net/DataSet1.aspx`. The page content is a table with four columns: ID, Title, AuthorID, and PublisherID. The table contains 13 rows of book data. The status bar at the bottom shows 'Fertig' and 'Local intranet'.

ID	Title	AuthorID	PublisherID
1	Mörder ohne Gesicht	1	1
2	Hunde von Riga	1	1
3	Die fünfte Frau	1	1
4	Die falsche Fährte	1	1
5	Die weiße Löwin	1	1
6	Die Brandmauer	1	2
7	Mitsommermord	1	2
8	Der Mann, der lächelte	1	2
9	Das Monglane-Spiel	2	3
10	Der November-Mörder	5	4
11	Das grabmaschige Netz	4	4
12	Das Verderben	3	5
13	Die Meute der Erben	6	6

Abbildung 14.14 Die Daten wurden von einem Web Service geliefert.

Möchten Sie den Web Service nutzen, erstellen Sie wie gewohnt mittels `wsdl.exe` beziehungsweise dem Web-Referenz-Dialog der Entwicklungsumgebung Visual Studio .NET eine Proxy-Klasse. Diese erkennt den Rückgabewert des Dienstes und implementiert diesen lokal ebenfalls als `DataSet`. Sie können daher das Ergebnis der Web-Methode `GetBooks` als Datenquelle für ein `DataGrid`-Control verwenden. Die Abbildung zeigt das Ergebnis des zweiten Listings.

Listing 14.18 `DataSet1.aspx`

```
<% @Assembly Src="GetBooksService.cs" %>
<% @Import Namespace="System.Data" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    GetBooksService service = new GetBooksService();
    DataSet dataset = service.GetBooks();
    dg.DataSource = dataset;
    dg.DataBind();
}

</script>

<ASP:DataGrid id="dg" runat="server"/>
```

Die benutzte Methode in der automatisch per `wsdl.exe` erstellten Proxy-Klasse trägt dafür Sorge, dass die Daten als `DataSet` zurückgeliefert werden.

```
...
    public System.Data.DataSet GetBooks() {
        object[] results = this.Invoke("GetBooks", new object[0]);
        return ((System.Data.DataSet) (results[0]));
    }
...
```

Andere und eigene Klassen

Die Klasse `DataSet` ist äußerst flexibel und kann durch die Aufnahme nahezu beliebiger Daten als universelles Transportmedium zwischen Dienst und Konsument dienen. Für manche Anwendungen ist die Klasse jedoch schlichtweg *zu* universell oder schlichtweg *zu oversized*. Selbstverständlich lassen sich jedoch auch andere und sogar eigene Klassen austauschen.

Rufen Sie sich bitte ins Gedächtnis, woraus ein Web Service besteht. Richtig, es handelt sich um XML-Daten, die mittels SOAP ausgetauscht werden. Folgerichtig werden die Daten einer übertragenen Klasse nicht als binärer Stream, sondern als XML abgebildet und müssen insofern serialisiert werden. Eine Grundvoraussetzung für die Übertragung einer Klasse ist daher die Möglichkeit, diese Klasse zu serialisieren und nach der Übertragung wieder zu deserialisieren.

Das folgende Beispiel zeigt die Übertragung einer individuellen Klasse `Person`. Die Klasse enthält zwei Eigenschaften `Firstname` und `Lastname`. Die initiiellen Werte können im Konstruktor der Klasse übergeben werden. Die Klasse ist in der Quellcode-Datei des Web Services hinterlegt und wird von dessen Web-Methode `GetPersonData` zurückgeliefert.

Listing 14.19 Complex1.asmx

```
<% @WebService Class="ComplexData" Language="C#" Debug=true %>

using System;
using System.Collections;
using System.Security;
using System.Web.Services;

public class ComplexData : WebService
{
    [WebMethod]
    public Person GetPersonData()
    {
```

```
    Person person = new Person("Patrick A.", "Lorenz");
    return(person);
}

[Serializable]
public class Person
{
    private string m_Firstname;
    private string m_Lastname;

    public Person() {}

    public Person(string firstname, string lastname)
    {
        Firstname = firstname;
        Lastname = lastname;
    }

    public string Firstname
    {
        get { return(m_Firstname); }
        set { m_Firstname = value; }
    }

    public string Lastname
    {
        get { return(m_Lastname); }
        set { m_Lastname = value; }
    }
}
```



Die vom Web Service zurückgelieferte Klasse `Person` ist mit dem Attribut `Serializable` versehen, das die Klasse als serialisierbar kennzeichnet. Eine Voraussetzung hierfür ist die Implementierung eines parameterlosen Standardkonstruktors. Diese wird für eine eventuell notwendige Deserialisierung verwendet. Es reicht aus, den Konstruktor wie im Listing ohne jeglichen Inhalt zu definieren.

Auch für einen Web Service, der individuelle Klassen zurückliefert, lässt sich die benötigte Proxy-Klasse mit Hilfe des Kommandozeilenprogramms `wSDL.exe` oder des Web-Referenz-Dialogs der Entwicklungsumgebung Visual Studio .NET automatisch erstellen. Anschließend lässt sich der Dienst wie gewohnt verwenden.

Listing 14.20 Complex1.aspx

```

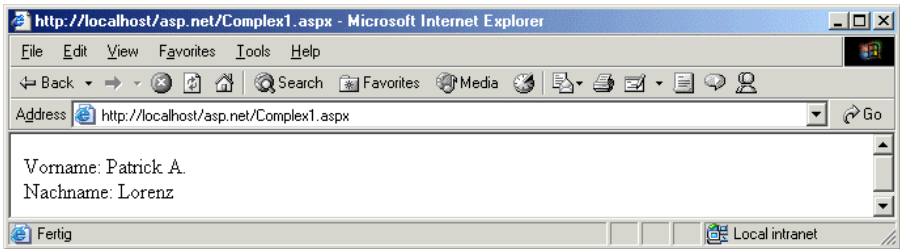
<% @Assembly Src="ComplexData.cs" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    ComplexData service = new ComplexData();
    Person person = service.GetPersonData();
    Response.Write("Vorname: " + person.Firstname + "<br>");
    Response.Write("Nachname: " + person.Lastname + "<br>");
}

</script>

```

**Abbildung 14.15** Die Daten wurden über eine eigene Klasse geliefert.

Das Listing zeigt, dass die innerhalb des Web Services auf dem Server hinterlegte Klasse `Person` nach der Einbindung der Proxy-Klasse auch auf dem Client zur Verfügung steht. Auch diese Klasse wurde also automatisch von `wsdl.exe` nachgebildet. Schaut man sich die erstellte Quellcode-Datei allerdings etwas genauer an, so fallen deutliche Unterschiede zwischen der originalen Version der Klasse und der lokalen Nachbildung auf.

Listing 14.21 ComplexData.cs

```

/// <remarks/>
[System.Xml.Serialization.XmlTypeAttribute(
    Namespace="http://tempuri.org/")]
public class Person {

    /// <remarks/>
    public string Firstname;

    /// <remarks/>
    public string Lastname;
}

```

Die ursprünglich als Eigenschaften implementierten Mitglieder `Firstname` und `Lastname` sind in der lokalen Version als öffentliche Felder implementiert. Dies scheint zunächst keinen nennenswerten Unterschied zu machen, ergibt jedoch Schwierigkeiten bei der Verwendung mit der `DataBinding`-Syntax von ASP.NET. Die hier oft genutzte statische Methode `DataBinder.Eval` kann per Reflection ausschließlich auf Eigenschaften zugreifen. Im Falle der automatisch erstellten lokalen Kopien klappt dies nicht, da die Eigenschaften als Felder implementiert werden.

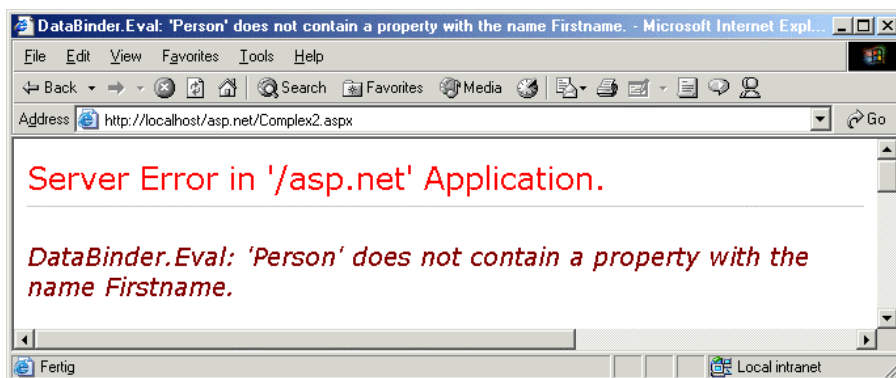


Abbildung 14.16 Die Klasse `Person` enthält keine Eigenschaft `Firstname`.

Listing 14.22 `Complex2.aspx`

```
<% @Assembly Src="ComplexData.cs" %>

<script runat="server">

Person person;

void Page_Load(object sender, EventArgs e)
{
    ComplexData service = new ComplexData();
    person = service.GetPersonData();
    DataBind();
}

</script>

Vorname: <%=# DataBinder.Eval(person, "Firstname") %> <br/>
Nachname: <%=# DataBinder.Eval(person, "Lastname") %> <br/>
```


Um dennoch die Vorteile einer Eval-Methode nutzen zu können, können Sie die bestehende einfach ergänzen. Da die Klasse `DataBinder` als `sealed` markiert ist und daher nicht abgeleitet werden kann, empfehle ich die Implementierung einer Eval-Methode innerhalb der aktuellen Page-Klasse. Zur Evaluierung wird die Methode `GetField` der Klasse `Type` verwendet.

Listing 14.23 Complex3.aspx

```
<% @Assembly Src="ComplexData.cs" %>
<% @Import Namespace="System.Reflection" %>

<script runat="server">

Person person;

void Page_Load(object sender, EventArgs e)
{
    ComplexData service = new ComplexData();
    person = service.GetPersonData();
    DataBind();
}

object Eval(object container, string expression)
{
    try
    {
        return(DataBinder.Eval(container, expression));
    }
    catch
    {
        Type t = container.GetType();
        FieldInfo field = t.GetField(expression);
        return(field.GetValue(container));
    }
}

</script>

Vorname: <%= Eval(person, "Firstname") %> <br/>
Nachname: <%= Eval(person, "Lastname") %> <br/>
```

Durch die Quasi-Erweiterung der Methode `Eval` können nun auch öffentliche Felder ausgewertet werden. Die Abbildung zeigt das bekannte Ergebnis im Browserfenster.

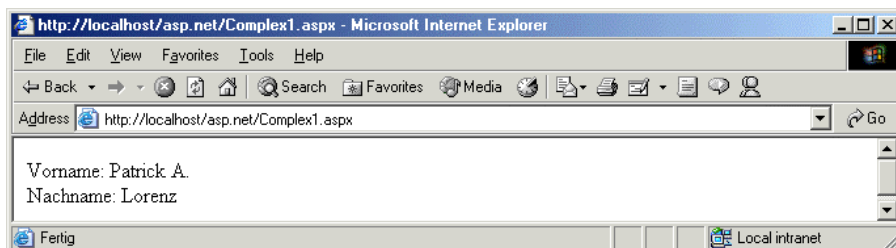


Abbildung 14.17 Dank Reflections werden die Daten aus den Feldern ausgelesen.

14.13 ... binäre Daten mit einem Web Service übertragen?

Die Übertragung von Daten zwischen Web Service und dessen Konsument ist nicht nur auf Formate beschränkt, die direkt in XML-Form abgebildet werden können. Auch binäre Daten können innerhalb von XML-Streams transferiert werden. So können Sie beispielsweise ganze Dateien zwischen den beiden Gegenstellen austauschen. Die Daten werden im Web Service als byte-Array definiert und vor der Übertragung in eine Base64-kodierte Zeichenkette umgewandelt.

Das Beispiel-Listing zeigt einen Web Service mit einer einzigen öffentlichen Web-Methode `GetPDF`. Diese liefert ein byte-Array mit dem Inhalt einer zuvor mit Hilfe der Klasse `FileStream` ausgelesenen PDF-Datei zurück.

Listing 14.24 Binary1.asmx

```
<% @WebService Class="BinaryData" Language="C#" Debug=true %>

using System;
using System.IO;
using System.Web.Services;

public class BinaryData : WebService
{
    [WebMethod]
    public byte[] GetPDF()
    {
        string filename = Server.MapPath("test.pdf");
        FileStream stream = new FileStream(filename,
            FileMode.Open, FileAccess.Read);
        byte[] bytes = new byte[stream.Length];
        stream.Read(bytes, 0, bytes.Length);
        return(bytes);
    }
}
```

Die Datei `test.pdf` im Verzeichnis des Web Services wird – und das ist bei konkurrierenden Zugriffen wichtig – lesend und somit nicht exklusiv geöffnet. Folgerichtig können mehrere Aufrufe parallel durchgeführt werden.

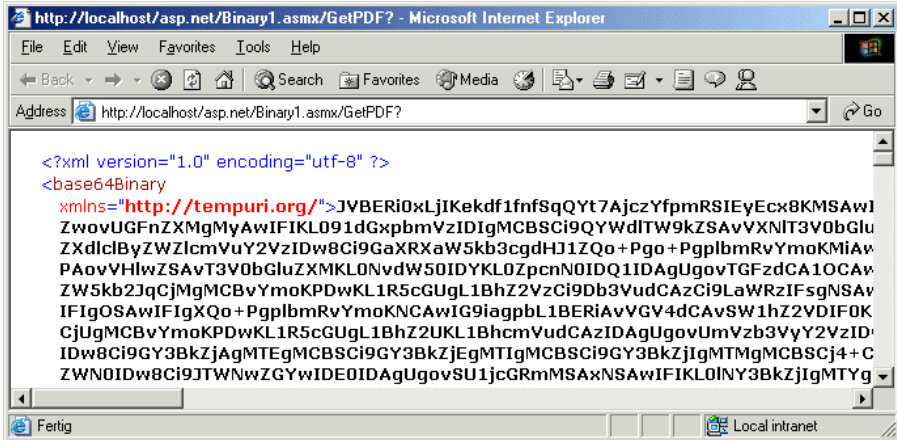


Abbildung 14.18 Die Base64-kodierten Daten der PDF-Datei

Die Abbildung zeigt einen Teil der vom Web Service zurückgelieferten, Base64-kodierten Daten. Ob die Daten korrekt übertragen wurden, lässt sich jedoch nicht erkennen. Hierzu bedarf es eines kleinen Beispiels, das den Web Service konsumiert. Nachfolgend sehen Sie ein solches. Die Daten werden als byte-Array abgefragt und an den Client übergeben. Der richtige Content-Type sorgt dafür, dass das Adobe PDF-PlugIn die übertragene PDF-Datei direkt im Browser anzeigt.

Listing 14.25 Binary1.aspx

```
<% @Assembly Src="BinaryData.cs" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    BinaryData service = new BinaryData();
    byte[] bytes = service.GetPDF();
    Response.ContentType = "application/pdf";
    Response.BinaryWrite(bytes);
}

</script>
```

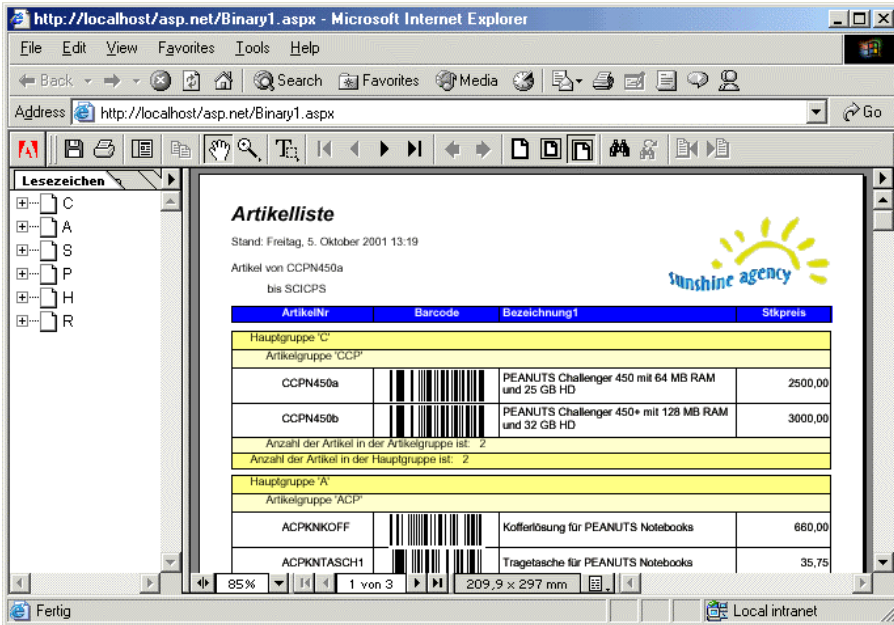


Abbildung 14.19 Der Inhalt der PDF-Datei wurde vom Web Service geliefert.



Bedenken Sie beim Austausch von binären Daten immer das übertragene Volumen. Die im Beispiel verwendete PDF-Datei ist knapp 600 KB groß. Diese Menge muss zweimal ausgetauscht werden, einmal zwischen Web Service und Konsument (Web Server) und dann noch einmal ausgehend vom Web Server bis zum Client-Browser. Zudem wird der Inhalt der Datei auf beiden Stationen in Form eines byte-Arrays im Speicher gehalten. Gerade bei mehreren parallelen Zugriffen kann dies den Arbeitsspeicher des jeweiligen Servers ganz schön unter Druck setzen.

14.14 ... Bilder von einem Web Service generieren lassen?

Web Services können zur dynamischen Generierung von Bildern verwendet werden. Statt als Datentyp Bitmap oder Image anzugeben, empfiehlt es sich, auch hier analog zum Rezept „... binäre Daten mit einem Web Service übertragen?“ ein byte-Array zu verwenden. Im Gegensatz zu dem proprietären Format bietet das Array Plattformunabhängigkeit und entspricht so ganz dem Grundgedanken von Web Services.

Das folgende Beispiel basiert auf dem in „... einen Counter als Web Service realisieren?“ vorgestellten Counter Web Service. Statt eines numerischen Werts liefert diese Implementierung jedoch ein byte-Array mit einer dynamisch erstellten GIF-Datei zurück. Der aktuelle Stand des Counters wurde mit Hilfe der Klassen `Bitmap` und `Graphics` visualisiert.

Listing 14.26 `GraphicCounter1.asmx`

```
<% @WebService Class="GraphicCounter" Language="C#" Debug=true %>

using System;
using System.Collections;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Web.Services;

public class GraphicCounter : WebService
{
    const int IPCount = 10;

    [WebMethod]
    public byte[] GetCount(string ip)
    {
        Queue IPList = (Queue) Application["iplist"];
        if(IPList == null)
        {
            IPList = new Queue();
            Application["iplist"] = IPList;
        }

        if(Application["counter"] == null)
            Application["counter"] = 0;

        if(!IPList.Contains(ip))
        {
            if(IPList.Count == IPCount) IPList.Dequeue();
            IPList.Enqueue(ip);
            Application["counter"] = ((int) Application["counter"]) + 1;
        }

        int count = (int) Application["counter"];

        Bitmap b = new Bitmap(150, 45);
        Graphics g = Graphics.FromImage(b);
        g.DrawString(count.ToString("0000000"), new Font("Comic Sans MS",
            25), new SolidBrush(Color.White), 0, 0);

        MemoryStream stream = new MemoryStream();
```

```
b.Save(stream, ImageFormat.Gif);
stream.Seek(0, SeekOrigin.Begin);
byte[] bytes = new byte[stream.Length];
stream.Read(bytes, 0, bytes.Length);

return(bytes);
}
}
```

Zur Umwandlung des Bildes in ein byte-Array wird die Grafik in einer neu instanziierten MemoryStream-Klasse abgespeichert. Da direkt anschließend ein Lesezugriff stattfinden soll, muss der Zeiger des Streams mittels `Seek` an den Beginn gesetzt werden. Anschließend sorgt die Methode `Read` für die Übertragung in das bereitgestellte Array. Dieses kann als Rückgabewert an den Konsumenten des Dienstes transferiert werden.

Das zweite Listing zeigt einen solchen Konsumenten. Das byte-Array wird hier ausgelesen und an den Client-Browser übertragen. Zuvor wird wie gewohnt der richtige Content-Type eingestellt.

Listing 14.27 GraphicCounter1.aspx

```
<% @Assembly Src="GraphicCounter.cs" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    GraphicCounter service = new GraphicCounter();
    byte[] bytes = service.GetCount(Request.UserHostAddress);
    Response.ContentType = "image/gif";
    Response.BinaryWrite(bytes);
}

</script>
```

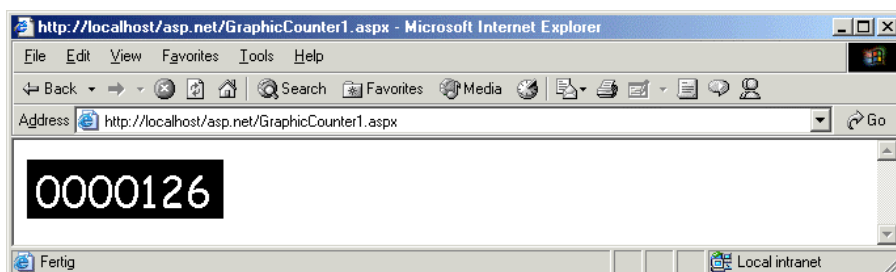


Abbildung 14.20 Die Grafik wurde von einem Web Service erstellt.

Die vorgestellte ASP.NET-Seite liefert ausschließlich das Bild zurück. Andere Elemente einer „regulären“ Seite können nicht aufgenommen werden, da pro Verbindung entweder HTML oder aber die Grafik übertragen werden kann. Was aber, wenn eine Seite beides enthalten soll?

Die Problematik hierbei ist weniger die Darstellung von HTML-Inhalt und Grafikobjekten auf einer Seite. Dies ist durch das Protokoll fest geregelt und wird zwangsläufig auf zwei Seiten hinauslaufen. Die eine gibt den HTML-Inhalt zurück und referenziert über ein `img`-Tag die zweite Seite, die das Bild liefert. Angenommen, der Web Service wird in der „Hauptseite“ angesprochen, wie wird das zurückerhaltene Bild an die zweite Seite übergeben? Eine gute Frage, die das folgende Listing beantwortet.

Listing 14.28 GraphicText1.aspx

```
<% @Assembly Src="GraphicText.cs" %>
<% @Import Namespace="System.Drawing" %>
<% @Import Namespace="System.IO" %>

<script runat="server">

void creategraphic(object sender, EventArgs e)
{
    GraphicText service = new GraphicText();
    byte[] bytes = service.DrawText(text.Text);

    Bitmap bitmap = new Bitmap(new MemoryStream(bytes));
    Session["bitmap"] = bitmap;
    image.Visible = true;
}

</script>

<form runat="server">

    <p> Bitte geben Sie einen Text ein: </p>

    <p>
        <ASP:TextBox id="text" runat="server" text="Hallo!"/>
        <ASP:Button runat="server" text="Grafik generieren"
            onclick="creategraphic"/>
    </p>

    <ASP:Image id="image" runat="server"
        imageurl="ShowImageFromSession.aspx" visible="false"/>

</form>
```

Die Seite spricht einen Web Service `GraphicText` an. Der Web-Methode `DrawText` wird eine vom Benutzer in eine `TextBox` eingegebene Zeichenkette übergeben. Auf dem Server wird der Text nun in eine Grafik umgewandelt. Zurückgeliefert wird das bereits bekannte `byte-Array`. Dieses wird jedoch *nicht* an den Client übertragen. Dies würde keinen Sinn haben, denn die Seite enthält ja bereits HTML-Elemente wie die `TextBox` und einen Button. Aus dem Array wird daher mittels einer `MemoryStream`-Instanz wieder ein `Bitmap` gewonnen, das anschließend in einer Session-Variablen abgelegt wird.

Die Darstellung des Bildes wird über ein `img`-Tag in Form eines Image-Controls vorgenommen. Dieses referenziert die Seite `ShowImageFromSession.aspx`. Im Quelltext der Seite wird das `Bitmap`-Objekt aus der Session-Variablen ausgelesen und in gewohnter Form an den Browser übergeben. Auf diese Weise ist eine parallele Darstellung von HTML und Grafik möglich.

Listing 14.29 `ShowImageFromSession.aspx`

```
<% @Import Namespace="System.Drawing" %>
<% @Import Namespace="System.Drawing.Imaging" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Bitmap bitmap = (Bitmap) Session["bitmap"];
    Response.ContentType = "image/jpeg";
    bitmap.Save(Response.OutputStream, ImageFormat.Jpeg);
}

</script>
```

Natürlich möchte ich Ihnen zum Schluss nicht mehr den dahinter liegenden Web Service vorstellen. Dieser ist dem vorherigen jedoch sehr ähnlich. Lediglich der darzustellende Text wird nun vom Benutzer bestimmt. Wenn Sie genau hinschauen, werden Sie eine interessante Entdeckung machen. Seitens des Web Services wird eine GIF-Datei an den Konsumenten des Dienstes gesendet. Dieser erzeugt daraus eine `Bitmap`-Instanz, die quasi geschlechtslos ist, gemeint ist ohne bestimmte Formatangabe. Es ist daher problemlos möglich, dass die Grafik von der zweiten Seite als JPEG an den Browser übertragen wird.

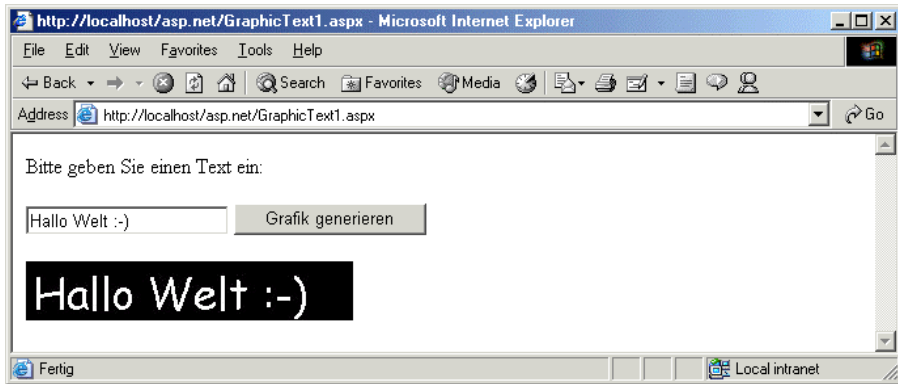


Abbildung 14.21 Der eingegebene Text wird als Grafik dargestellt.

Listing 14.30 GraphicText1.asmx

```
<% @WebService Class="GraphicText" Language="C#" Debug=true %>

using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Web.Services;

public class GraphicText : WebService
{
    [WebMethod]
    public byte[] DrawText(string text)
    {
        Bitmap b = new Bitmap(250, 45);
        Graphics g = Graphics.FromImage(b);
        g.DrawString(text, new Font("Comic Sans MS", 25),
            new SolidBrush(Color.White), 0, 0);

        MemoryStream stream = new MemoryStream();
        b.Save(stream, ImageFormat.Gif);
        stream.Seek(0, SeekOrigin.Begin);
        byte[] bytes = new byte[stream.Length];
        stream.Read(bytes, 0, bytes.Length);

        return (bytes);
    }
}
```



Das vorgestellte System hat immer dann Sinn, wenn der Web Service in einer regulären Datei aufgerufen werden muss und dessen Grafikergebnis unabhängig davon an den Client gesendet werden soll. Ein prominenter Vertreter für dieses Modell sind die Microsoft MapPoint .NET Web Services. Hier wird individuell angefordertes Kartenmaterial im Browser dargestellt. Bedenken Sie bei eigenen Anwendungen immer die daraus resultierende Notwendigkeit von Sessions und die Speicherung des Bildes im Arbeitsspeicher des Servers.

14.15 ... die automatische asmx-Hilfeseite verändern?

Ruft man einen Web Service mit der Dateiendung `asmx` im Browser auf, so wird eine automatisch erzeugte Hilfe-Datei angezeigt. Hier werden die hinterlegten Hilfetexte angezeigt und alle verfügbaren Web-Methoden aufgelistet. Sofern diese Standarddatentypen verwenden, können Sie die Dienste zudem an Ort und Stelle ausprobieren.

Die Grundlage für die Hilfedatei ist eine reguläre, wenngleich durchaus komplexe ASP.NET-Seite. Die Datei liegt unter dem `DefaultWsdHelpGenerator.aspx` im folgenden Verzeichnis:

```
<WINDIR>\Microsoft.NET\Framework\<Version>\Config\
```

Falls gewünscht können Sie die Datei nach Ihren eigenen Vorstellungen anpassen. Sie sollten dazu jedoch eine Kopie der Datei unter anderem Namen erstellen. Anschließend können Sie diesen Namen in der globalen Konfigurationsdatei `machine.config` hinterlegen. Das Listing zeigt den entsprechenden Abschnitt.

Listing 14.31 machine.config

```
<configuration>
  <webServices>
    <wsdlHelpGenerator
      href="PALWsdHelpGenerator.aspx" />
    </webServices>
  </configuration>
```

Ich habe die hier angegebene Datei auf Basis der Standardseite erstellt und nur minimal angepasst. Auf jeder Seite soll eine individuelle Überschrift sowie am Ende ein Copyright-Hinweis angezeigt werden. Diese Änderungen sind durchaus praxisnah; wenn Sie der Öffentlichkeit eigene anbieten, werden Sie diese Punkte am ehesten ändern wollen. Die Abbildung zeigt das Ergebnis der Änderung. Deren Grundlage finden Sie auf der begleitenden Buch-CD-ROM.

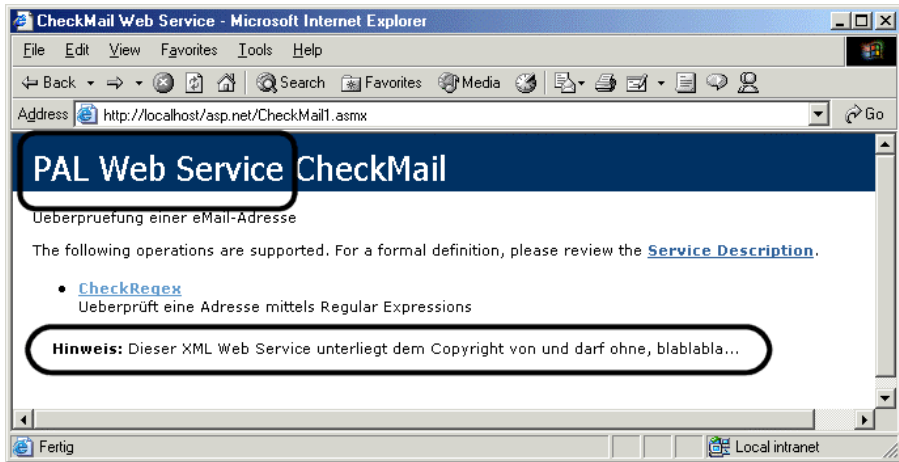


Abbildung 14.22 Die Hilfedatei wurde individuell angepasst.

14.16 ... einen Web Service im UDDI-Verzeichnis finden?

Das Universal Description Discovery Integration-Verzeichnis (kurz UDDI) stellt eine zentrale Anlaufstelle zum Suchen von Web Services zur Verfügung. Jeder, der einen Dienst der Öffentlichkeit zur Verfügung stellen möchte, sei es kostenlos oder gegen Gebühr, ist angehalten, diesen Dienst im UDDI-Verzeichnis zu registrieren.

Sind Sie auf der Suche nach einem passenden Dienst, beispielsweise um Kreditkartenbuchungen durchführen zu können oder aktuelle Wetterinformationen zu beziehen, so können Sie Anbieter über das UDDI-Verzeichnis finden. Sie erreichen dieses über die folgenden zwei Adressen:

www.uddi.org
uddi.microsoft.com

Sie haben hier die Möglichkeit, direkt nach Schlüsselwörtern zu suchen oder sich durch die hierarchisch angeordneten Ebenen des Verzeichnisses zu klicken. Zu allen Services werden kleine Beschreibungen sowie Kontaktadressen angegeben. In vielen Fällen wird auch direkt auf die Discovery-URL verwiesen, so dass Sie eine Proxy-Klasse erstellen und den Dienst so direkt einsetzen können.

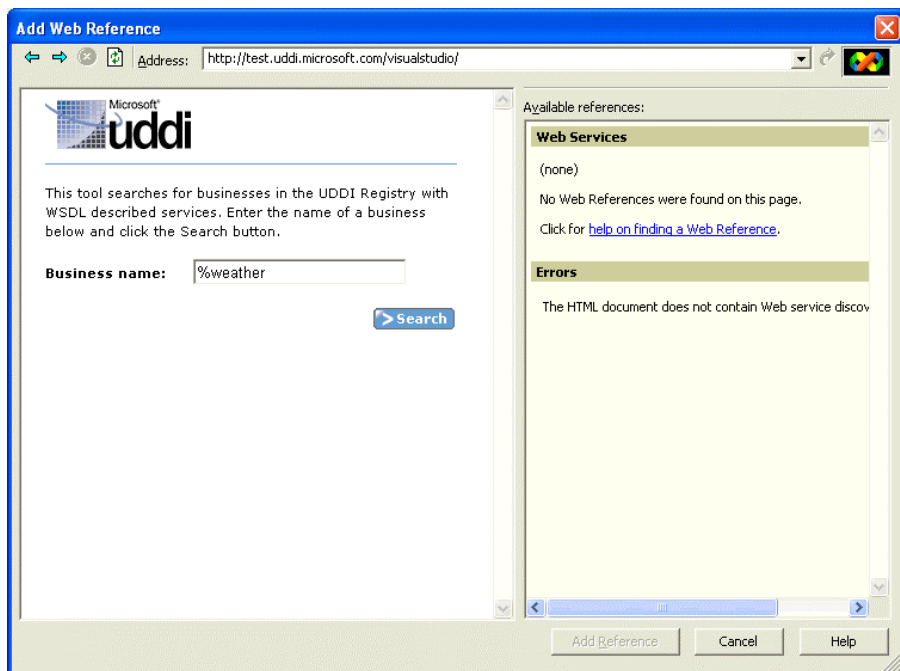


Abbildung 14.23 Im UDDI-Verzeichnis können Sie nach Web Services suchen.

Die Entwicklungsumgebung Visual Studio .NET löst die Integration besonders komfortabel. Öffnen Sie hier den Web-Referenz-Dialog, so wird Ihnen das UDDI-Verzeichnis direkt angeboten. Sie können hier nach dem gewünschten Begriff suchen. Es handelt sich leider um eine Anfangssuche, die Sie jedoch durch Angabe eines Prozentzeichens überlisten können; intern scheint der SQL-Befehl `LIKE` zum Einsatz zu kommen. Suchen Sie beispielsweise nach einem Wetterdienst, geben Sie wie in der Abbildung gezeigt „%weather“ ein.

Ein Buttonklick sucht im Verzeichnis nach passenden Diensten. Wie in der zweiten Abbildung zu sehen, wird tatsächlich ein Web Service gefunden, der das aktuelle Wetter für die USA liefert. Ein weiterer Klick öffnet die WSDL-Beschreibung des Dienstes. Nun wird der Button „Add Reference“ aktiv, und Sie können den Web Service mit einem einfachen Klick in Ihre Applikation einbinden. Die benötigte Proxy-Klasse wird automatisch erzeugt.

Anschließend können Sie den Dienst direkt benutzen. Im Falle der hier gezeigten Wetterinformationen können Sie beispielsweise die aktuelle Temperatur anhand einer US-Postleitzahl erfragen. Der gezeigte Wert „90012“ entspricht der Innenstadt von Los Angeles in Kalifornien. Fans von überflüssigen Soaps dürfen das Beispiel selbstverständlich modifizieren und stattdessen „90210“ eingeben, was die Postleitzahl eines bekannten Vororts von L.A. darstellt.

```

WeatherRetriever weather = new WeatherRetriever();
CurrentWeather curweather = weather.GetWeather("90012");
Response.Write("Temperatur in LA: " +
    curweather.CurrentTemp.ToString() + "F");

```

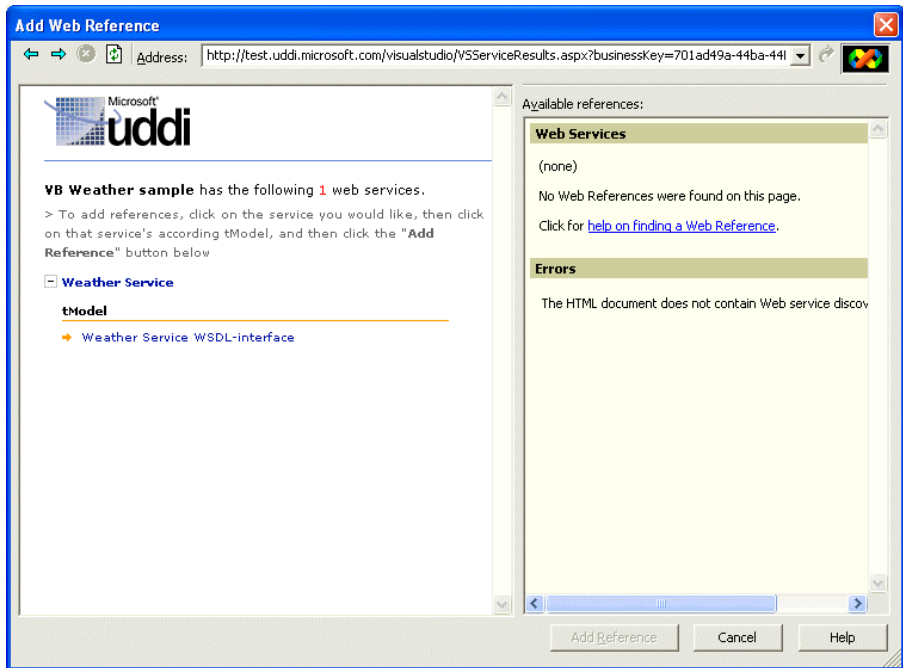


Abbildung 14.24 Der Dienst liefert Wetterinformationen für die USA.

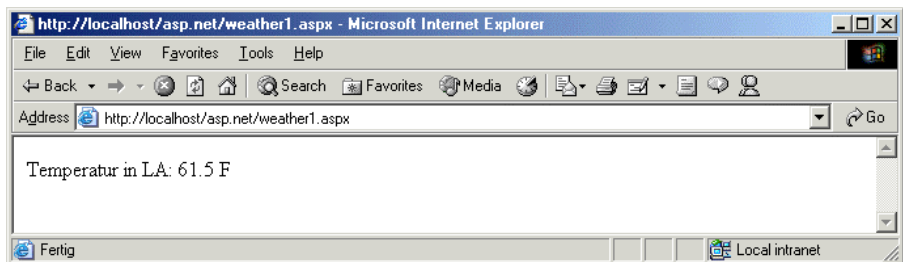


Abbildung 14.25 Das Wetter in Los Angeles, CA.

Die Temperatur in L.A. liegt bei 61,5° Fahrenheit, was 16,4° Celsius entspricht. Nicht gerade viel; bedenkt man, dass es, während ich dies schreibe, gerade einmal 2 Uhr in der Nacht Ortszeit ist, so ist die Temperatur für den Mai durchaus in Ordnung. Quizfrage: wie spät ist es hier? Na ja, wie auch immer, der Web Service funktioniert und wurde mit einfacher Unterstützung des UDDI-Verzeichnisses gefunden.

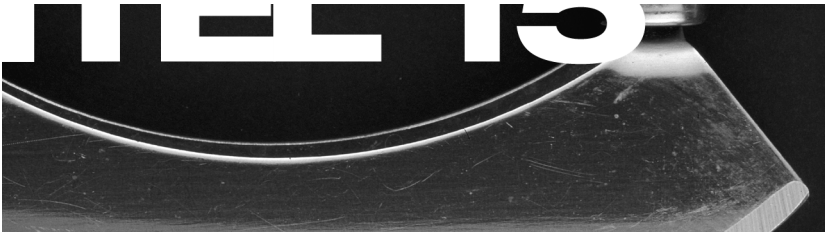
Teil II – Lösungen



Der sich anschließende zweite Teil des Buches enthält komplexe Lösungen. Eine Lösung fasst zumeist mehrere Rezepte in einem logischen Anwendungskontext zusammen.

15

Content-Management



Wie kann ich ...

15 Content-Management

Die Erstellung und Verwaltung von Inhalten gehört zu den primären Aufgaben einer datenbankbasierten Web-Applikation. Statt Daten und Inhalte in statischen Seiten abzulegen, werden diese über eine Backend-Administration oder automatisiert eingepflegt. Im Frontend werden die Daten *on the fly* aufbereitet und dem Besucher in der gewünschten Form präsentiert.

Dieses Kapitel befasst sich mit unterschiedlichen Bereichen des Content-Managements, zählt nützliche Tipps auf und verschafft mit wertvollen Rezepten Einblick in die Möglichkeiten.

15.1 ... ein Excel-Sheet dynamisch erstellen?

Dass Excel ein umfangreiches Objektmodell zum programmatischen Zugriff bietet, ist hinlänglich bekannt. Es können hierüber dynamisch und *on the fly* Sheets erstellt, Daten eingepflegt und Diagramme angezeigt werden. Die Möglichkeiten entsprechen weitgehend der Desktop-Applikation selbst, kommen natürlich nur ohne Benutzerschnittstelle aus.

Ein grundlegendes Muss für die Erstellung von Sheets *mit* Excel ist, dass das Programm auf dem Server installiert ist. Eigentlich logisch. Alternativ dazu können Sheets jedoch auch *ohne* Excel und dafür in Verbindung mit einer Komponente eines Drittherstellers erstellt werden. Dies soll jedoch nicht Bestandteil dieses Rezepts sein. Um das weiter unten gezeigte Beispiel auszuprobieren, benötigen Sie daher eine aktuelle Excel-Version, beispielsweise 2000 oder 2002/XP.

Erstellung der Proxy-DLL

Um ein Excel-Sheet auf dem Server zu erstellen, ist im Prinzip nur gewisse Kenntnis des Excel-Objektmodells notwendig. Allerdings ist es nicht ganz so leicht, auf dieses zuzugreifen, denn schließlich handelt es sich um ein COM Automation-Schnittstelle und somit unmanaged Code, der in .NET nicht gerne gesehen wird.

Um auf den Code zuzugreifen, müssen Sie zunächst eine Proxy-DLL erstellen. Diese bietet das Objektmodell als managed Code an und schleust alle Anfragen an Excel weiter. Wie dies prinzipiell geht, erfahren Sie im Rezept „... eine COM-Komponente in ASP.NET verwenden?“. Im konkreten Fall müssen Sie das Tool benutzen, um eine Proxy-DLL für die Datei `excel.olb` zu erstellen, die sich im Office-Installationsverzeichnis befindet:

```
tlbimp c:\programme\Microsoft Office\Office\EXCEL9.OLB
```

Sie erhalten eine Datei mit dem Namen `excel.dll`, die Sie in das bin-Verzeichnis Ihrer Web-Applikation verschieben.

Alternativ können Sie die DLL auch mit Hilfe von Visual Studio .NET erstellen. Wechseln Sie hierzu in den Dialog `Project > Add Reference`, und fügen Sie eine neue Referenz auf „Microsoft Excel ...“ ein. Die Abbildung zeigt den entsprechenden Dialog der Entwicklungsumgebung.

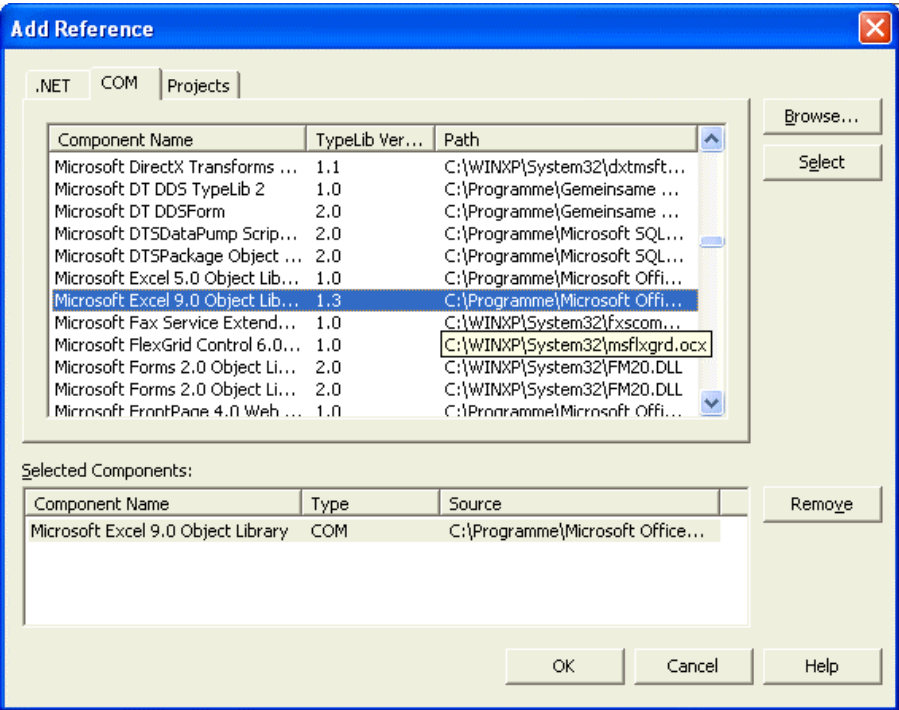


Abbildung 15.1 Die Entwicklungsumgebung erzeugt die notwendige Proxy-DLL.

Einbindung von Excel

Ist die Proxy-DLL im bin-Verzeichnis der Web-Applikation abgelegt, so wird diese Assembly automatisch geladen, und Sie können auf die enthaltenen Klassen zugreifen. Hierbei handelt es sich – wie beschrieben – um Wrapper-Klassen, die alle Aufrufe an das eigentliche Excel-Objektmodell weiterreichen. Um einen direkten Zugriff zu erhalten, können Sie den Namespace in Ihre Seiten importieren:

```
<% @Import Namespace="Excel" %>
```

Anschließend stehen Ihnen die Objekte wie `Application`, `Workbook` und `Worksheet` wie gewohnt zur Verfügung. Das folgende Beispiel zeigt deren Verwendung.

Listing 15.1 Excel1.aspx

```
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    Excel.Application excel = new Excel.Application();
    Excel.Workbook workbook = excel.Workbooks.Add("Arbeitsmappe");
    Excel.Worksheet worksheet = (Excel.Worksheet) workbook.Worksheets[1];

    Excel.Range range = (Excel.Range) worksheet.Cells[1,1];
    range.Value="hallo Welt";

    string tempfile = Path.GetTempFileName();
    File.Delete(tempfile);
    workbook.Close(true, tempfile, false);
    excel.Quit();

    Response.ContentType = "application/vnd.ms-excel";
    Response.WriteFile(tempfile);
    File.Delete(tempfile);
}

</script>
```

Was passiert im gezeigten Beispiel? Zunächst wird Excel gestartet, indem das `Application`-Objekt instanziiert wird. Nun wird eine neue Arbeitsmappe auf Basis der gleichnamigen Vorlage angelegt.



Hier zeigt sich bereits ein wichtiger Unterschied zum Zugriff auf das Objektmodell beispielsweise aus VBA. Optionale Parameter werden nicht unterstützt und auch nicht als Überladungen implementiert. Somit müssen Sie immer alle Parameter explizit mit Ihren Standardwerten übergeben.

Die neue Arbeitsmappe enthält gleich auch ein Worksheet, das über die gleichnamige Collection abgefragt werden kann. Nun kommt, was immer kommen soll. Es bleibt Ihnen überlassen, was Sie an dieser Stelle mit dem Arbeitsblatt anstellen. Sie können einfach Daten übergeben oder auch komplexe Auswertungen realisieren – Sie haben die freie Auswahl. Ich habe mich für eine sehr einfache Verwendung von Excel entschieden und weise lediglich der ersten Spalte in der ersten Reihe den Text „hallo welt“ zu.

Ist die Arbeitsmappe erstellt, soll diese natürlich noch zum Client übertragen werden. Aus ziemlich offensichtlichen Gründen bietet Excel keine Unterstützung für die .NET-Stream-Klassen, und so bleibt nichts anderes übrig, als die Datei unter einem temporären Dateinamen abzuspeichern. Wie Sie einen solchen erstellen, sehen Sie im Listing, und erfahren Sie genauer im Rezept „... einen eindeutigen temporären Dateinamen erstellen?“ im Kapitel „Dateisystem“.

Da die statische Methode `Path.GetTempFileName` die temporäre Datei bereits mit 0 Byte anlegt, muss diese *unbedingt* zunächst wieder gelöscht werden, bevor die Arbeitsmappe darunter mit Hilfe von `Close` abgespeichert wird. Nun kann die Datei mit Hilfe von `Response.WriteFile` an den Client übertragen und anschließend wieder gelöscht werden.

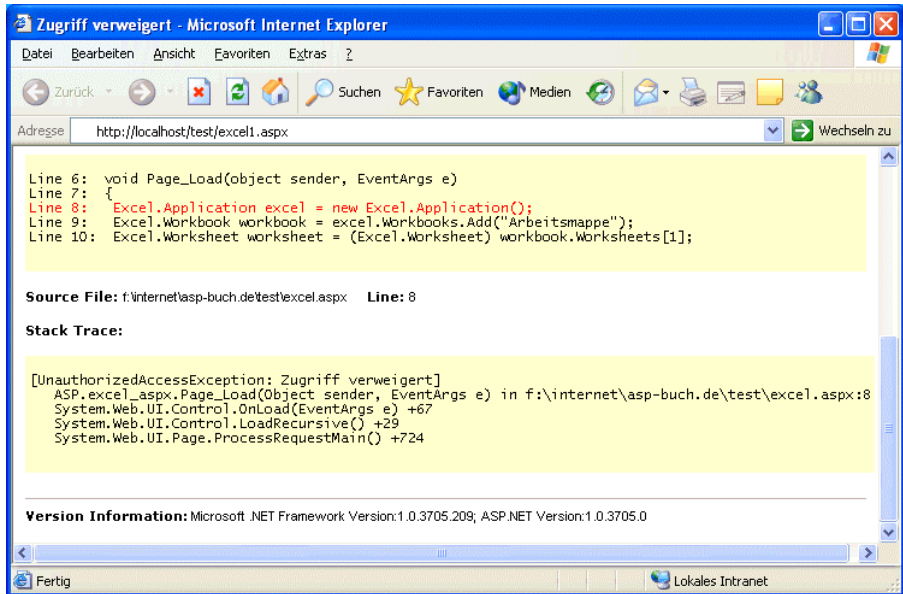


Abbildung 15.2 Der Zugriff auf Excel wurde verweigert.



Möglicherweise erhalten Sie beim Aufruf des Beispiels eine Zugriffsverweigerung. In diesem Fall verfügt der Benutzer-Account „ASPNET“ nicht über die notwendigen Dateirechte auf das Excel-Installationsverzeichnis. Sie müssen diese explizit vergeben, indem Sie die Eigenschaften des übergeordneten Verzeichnisses im Explorer öffnen und auf der Sicherheitslasche „Full Control“ für den genannten Benutzer aktivieren.

Berücksichtigen Sie bitte zudem auch folgende Hinweise:

- Vergeben Sie explizite Dateirechte auf das Verzeichnis, in dem die Arbeitsmappe temporär abgelegt wird. Sie können dieses Verzeichnis über `Path.GetTempPath` abfragen.
- Löschen Sie die temporäre Datei, bevor Sie die Arbeitsmappe abspeichern.
- Setzen Sie den korrekten MIME-Typen „application/vnd.ms-excel“ über die Eigenschaft `Response.ContentType`, bevor Sie die Daten an den Client übertragen.
- Löschen Sie die temporäre Datei, nachdem Sie deren Inhalt an den Client übertragen haben.
- Beenden Sie Excel unbedingt mit Hilfe der Methode `Quit`, da ansonsten für jeden Aufruf eine separate Instanz des Programms im Arbeitsspeicher bestehen bleibt.

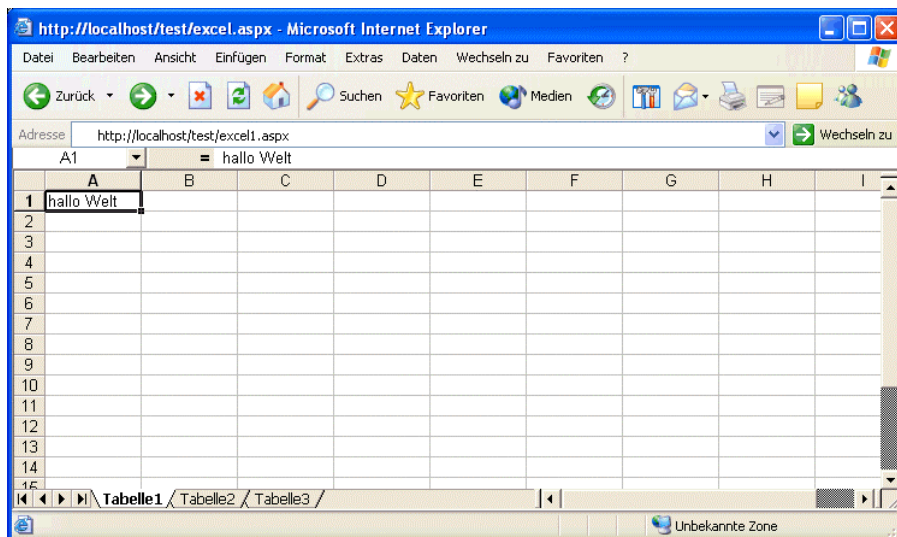


Abbildung 15.3 Das Excel-Sheet wurde on the fly erstellt.

Haben Sie alle Hinweise berücksichtigt, werden Sie mit der in der zweiten Abbildung gezeigten Pracht beglückt. Das dynamisch erstellte Excel-Sheet wurde erfolgreich an den Client übertragen und wird dort vom Excel-PlugIn direkt im Browserfenster angezeigt. Voraussetzung hierfür ist natürlich, dass Excel auf dem Zielrechner installiert ist. Ansonsten würde der Benutzer zum Speichern der Datei aufgefordert. Wie, Sie möchten genau dies erreichen, damit der Benutzer die Excel-Datei abspeichern und weiterverwenden kann? Kein Problem, einige kleinere Änderungen am Beispiel genügen. Beachten Sie bitte das folgende Listing:

Listing 15.2 Excel2.aspx

```
...

Response.ContentType = "application/x-msdownload;filename=test.xls";
Response.AppendHeader("Content-Disposition", "attachment;
    filename=test.xls; alternative: inline");

Response.WriteFile(tempfile);
File.Delete(tempfile);
}

</script>
```

Über den modifizierten Content-Type sowie den zusätzlichen Kopfzeileneintrag erscheint nun immer der Dialog „Speichern unter...“ des Browsers. Der dabei vorgeschlagene Dateiname kann individuell festgelegt werden und wird dazu sowohl

dem Content-Type als auch dem zusätzlichen Eintrag „Content-Disposition“ übergeben. Wie im Listing und der Abbildung zu sehen, wurde im Beispiel `test.xls` angegeben.

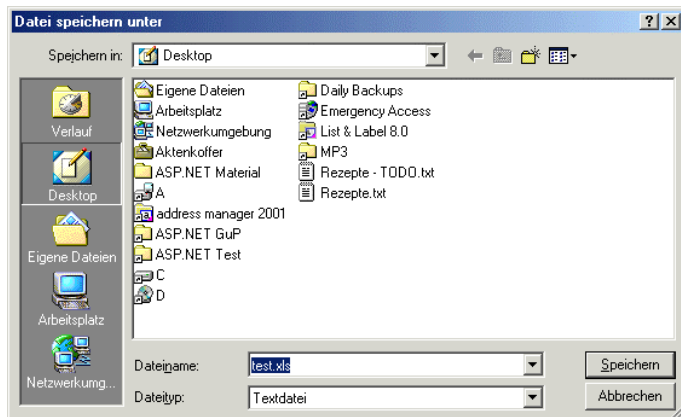


Abbildung 15.4 Die Datei kann vom Benutzer abgespeichert werden.

15.2 ... eine dynamisch erstellte PDF-Datei im Browser anzeigen?

Das Format PDF des kalifornischen Herstellers Adobe hat sich als fester Standard im Internet etabliert. Es erlaubt die plattformübergreifende Verteilung von druckbaren Daten. Das Format gilt als (nahezu) virenfrei und ermöglicht so einen risikoarmen Download, weswegen insbesondere Systemadministratoren PDF Word-Dokumenten vorziehen.

Dynamisches Generieren

In aller Regel werden PDF-Dateien aus einem Anwendungsprogramm wie Word, Excel oder PowerPoint erstellt und anschließend statisch auf dem Web-Server zum Download angeboten. Oftmals ist es jedoch wünschenswert, die Daten dynamisch entsprechend einer Benutzerauswahl oder -eingabe zu generieren. Was bisher nur unter großem Aufwand möglich war, ist mit Hilfe der .NET-Komponente List & Label ein Kinderspiel.

combit **List & Label®**

Welcome

Start Sample

Go to...

combit Website

L&L Website

Order L&L

L&L Website

Create personalized reports on the fly

This sample shows how to create personalized reports just in the time your visitors ask for them. For demonstration purposes please enter your firstname, lastname and email-address or of course any fictional data you want to use.

Your firstname:

Your lastname:

Your email-address:

Output format:

Create report

Local intranet

Abbildung 15.5 Das Formular fordert zur Eingabe persönlicher Daten auf.

List & Label ist ein Reportgenerator, der im Verlaufe von insgesamt acht Produktversionen sowohl in Deutschland als auch weltweit in zahllosen Applikationen Einsatz findet. Gegenüber der Konkurrenz hebt sich das deutsche Produkt insbesondere durch den frei distribuierbaren Designer für Endkunden aus. Hier kann per WYSIWYG wie in einem DTP-Programm eine Vorlage gestaltet werden. Die hinterlegten Platzhalter werden zur Laufzeit mit Daten gefüllt. Neben der Ausgabe auf dem Drucker stehen auch zahlreiche Exportformate zur Verfügung, die eine Verteilung und Weiterbearbeitung erlauben.

Als einer der ersten Tool-Hersteller hat combit bereits zur .NET Beta 1 eine Version der Komponente für Visual Studio .NET kostenlos für bestehende Kunden angeboten. Die Komponente wurde seitdem sehr stark erweitert und in die Architektur von .NET integriert. Besonderes Merkmal ist die Datenbindung an nahezu beliebige Quellen wie DataSet, DataReader, IEnumerable (auf Basis von Reflection) und so weiter.

Das Produkt aus dem Hause des deutschen Software-Herstellers kann direkt innerhalb von ASP.NET Web-Applikationen beispielsweise zum Generieren von PDF-Dateien genutzt werden. Der Designer steht derzeit ausschließlich als Desktop-Applikation zur Verfügung.

Beispiel mit combit List & Label

Das folgende Beispiel zeigt die Verwendung von List & Label innerhalb einer Web-Applikation. Die Sourcen verwende ich mit freundlicher Genehmigung von combit; sie entstammen der 30-Tage-Trial-Version¹.

Das in der ersten Abbildung zu sehende Web-Formular enthält mehrere Eingabefelder für persönliche Daten des Besuchers. Zudem wurde eine DropDownList zur Auswahl des gewünschten Zielformates integriert. Neben PDF können hier Formate wie HTML, MHTML, XML, RTF, JPEG sowie das interne Format von List & Label ausgewählt werden. Ein Klick auf den Button „Create Report“ erzeugt den Report und zeigt ihn im Browser an.

Listing 15.3 personal.aspx

```
<%@ Import Namespace="combit.ListLabel8x" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>

<script runat="server">

protected ListLabel LL;
protected string TempDir;
protected string Format;
protected string Filename;

protected void CreateReport(object sender, EventArgs e)
{
    // Initialize L&L
    LL = new ListLabel(LlLanguage.Default, false);

    LL.Debug = LlDebug.Enabled | LlDebug.LogToFile;

    // Set options
    LL.Core.LlSetOptionString(LlOptionString.LLXPathList, "^*");
    LL.Core.LlSetOptionString(LlOptionString.LLXPathList, "");
    LL.DefineVariables += new DefineVariablesHandler(LlDefineVariables);
    LL.DefinePrintOptions +=
        new DefinePrintOptionsHandler(LlDefinePrintOptions);

    ...

    LL.Print(LlProject.Card, ProjectFile, false, LlPrintMode.Export,
        LlBoxType.NormalMeter, "", false, TempDir);
```

¹ Auch wenn das Beispiel nicht speziell für dieses Buch geschrieben wurde, so stammt es doch aus meiner Feder. Bis Juni 2002 habe ich bei combit gearbeitet und in diesem Zusammenhang auch verantwortlich die hier vorgestellte .NET-Komponente konzipiert und entwickelt.

```
LL.Dispose();

if(Format == "PRV")
    Response.Redirect("displayllfile.aspx?file=temp/" +
        Session.SessionID + "/" + Filename, true);
else if(Format == "PDF")
{
    Response.ContentType = "application/pdf";
    Response.WriteFile("temp/" + Session.SessionID + "/" + Filename);
    Response.End();
}
else
    Response.Redirect("temp/" + Session.SessionID + "/" + Filename,
        true);
}

...

private void LlDefineVariables(object sender,
    DefineElementsEventArgs e)
{
    // Define variables
    LL.Variables.Add("Firstname", tb_firstname.Text);
    LL.Variables.Add("Lastname", tb_lastname.Text);
    LL.Variables.Add("Email", tb_email.Text);

    // Resolve Hostname
    IPHostEntry UserIPHost =
        Dns.Resolve(Request.ServerVariables["REMOTE_ADDR"]);
    LL.Variables.Add("Host", UserIPHost.HostName);
    LL.Variables.Add("IP", UserIPHost.AddressList[0].ToString());

    ...

    if(!e.IsDesignMode) e.IsLastRecord = true;
}

...
```

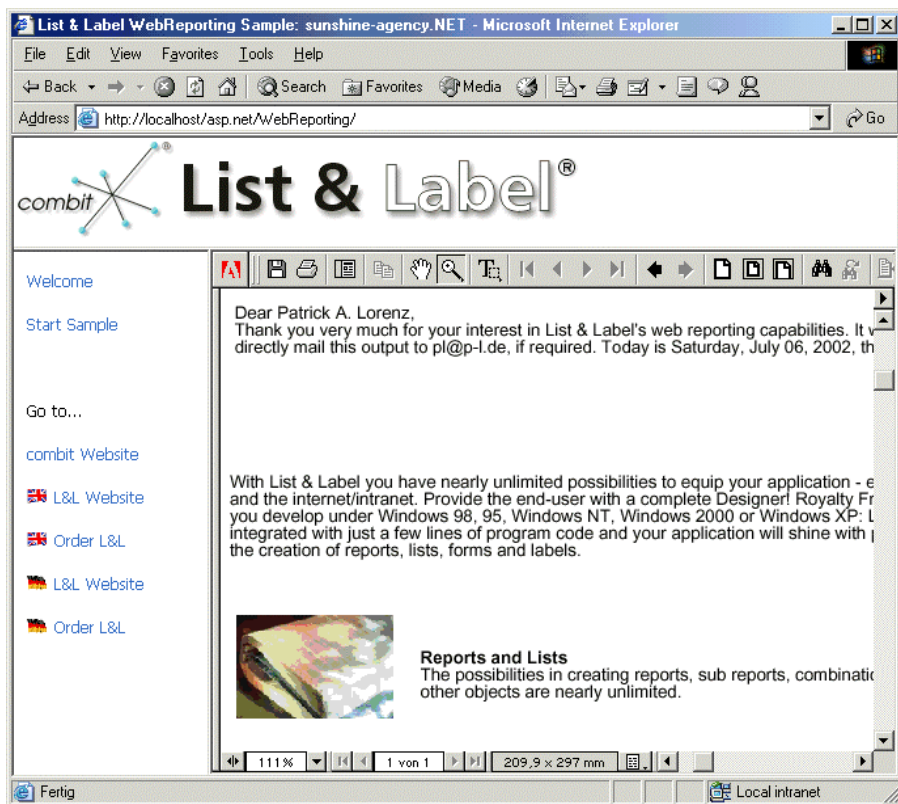


Abbildung 15.6 Das PDF wurde on the fly und personalisiert erstellt.

Das Listing zeigt die interne Verarbeitung beim Klick auf den Button. Es wird eine neue Instanz der Komponente angelegt und mit einigen Werten initialisiert. Über das zugeordnete Ereignis `DefineVariables` fragt List & Label die Benutzerdaten nach dem Aufruf der `Print`-Methode ab. Die erzeugte Datei wird anschließend an den Client übertragen. Der Beweis ist in Abbildung zwei zu sehen. Die PDF-Datei wird über das Adobe Acrobat-PlugIn direkt im Browser angezeigt. Die Personalisierung mit Hilfe der eingegebenen Daten ist deutlich erkennbar.

Alternativ stehen wie beschrieben auch weitere Formate zur Verfügung. Das interne Format von List & Label ist hervorzuheben. Wie PDF erlaubt auch dieses eine einfache Distribution inklusive Druckmöglichkeit ohne jeglichen Qualitätsverlust. Die Anzeige im Browser erfolgt unter Windows mit Hilfe eines ActiveX-PlugIns. Ein Klick genügt, um die Datei auf dem lokalen Drucker auszugeben.

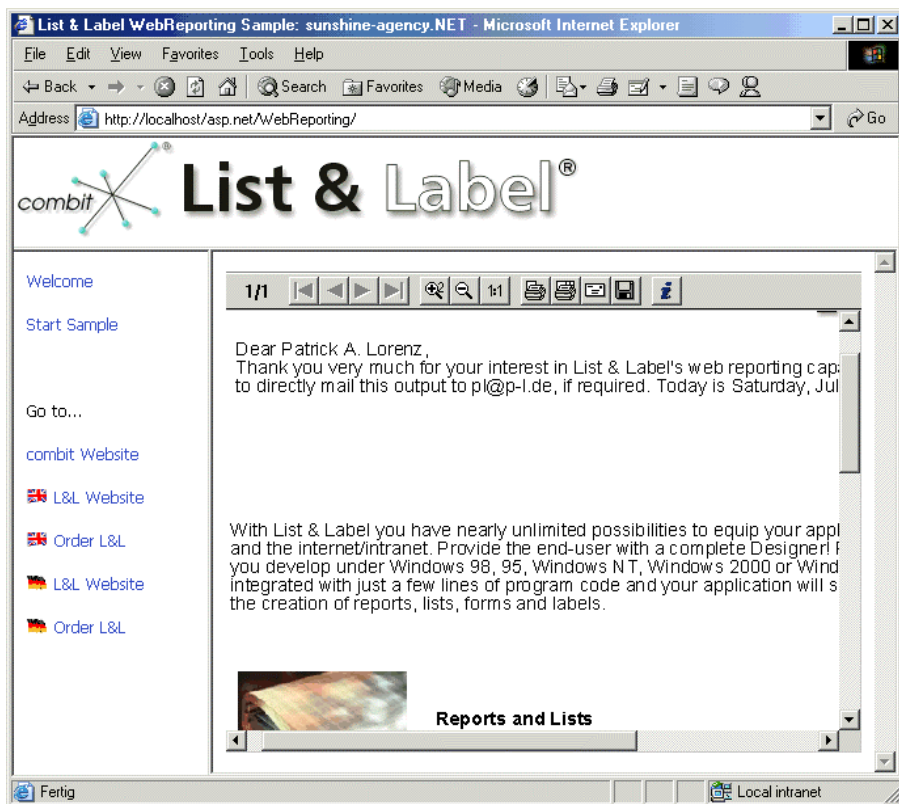


Abbildung 15.7 Das Dokument wird über ein ActiveX-Plugin angezeigt.

List & Label bietet sich für ASP.NET-Entwickler insbesondere zur dynamischen Erzeugung von Reports und Druckergebnissen an. Diese können einfach heruntergeladen oder auch per Email versendet werden. Ein Beispiel für den Einsatz ist die Online-Generierung von Rechnungen in einem Online-Shop. Der Einkäufer kann sich die Rechnung so bequem auf dem lokalen Drucker ausgeben lassen, muss aber nicht die üblichen Einschränkungen beim Druck von HTML-Seiten im Browser hinnehmen.

Weitere Informationen zu List & Label finden Sie auf der Website des Unternehmens unter folgender Adresse:

<http://www.combit.net>

Die zur Drucklegung des Buches aktuelle 30-Tage-Trial-Version finden Sie auf der begleitenden CD-ROM. Sie enthält alle notwendigen Komponenten sowie zahlreiche Beispiele, darunter auch zu Visual Basic .NET und C#. Während ich dies schreibe, ist die neue Version 9.0 bereits in Arbeit. Auch die .NET-Komponente

wird mit diesem Release noch einmal stark erweitert werden. Die jeweils aktuelle Trial-Version finden Sie auf der Website zum Download. Dort können Sie List & Label auch direkt online erwerben. In der Knowledge Base sowie der Newsgroup erhalten Praxistipps zum Einsatz des Produktes.

Download von PDF-Dateien

Aufgrund eines Fehlers im Internet Explorer ist der Download von PDF-Dateien mitunter problematisch. Dies gilt insbesondere, wenn die Datei per `Response.Redirect` an den Client gesendet wird. Je nach Systemkonstellation erhält der Besucher nur eine leere Seite, obwohl das Dokument korrekt an den Browser übertragen wurde. Abhilfe schafft die Verwendung von `Response.WriteFile`, wie im folgenden Beispiel gezeigt:

```
Response.ContentType = "application/pdf";  
Response.WriteFile(fileName);  
Response.End();
```

Beachten Sie unbedingt den Aufruf von `Response.End`, da ansonsten der HTML-Inhalt der Seite ebenfalls übertragen und die Darstellung der PDF-Datei somit nicht möglich wäre. Auf das hier beschriebene Problem geht auch der Artikel Q247663 aus der Microsoft Knowledge Base mit dem Titel „pdf File Is Displayed as a Blank Page on Redirects“ ein. Dort finden Sie weitere Hintergrundinformationen.

15.3 ... Inhalte anderer Seiten scrapen?

Das englische Verb *scrape* (wohlgemerkt mit „e“) bedeutet so viel wie abschaben oder auch abkratzen. Webmaster verstehen darunter das komplette Übernehmen von Inhalten anderer Websites, unabhängig davon, ob es sich um eine freundliche oder feindliche Übernahme handelt. In diesem Kontext soll auch nicht eine weitere, passende Bedeutung des Wortes unterschlagen werden. Demnach wird *scrape* mit „sich über Wasser halten“ ins Deutsche übersetzt. Doch zurück zur Technik ...

Das Übernehmen von anderen Seiten ist mit Hilfe von ASP.NET ein absolutes Kinderspiel. Die Klasse `WebRequest` aus dem Namespace `System.Net` stellt sich gehorsam Ihren Befehlen zur Verfügung. Deren statischer Methode `Create` übergeben Sie die gewünschte Adresse. Zurückgeliefert wird eine Instanz der Klasse beziehungsweise einer Ableitung. Hier können Sie die Adresse mittels `GetResponse` abfragen und auf das Ergebnis über einen `Stream` zurückgreifen.

Listing 15.4 srcape1.aspx

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="System.Net" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string url = "http://localhost/asp.net/test.aspx";

    WebRequest request = WebRequest.Create(url);
    WebResponse response = request.GetResponse();
    StreamReader reader = new StreamReader(response.GetResponseStream());
    Response.Write(reader.ReadToEnd());
}

</script>
```

Das Listing zeigt den Einsatz der Technik. Es wird der Inhalt einer anderen Seite abgefragt und im Browser ausgegeben. Die Abbildung zeigt das unscheinbare Ergebnis.



Abbildung 15.8 Der Inhalt entstammt einer intern abgefragten Seite.

Statt derart einfache Seiten können Sie selbstverständlich auch umfangreichere übernehmen. Der Umfang der Daten ist keinerlei Beschränkungen unterworfen, wohl aber die Verwendung von Grafiken. Ein Großteil der Websites gibt statt absoluter lieber relative Dateinamen für verwendete Grafik an. Nicht ohne Sinn, denn so kann die Größe der HTML-Seite mitunter deutlich verringert werden. Werden diese Daten jedoch übernommen und über einen anderen Server an den Client übertragen, sind diese relativen Pfade nicht mehr zu verwenden. Das Verhalten lässt sich auch mit Hilfe eines Unterverzeichnisses nachvollziehen. Die zweite Abbildung zeigt das Problem.



Abbildung 15.9 Die Grafik kann nicht gefunden werden.

Sofern Sie selbst Entwickler der übernommenen Seite sind, empfiehlt sich eine entsprechende Anpassung und die Verwendung von absoluten Grafikpfaden. In anderen Fällen müssen Sie die Seite parsen und alle Pfade „umbiegen“. Dies gilt übrigens im vollen Maße auch für alle sonstigen relativen Links.

Ich höre Sie die Lösung für Probleme wie dieses schon sagen ... genau, es ist wieder Zeit für reguläre Ausdrücke. Über die Methode `Replace` können alle `img`-Tags gesucht und mit einem absoluten Pfad versehen werden. Das Beispiel zeigt dies in vereinfachter Form.

Listing 15.5 `srcscape3.aspx`

```
<% @Import Namespace="System.IO" %>
<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.Text.RegularExpressions" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string url = "http://localhost/asp.net/subdir/test2.aspx";

    WebRequest request = WebRequest.Create(url);
    WebResponse response = request.GetResponse();
    StreamReader reader = new StreamReader(response.GetResponseStream());

    string PageContents = reader.ReadToEnd();
    Regex regex = new Regex(@"<img.+src=\"");
    PageContents = regex.Replace(PageContents,
        "$0http://localhost/asp.net/subdir/");

    Response.Write(PageContents);
}

</script>
```

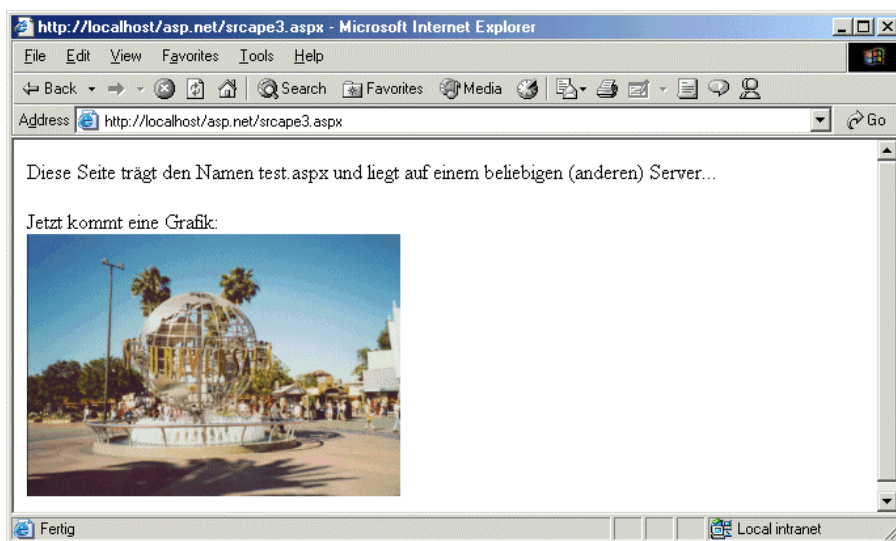


Abbildung 15.10 Der Grafikpfad wurde per regulärem Ausdruck umgebogen.

Und? Klappt's? Natürlich, die Grafik im Beispiel erstrahlt nun in voller Pracht, den Universal Studios sei Dank. Die Abbildung zeigt's. Übrigens: Im Kapitel „User Controls“ findet sich der gezeigte Quellcode zum Scrapen von Seiten wieder. Im Rezept „... ein komplexes Custom Control erstellen?“ wird die Funktionalität zur Ausgabe eines eigens entwickelten Custom Controls gemacht.



Rein technisch spricht nichts gegen die Verwendung dieser Technik, rechtlich schon. Ich gehe davon aus, dass Sie sich vor der Einbindung von fremden Websites mit den jeweiligen Anbietern und Rechteinhabern auseinander setzen und so die rechtliche Grundlage festigen. Auch wenn viele es leider anders sehen, auch im Internet ist mit dem Urheberberrecht nicht zuletzt aus moralischen Gründen nicht zu scherzen. </moralpredigt>

15.4 ... eine Tabelle parsen?

In Ergänzung zu dem vorangegangenen Rezept möchte ich Ihnen noch eine interessante Möglichkeit vorstellen, wie Sie ganz einfach Tabellen parsen können, die Sie anderen Seiten (hoffentlich legal) entnommen haben. Wenn Sie schon einmal eine Tabelle geparkt haben, dann wissen Sie vermutlich, wie nervig das ist. Schöner ist da doch, wenn man die Arbeit jemand anderem überlassen kann. Und wer könnte das schon sein? Klar, ASP.NET und die Klassen des .NET Framework.

Im Rezept „... ein Control aus einer Zeichenkette parsen?“ im Kapitel „User Controls“ haben ich Ihnen eine Möglichkeit beschrieben, zur Laufzeit aus einer Zeichenkette ein voll funktionsfähiges Server Control zu gewinnen. Mit den hier vorgestellten Möglichkeiten und ein paar Tricks können Sie eine Zeichenkette in eine Instanz der Klasse `HtmlTable` umwandeln und die einzelnen Elemente der Tabelle über das Objektmodell abfragen.

Zunächst möchte ich Ihnen die zu parsende Tabelle vorstellen. Diese befindet sich in einer statischen HTML-Seite. Sie enthält zwei Spalten und drei Zeilen. Während fünf der Zellen irrelevante Daten enthalten, befindet sich in der ersten Spalte der dritten Zeile der abzufragende Wert.

Listing 15.6 table.html

```
<html>
<head/>
<body>

  <table width="100%">
    <tr>
      <td> irgendwas </td>
      <td> irgendwas </td>
    </tr>

    <tr>
      <td> irgendwas </td>
      <td> irgendwas </td>
    </tr>

    <tr>
      <td> Hallo Welt </td>
      <td> irgendwas </td>
    </tr>
  </table>

</body>
</html>
```

Selbstverständlich könnten Sie das Element bei einem so einfachen Beispiel auch über reguläre Zeichenkettenoperationen herausfiltern. Das folgende Listing jedoch zeigt einen eleganteren Weg. Zunächst muss die Seite über die Klasse `WebClient` abgefragt werden. Die Methode `OpenRead` liefert einen `Stream`, der über eine neue Instanz der Klasse `StreamReader` ausgelesen und in einer Zeichenkette abgelegt wird. Da diese nun die komplette Seite enthält, muss zunächst die Tabelle ausgeschnitten werden. Hier lassen sich Zeichenkettenoperationen genauso gut wie ein regulärer Ausdruck verwenden.

Ist die Tabelle ausgeschnitten, wird dem HTML-Tag `table` mit Hilfe der Methode `string.Replace` das Attribut `runat="server"` angefügt. Anschließend wird die Zeichenkette der Methode `Page.ParseControl` übergeben. Diese wandelt die Daten in ein Server Control um, das nun in Form einer Instanz der Klasse `HtmlTable` abgefragt werden kann.

Im letzten Schritt verwenden Sie das Objektmodell der Klasse, um die dritte Zeile und daraus die erste Spalte beziehungsweise Zelle abzufragen. Die Eigenschaft `InnerText` liefert deren Inhalt.

Listing 15.7 ParseTable1.aspx

```
<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.IO" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string url = "http://localhost/asp.net/table.html";

    WebClient webclient = new WebClient();
    StreamReader reader = new StreamReader(webclient.OpenRead(url));

    string rawdata = reader.ReadToEnd();

    int pstart = rawdata.IndexOf("<table");
    int pend   = rawdata.IndexOf("</table>", pstart);
    rawdata = rawdata.Substring(pstart, pend-pstart+8);

    rawdata = rawdata.Replace("<table ", "<table runat=\"server\" ");
    HtmlTable table = (HtmlTable) this.ParseControl(rawdata).Controls[0];

    HtmlTableCell cell = table.Rows[2].Cells[0];
    string value = cell.InnerText;

    Response.Write(value);
}

</script>
```

Und nun raten Sie bitte, welche Ausgabe das Beispiel im Browser erzeugt. Na klar, das ist eigentlich keine Frage, aber sehen Sie selbst ...

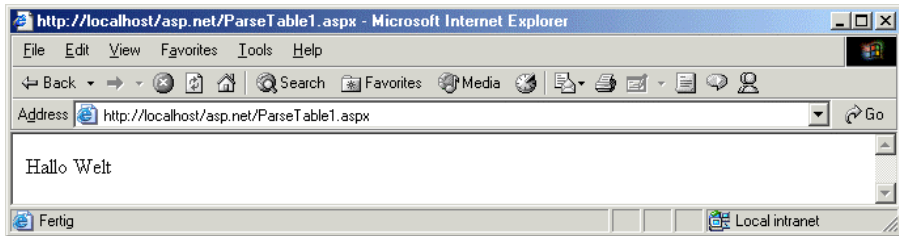


Abbildung 15.11 Der Text wurde aus der Tabelle einer anderen Seite übernommen.

Statt Teile zu übernehmen, können Sie natürlich auch andere Operationen mit Hilfe der `HtmlTable`-Instanz durchführen. Das Objektmodell steht Ihnen uneingeschränkt zur Verfügung. Sie können beispielsweise die Hintergrundfarbe einer Zelle ändern und das Control anschließend der Controls-Collection übergeben und somit im Browser ausgeben ...

Listing 15.8 ParseTable2.aspx

```
rawdata = rawdata.Replace("<table ", "<table runat=\"server\" ");
HtmlTable table = (HtmlTable) this.ParseControl(rawdata).Controls[0];

HtmlTableCell cell = table.Rows[2].Cells[0];
cell.BgColor = "Red";

this.Controls.Add(table);
}
</script>
```

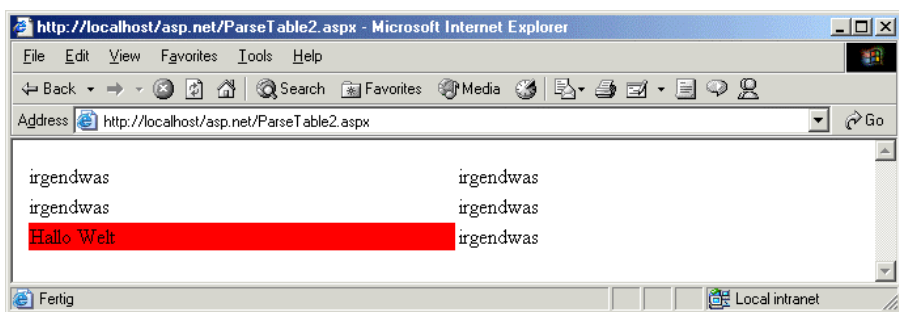


Abbildung 15.12 Die Hintergrundfarbe wurde geändert.

Wenngleich die Tabelle eines der komplexesten HTML-Tags ist, lässt sich analog auch mit anderen Objekten verfahren. Sie müssen diese lediglich in einer Zeichenkette atomisieren und mit dem `runat`-Attribut versehen. Anschließend können Sie das Objektmodell des jeweiligen HTML Server Controls uneingeschränkt verwenden.

15.5 ... einen Linkzähler einrichten?

Viele Mailings und Aktionen enthalten einen Link, der auf weitergehende Informationen im Internet führt. Doch wie erfolgreich ist dieser Link eigentlich, und wie häufig wird er benutzt? Mit einem einfachen Linkzähler können Sie diese Frage schnell beantworten. Das System ist einfach. Statt auf den eigentlichen Hinweis zu verweisen, ruft das Mailing den Linkzähler auf. Im Query-String wird die ID des gewünschten Links übergeben. Dessen Hit-Counter wird inkrementiert und ein Redirect auf die eigentliche Seite durchgeführt, deren Adresse in der Datenbank hinterlegt ist.

Datenbank

Zunächst gilt es natürlich, eine Datenbank als Basis einzurichten. Diese muss neben dem obligatorischen Primärindex zumindest ein Feld für den Hit-Counter sowie die Adresse der eigentlichen Seite enthalten. Die Abbildung zeigt die Struktur der Tabelle `LinkCounter`.

Die beiden zusätzlichen Felder `Text` und `ServerRedirect` enthalten einen optionalen Beschreibungstext sowie die Information, ob es sich um eine lokale Adresse handelt.

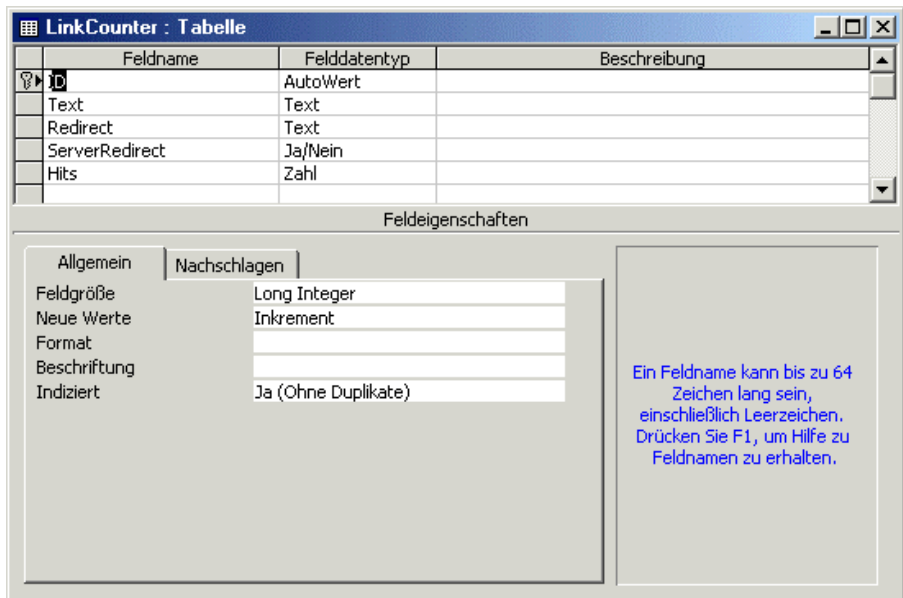


Abbildung 15.13 Die Tabelle bildet die Basis des Linkzählers.

Linkzähler

Der Linkzähler besteht aus einer einfachen ASP.NET-Seite, die die beschriebenen Daten im Page_Load-Ereignis abfragt und den Counter inkrementiert. Ist das Flag `ServerRedirect` gesetzt, wird die Methode `Server.Transfer` verwendet, ansonsten `Response.Redirect`. Wird kein passender Datensatz gefunden, wird ein Redirect auf die Hauptseite des aktuellen Servers durchgeführt.

Listing 15.9 LinkCounter1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    string qs = Request.ServerVariables["QUERY_STRING"];

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "UPDATE LinkCounter SET Hits=Hits+1 WHERE ID=@ID;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@ID", qs);
}
```

```
string url = "/";
bool localurl = false;

if(cmd.ExecuteNonQuery() == 1)
{
    SQL = "SELECT Redirect, ServerRedirect FROM LinkCounter
        WHERE ID=@ID;";
    cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@ID", qs);
    OleDbDataReader reader = cmd.ExecuteReader();

    if(reader.Read())
    {
        url = reader.GetString(0);
        localurl = reader.GetBoolean(1);
    }

    reader.Close();
}

conn.Close();

if(localurl)
    Server.Transfer(url);
else
    Response.Redirect(url);
}

</script>
```

Der Aufruf der Seite erfolgt einfach mit Angabe der gewünschten ID im Query-String. Das sieht zum Beispiel so aus:

```
http://localhost/asp.net/LinkCounter1.aspx?1
```

Das sieht zugegebenermaßen nicht besonders schön aus. Alternativ können Sie die Datei in `default.aspx` umbenennen und in einem speziell dafür angelegten Unterverzeichnis ablegen. Dieses könnte beispielsweise `go` heißen. Der Aufruf der Seite sieht dann gleich wesentlich schöner aus:

```
http://localhost/asp.net/go?1
```

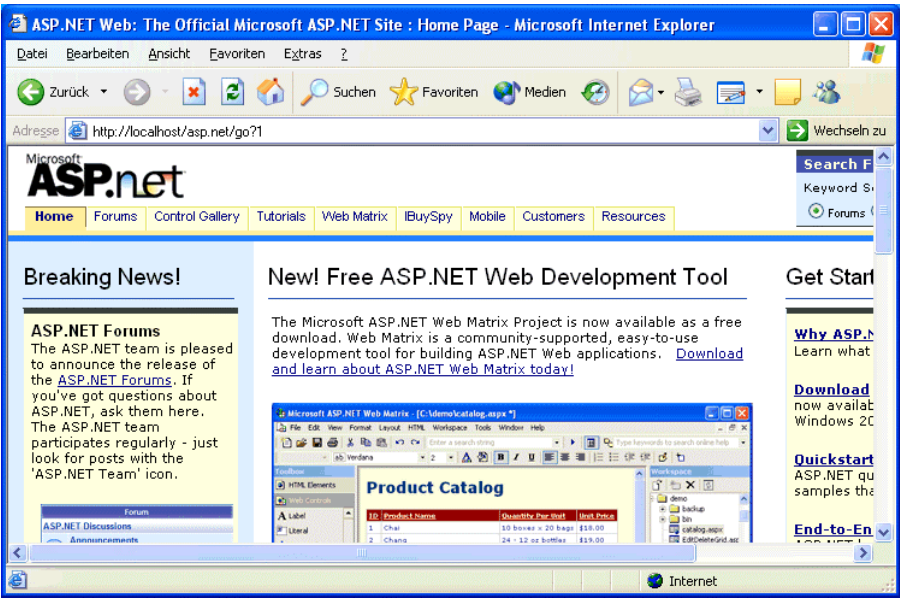



Abbildung 15.14 Die Seite wurde über den Link-Counter aufgerufen.

LinkCounter : Tabelle					
	ID	Text	Redirect	ServerRedirec	Hits
▶	1	Offizielle ASP.NET-Website	http://www.asp.net	☐	3
✱	(AutoWert)			☐	0

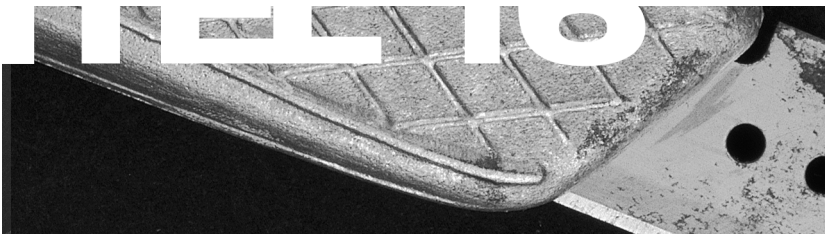
Datensatz: 1 | 1 von 1

Abbildung 15.15 Im Datensatz wurde jeder Zugriff auf den Link gezählt.

Die beiden Abbildungen zeigen die Verwendung des Linkzählers sowie die daraus korrespondierenden Einträge in der Datenbank. Das Feld `Hits` wird bei jedem Zugriff inkrementiert. Als Ergänzung dieser Funktionalität bietet sich beispielsweise ein kleines Administrationstool zur Anlage und Verwaltung der Hits an. Sollen weitergehende Auswertungen gemacht werden, empfiehlt sich die Verwendung einer relationalen Struktur, bei der jeder Hit in einem eigenen Datensatz mitsamt Uhrzeit und mit möglicherweise weiteren Informationen über den Benutzer abgelegt wird.

16

Community



Wie kann ich ...

16 Community

Das englische Wort Community wird als Gemeinde oder auch Gemeinschaft übersetzt. Das Internet hat den Begriff sehr stark geprägt, doch auch hier steht er letztlich für eine Gemeinschaft von Gleichinteressierten. Eine derartige Community wird zumeist durch ein einzelnes Internet-Angebot oder ein Portal definiert. Dieses Portal legt die Interessen der Zielgruppe fest und bietet dieser eine Plattform zum Austausch an.

Die technischen Anforderungen einer Community im Internet sind oftmals sehr ähnlich. Die angebotenen Dienste leben von der Aktualität und sollen die Interaktion der Benutzer fördern. Ohne diese Mitglieder der Gemeinschaft hört die Community auf zu existieren, wird ähnlich wie ein verlassener Ort zu einer Geisterstadt.

In diesem Kapitel möchte ich Ihnen technische Lösungen für die allerhäufigsten Anforderungen einer Community liefern. Viele Bereiche einer virtuellen Gemeinschaft sollen abgedeckt werden, wenngleich dies aufgrund der Individualität jeder Community pauschal sicher nie zu 100 Prozent gelingen kann.

Die Beispiele arbeiten zumeist auf Basis einer Datenbank. Diese liegt separat unter dem Namen `community.mdb` vor und sollte in das bekannte Verzeichnis kopiert werden; vergleiche Einführung, Kapitel 1.

16.1 ... ein News-System entwickeln?

Eine Website und insbesondere eine Community lebt von der Aktualität, und wie besser könnte man diese präsentieren, wenn nicht durch aktuelle Neuigkeiten auf der Startseite. So erfährt der Besucher auf den ersten Blick, was sich in letzter Zeit getan hat, und kann auf Wunsch auch den Verlauf der Website in den vergangenen Monaten verfolgen.

Die Anforderungen an ein News-System gegenüber statischen Inhalten sind sehr unterschiedlich. Die einheitliche Aufbereitung der Datenbankinhalte gehört jedoch ebenso zu den Standardwünschen der Kunden wie die umgekehrt chronologische Auflistung, die Anzeige eines Kurztexes sowie einer automatischen Archivierung älterer News.

In diesem Rezept möchte ich Ihnen eine beispielhafte Implementierung eines News-Systems auf Basis von ASP.NET vorstellen. Die genannten Anforderungen werden selbstverständlich berücksichtigt.

Datenbank

Die Basis eines News-Systems bildet die Datenbank. Hier sind die vorhandenen Neuigkeiten atomisiert hinterlegt. Diese Atomisierung ist sehr wichtig, denn nur so können die hinterlegten Inhalte später individuell aufbereitet werden. Die Abbildung zeigt die Tabelle `News` der Community-Datenbank und die dort hinterlegten Felder.



Abbildung 16.1 Jeder Datensatz der Tabelle repräsentiert eine Neuigkeit.

Einfache Ausgabe der News

Zur Ausgabe der News bietet sich eines der drei datengebundenen Data-Controls an. Ich habe mich für die `DataList` entschieden, da diese Klasse die größtmögliche Flexibilität mit dem größtmöglichen Komfort verbindet. Die Abfrage und Zuweisung der Daten ist wenig spektakulär, lediglich die SQL-Query zeigt die beschriebene umgekehrt chronologische Reihenfolge. Nur so wird die neueste Nachricht ganz oben und die älteste ganz unten angezeigt.

Die Visualisierung der einzelnen News erfolgt über ein hinterlegtes `ItemTemplate` des `DataList`-Controls. Es werden reguläre HTML-Tags in Verbindung mit der `DataBinding`-Syntax genutzt.

Listing 16.1 news1.aspx

```
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Data.OleDb" %>

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "SELECT * FROM News ORDER BY Date DESC;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    news.DataSource = cmd.ExecuteReader();
    news.DataBind();
    conn.Close();
}

</script>

<ASP:DataList id="news" runat="server">

    <ItemTemplate>
        <p>
            <b><%# DataBinder.Eval(Container.DataItem, "Title") %></b><br>
            <i><%# DataBinder.Eval(Container.DataItem, "Subtitle") %></i><br>
            <%# DataBinder.Eval(Container.DataItem, "Author") %> am
            <%# DataBinder.Eval(Container.DataItem, "Date") %><br>

            <%# DataBinder.Eval(Container.DataItem, "Text") %>
        </p>
    </ItemTemplate>

    <SeparatorTemplate> &nbsp; </SeparatorTemplate>

</ASP:DataList>
```

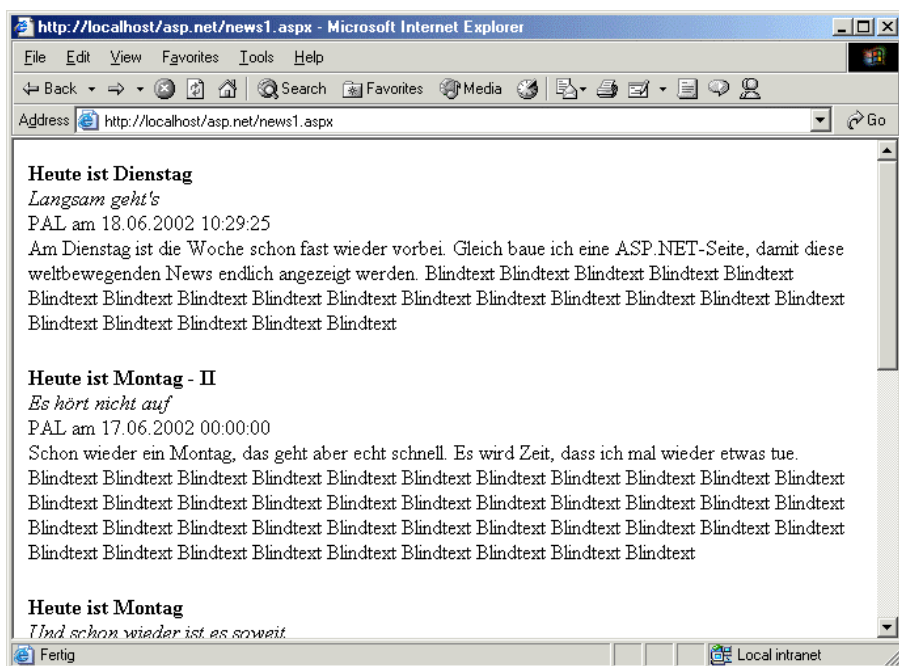


Abbildung 16.2 Die Neuigkeiten aus der Datenbank werden sortiert angezeigt.

Die Ausgabe dieses ersten Versuchs ist in der Abbildung zu sehen. Neben dem etwas obsoleten Layout fallen vor allem folgende Punkte auf:

- Es wird der gesamte Text angezeigt, nicht nur eine Kurzfassung.
- Der Text wird ohne jegliche Umbrüche dargestellt.
- Das Datum wird mit Uhrzeit ausgegeben.

Diese offensichtlichen Fehler oder Unschönheiten soll die zweite Variante ausbügeln. Um nur eine Zusammenfassung des Nachrichtentextes anzuzeigen, wurde eine Methode `GetWords` implementiert, die von einer Zeichenkette die angegebene Anzahl Wörter liefert. Soll der gesamte Inhalt angezeigt werden, kann die Zeile über einen Weiter-LinkButton selektiert werden, so dass die zweite hinterlegte Vorlage ausgewertet wird. Hier wird eine Methode `FormatText` verwendet, um die Umbrüche des Textes auf HTML umzusetzen. Beim ersten Laden der Seite ist die aktuellste Neuigkeit bereits selektiert, so dass diese mit dem ausführlichen Text angezeigt wird.

Listing 16.2 news2.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        news.SelectedIndex = 0;
        BindDataList();
    }
}

void BindDataList()
{
    ...
}

string GetWords(string source, int wordcount)
{
    int pos = 0;
    for(int i=0; i<wordcount; i++)
    {
        pos = source.IndexOf(' ', pos+1);
        if(pos == -1) return(source);
    }
    return(source.Substring(0, pos));
}

string FormatText(string source)
{
    source = source.Replace("\r\n\r\n", "</p><p>");
    source = source.Replace("\r\n", "<br>");
    return(source);
}

void news_ItemCommand(object sender, DataListCommandEventArgs e)
{
    if(e.CommandName == "select")
    {
        DataList dl = (DataList) sender;
        dl.SelectedIndex = e.Item.ItemIndex;
        BindDataList();
    }
}

</script>

<form runat="server">
```

```
<ASP:DataList id="news" runat="server"
OnItemCommand="news_ItemCommand">

<ItemTemplate>
<p>
<b><%=# DataBinder.Eval(Container.DataItem, "Title") %></b><br>
<i><%=# DataBinder.Eval(Container.DataItem, "Subtitle") %></i><br>
<%=# DataBinder.Eval(Container.DataItem, "Author") %> am
<%=# ((DateTime) DataBinder.Eval(Container.DataItem,
    "Date")).ToString("D") %><br>

<%=# GetWords(DataBinder.Eval(Container.DataItem,
    "Text").ToString(), 20) %>
<ASP:LinkButton runat="server" commandname="select"
    text="Weiter..." />
</p>
</ItemTemplate>

<SelectedItemTemplate>
<b><%=# DataBinder.Eval(Container.DataItem, "Title") %></b><br>
<i><%=# DataBinder.Eval(Container.DataItem, "Subtitle") %></i><br>
<%=# DataBinder.Eval(Container.DataItem, "Author") %> am
<%=# ((DateTime) DataBinder.Eval(Container.DataItem,
    "Date")).ToString("D") %><br>

<%=# FormatText(DataBinder.Eval(Container.DataItem,
    "Text").ToString()) %>
</p>
</SelectedItemTemplate>

<SeparatorTemplate> &nbsp; </SeparatorTemplate>

</ASP:DataList>

</form>
```

Die Abbildung zeigt das Ergebnis der überarbeiteten Variante. Die genannten Fehler sind behoben, und die Neuigkeitenliste macht abgesehen vom Layout einen schon durchaus brauchbaren Eindruck.

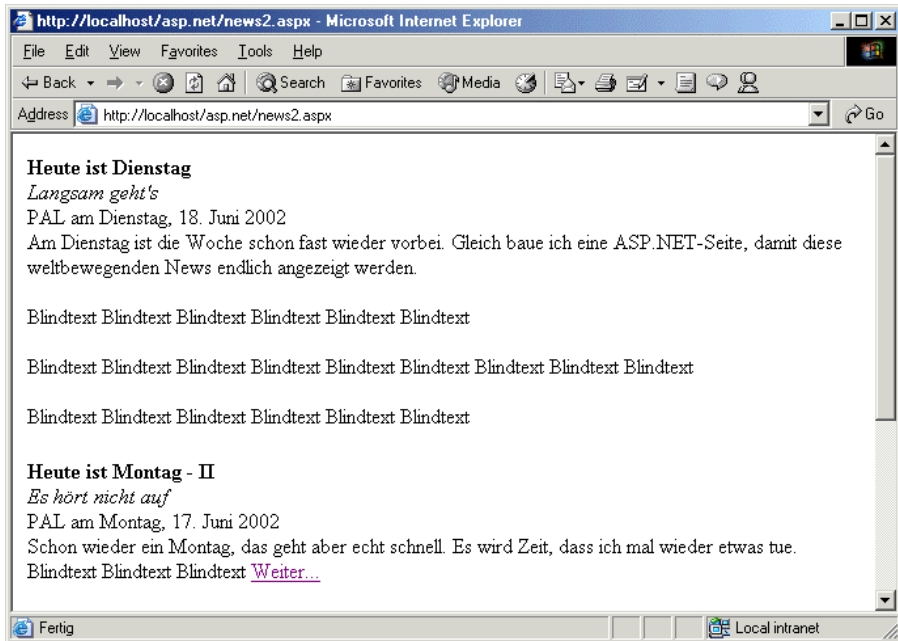


Abbildung 16.3 Die erste Neuigkeit ist selektiert.

Archivierung

Ich habe zu Beginn des Rezepts die Archivierung älterer Neuigkeiten erwähnt, die bisher noch keine Berücksichtigung fand. Es gibt zwei grobe Ansätze. Sie können zur Anzeige der alten Einträge entweder eine zweite Seite verwenden oder diese innerhalb der aktuellen anzeigen. Das dritte Listing zeigt eine Implementierung, bei der die alten Beiträge auf Knopfdruck innerhalb der aktuellen Seite aufgelistet werden. Über einen zweiten Button kann dieser Zustand wieder zurückgenommen werden.

Die aktuelle Ansicht wird über eine Eigenschaft `ShowAll` im `ViewState-StateBag` gespeichert. Bei der Abfrage der Daten wird die SQL-Query bei Bedarf mit einem `TOP 2` versehen, damit nur die letzten zwei Nachrichten ausgegeben werden. Ist die Option `ShowAll` hingegen gesetzt, so wird der Zusatz weggelassen, und alle Einträge werden aufgelistet.

Listing 16.3 news3.aspx

```
...

void BindDataList()
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = string.Format("SELECT {0}* FROM News ORDER BY Date
        DESC;", (this.ShowAll ? "" : "TOP 2 "));
    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    news.DataSource = cmd.ExecuteReader();
    DataBind();
    conn.Close();
}

...

bool ShowAll
{
    get
    {
        if(ViewState["ShowAll"] == null) return(false);
        return((bool) ViewState["ShowAll"]);
    }
    set { ViewState["ShowAll"] = value; }
}

void lb_ItemCommand(object sender, CommandEventArgs e)
{
    this.ShowAll = (e.CommandName == "showall");
    BindDataList();
}

</script>

<form runat="server">

    <p>
        <ASP:LinkButton runat="server"
            text="Auch alte Beiträge anzeigen"
            commandname="showall"
            OnCommand="lb_ItemCommand"
            visible="<%= !this.ShowAll %>" />

        <ASP:LinkButton runat="server"
```

```

text="Nur neue Beiträge anzeigen"
commandname="shownew"
OnCommand="lb_ItemCommand"
visible="<%# this.ShowAll %>"/>
</p>

<ASP:DataList id="news" runat="server"
...

```

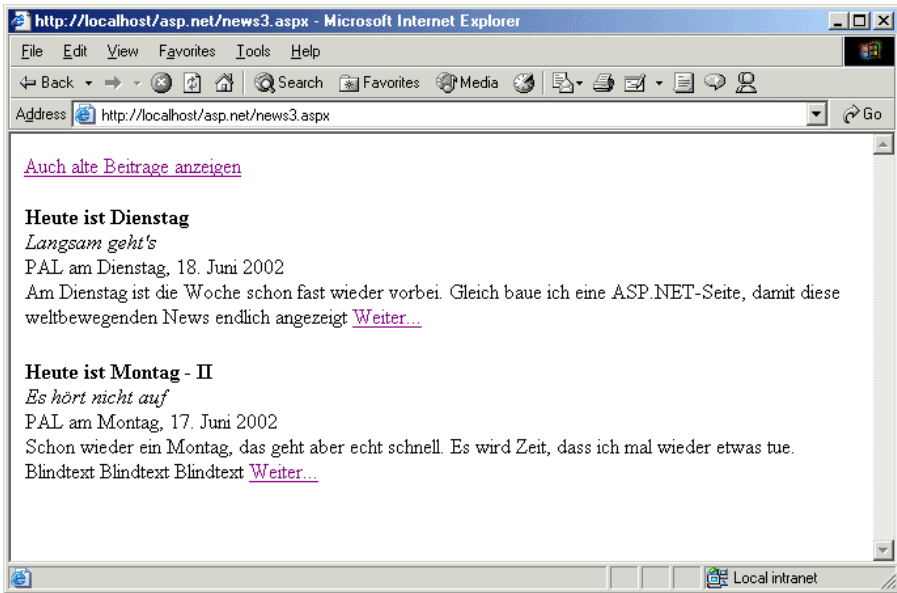


Abbildung 16.4 Ältere Beiträge werden nur bei Bedarf angezeigt.

16.2 ... neue Inhalte seit dem letzten Besuch markieren?

Gerade auf Seiten, die ständig aktualisiert werden, bietet sich eine besucherspezifische Markierung neuer Inhalte an. So sieht der Besucher auf einen Blick, welche Inhalte er vermutlich bereits kennt und welche noch nicht.

Ein solches besucherspezifisches System lässt sich am einfachsten mit Hilfe eines Cookies realisieren. Beim ersten Besuch wird der Cookie auf dem Client angelegt und mit dem aktuellen Datum versehen. Bei jedem weiteren Besuch wird der Cookie abgefragt und so das Datum des letzten Besuchs ermittelt. Zurückgesendet wird wieder das aktuelle Datum. Das Listing zeigt eine entsprechende Implementierung in Form einer C#-Quellcode-Datei.

Listing 16.4 CheckNew.cs

```
using System;
using System.Web;
using System.Web.SessionState;

public class CheckNew
{
    public static bool IsNew()
    {
        return(IsNew(DateTime.Now));
    }

    public static bool IsNew(DateTime dt)
    {
        const string lvstr = "LastVisit";

        HttpContext Context = HttpContext.Current;
        HttpRequest Request = Context.Request;
        HttpResponse Response = Context.Response;
        HttpSessionState Session = Context.Session;

        DateTime lastdt;

        if(Session[lvstr] == null)
        {
            HttpCookie cookie = Request.Cookies[lvstr];
            if(cookie != null)
            {
                lastdt = DateTime.Parse(cookie.Value);
            }
            else
                lastdt = DateTime.Now.AddDays(-7);

            Session[lvstr] = lastdt;
            Response.Cookies.Add(new HttpCookie(lvstr,
                DateTime.Now.ToString()));
        }
        else
            lastdt = (DateTime) Session[lvstr];

        return(dt > lastdt);
    }
}
```

Die statische Methode kümmert sich selbstständig um das komplette Handling der Cookies. Wird die Datei kompiliert und als DLL im bin-Verzeichnis der Web-Applikation abgelegt, kann die Methode innerhalb einer beliebigen Seite oder eines User Controls ohne explizite Einbindung aufgerufen werden. Das übergebene Datum wird auf Basis des übertragenen Cookies überprüft. Existiert noch kein Cookie, gelten alle innerhalb der letzten sieben Tage geänderten Inhalte als neu.

Ich habe die Funktionalität in einem kleinen User Control gekapselt. Dieses fragt das Datum der letzten Dateiänderung an der einbindenden Seite ab. Ist dieses jünger als der letzte Besuch, so wird über ein Label ein kleiner Hinweistext ausgegeben, ansonsten wird das Control nicht visualisiert.

Listing 16.5 CheckNew1.ascx

```
<% @Control Language="C#" Debug="true" %>

<script runat="server">

public DateTime ChangeDate
{
    get
    {
        if(ViewState["ChangeDate"] == null)
        {
            string file = Server.MapPath(Request.Path);
            return(System.IO.File.GetLastWriteTime(file));
        }
        return((DateTime) ViewState["ChangeDate"]);
    }
    set { ViewState["ChangeDate"] = value; }
}

public bool IsNew
{
    get { return(CheckNew.IsNew(this.ChangeDate)); }
}

protected override void OnPreRender(EventArgs e)
{
    lb.Visible = IsNew;
    base.OnPreRender(e);
}

</script>

<ASP:Label runat="server" id="lb">
    <p> <b> Achtung! </b> Diese Seite ist neu! </p>
</ASP:Label>
```

Dieses Listing zeigt die Verwendung des User Controls in einer Seite.

Listing 16.6 CheckNew1.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register TagPrefix="PAL" TagName="CheckNew"
    Src="CheckNew1.ascx" %>

<PAL:CheckNew runat="server"/>

Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
```

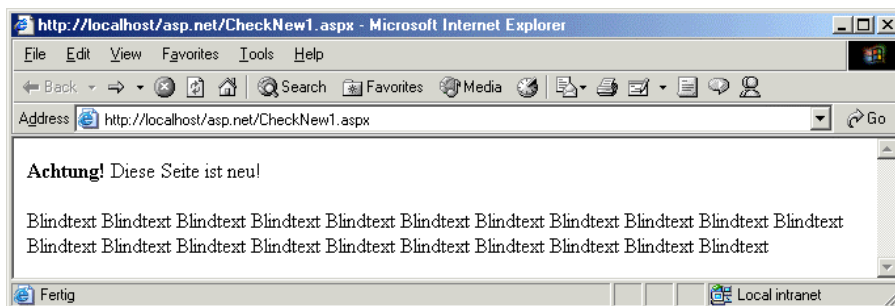


Abbildung 16.5 Die Seite wurde nach dem letzten Besuch geändert.

Um zu überprüfen, dass der Hinweis nicht bei jedem Aufruf ausgegeben wird, habe ich das Änderungsdatum als Eigenschaft gesetzt. Sie können dieses daher auch explizit von außen vorgeben. Im folgenden Fall wird die Seite bestimmt nicht mehr als neu proklamiert:

Listing 16.7 CheckNew2.aspx

```
<% @Page Language="C#" Debug="true" %>
<% @Register TagPrefix="PAL" TagName="CheckNew"
    Src="CheckNew1.ascx" %>

<PAL:CheckNew runat="server" ChangeDate="01.01.2002"/>

Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext Blindtext
```

Neben dieser eher impliziten Verwendung der statischen Hilfsmethode `CheckNew.IsNew` können Sie diese selbstverständlich auch an beliebiger Stelle Ihres Quellcodes verwenden ...

```
if (CheckNew.IsNew(DateTime.Now))  
{  
    ...  
}
```

... oder auch mit Hilfe der `DataBinding`-Syntax als Darstellungsbedingung für Server Controls nutzen:

Listing 16.8 CheckNew3.aspx

```
<script runat="server">  
  
void Page_Load(object sender, EventArgs e)  
{  
    DataBind();  
}  
  
</script>  
  
<ASP:PlaceHolder runat="server" visible="<%= CheckNew.IsNew() %>">  
  
<h1>Hallo!</h1>  
  
<p> Ja, genau Sie! Sie sind neu hier oder haben uns schon länger nicht  
besucht. Stimmt's?</p>  
  
<p> Das hier haben wir Neues für Sie:<br>  
    ...  
</p>  
  
</ASP:PlaceHolder>
```

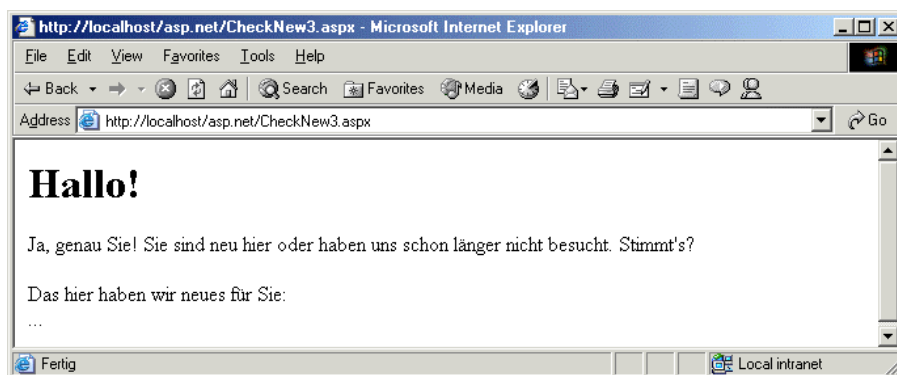


Abbildung 16.6 Die statische Methode wird als Darstellungsbedingung verwendet.

Sofern Sie das Datum der letzten Änderung in Verbindung mit einer SQL-Query verwenden möchten, rufen Sie zunächst die Methode `IsNew` auf und fragen anschließend die Session-Variable „LastVisit“ ab. Diese können Sie direkt als Parameter an eine Query übergeben:

```
...  
cmd.Parameters.Add("@LastVisit", Session["LastVisit"]);  
...
```

16.3 ... eine Newsletter-An- und Abmeldung realisieren?

Um einmal gewonnene Besucher immer wieder zu gewinnen und langfristig an ihr Angebot zu binden, bieten viele Betreiber Newsletter an. Ähnlich wie bei einem News-System innerhalb der Seite werden die potenziellen Wiederbesucher so über aktuelle Neuerungen informiert und mit hoffentlich interessanten Beiträgen versorgt.

Die Grundanforderungen an ein Newsletter-System ist die Möglichkeit zur einfachen An- und wieder Abmeldung. Die Daten werden in einer Datenbank hinterlegt, und der Betreiber muss lediglich beim Versand der Newsletter aktiv werden. Eher selten genutzt, aber durchaus sehr wirkungsvoll ist die Personalisierung von Newslettern, die deren Akzeptanz deutlich steigern kann. Im elementarsten Fall wird dies über eine persönliche Anrede erreicht, eine inhaltliche Verknüpfung mit einem hinterlegten Personenprofil ist jedoch ebenso denkbar.

Dieses Rezept zeigt die Erstellung eines einfachen Newsletter-Systems auf. Dabei wird zunächst das Frontend zum An- und Abmelden erstellt. Optional wird eine Überprüfung der angegebenen Email-Adresse möglich sein. Im zweiten Schritt zeige ich Ihnen ein kleines Backend zum Versenden der Newsletter, wobei auch eine einfache Personalisierung vorgenommen wird.

Datenbank

Im Grunde benötigen Sie für ein Newsletter-System lediglich ein Datenbankfeld zur Ablage der Email-Adresse. Da es Platz genug gibt, war ich aber großzügig und habe der Tabelle `Newsletter` ein paar Felder mehr spendiert. Neben einem Primärschlüssel gibt es Felder für Anrede, Vorname und Nachname des Abonnenten. Für die optional zu implementierende Überprüfung der Email-Adresse sind zwei weitere Felder vorhanden, die ich später beschreibe. Die Abbildung zeigt das Entwurfswindow der Tabelle.

	Feldname	Felddatentyp	Beschreibung
☒	ID	AutoWert	Eindeutige ID
☐	Email	Text	Email-Adresse, auch eindeutig
☐	Title	Text	Anrede
☐	Firstname	Text	Vorname
☐	Lastname	Text	Nachname
☐	AuthenticationCode	Text	Code zur optionalen Authentifizierung
☐	Authenticated	Ja/Nein	Gibt an, ob die Adresse bereits authentifiziert wurde

Feldeigenschaften	
Allgemein	Nachschlagen
Feldgröße	Long Integer
Neue Werte	Inkrement
Format	
Beschriftung	
Indiziert	Ja (Ohne Duplikate)

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Abbildung 16.7 Die Struktur der Tabelle Newsletter

Das Frontend

Die nächste Abbildung zeigt eine einfache Variante des zu erstellenden Frontends.



Abbildung 16.8 Eine einfache Anmeldung für den Newsletter

Nach Eingabe seiner persönlichen Daten reicht ein Mausklick aus, um sich für den Newsletter anzumelden. Zuvor prüfen allerdings insgesamt vier Validation-Controls die Eingaben des Benutzers. Es verwundert daher nicht, dass der Quelltext zum Beispiel ein wenig umfangreicher ist, als die eigentliche Funktionalität vermuten lässt.

Listing 16.9 Newsletter1.aspx

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "SELECT ID FROM Newsletter WHERE Email=@Email;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Email", tb_email.Text);
}
```

```

if(cmd.ExecuteScalar() != null)
{
    lb_msg.Text = "Sie haben den Newsletter bereits abonniert.";
}
else
{
    SQL = "INSERT INTO Newsletter (Title, Firstname, Lastname, Email)
    VALUES (@Title, @Firstname, @Lastname, @Email);";
    cmd = new OleDbCommand(SQL, conn);
    cmd.CommandText = SQL;
    cmd.Parameters.Add("@Title", rb_title.SelectedItem.Value);
    cmd.Parameters.Add("@Firstname", tb_fn.Text);
    cmd.Parameters.Add("@Lastname", tb_ln.Text);
    cmd.Parameters.Add("@Email", tb_email.Text);
    cmd.ExecuteNonQuery();
    lb_msg.Text = "Vielen Dank für Ihre Anmeldung!";
}

conn.Close();
}

</script>

<h1>Newsletter</h1>

<p> Vielen Dank für Ihr Interesse an unserem Newsletter! Bitte füllen
Sie die folgenden Felder aus, und klicken Sie auf "Absenden", um sich
anzumelden. </p>

<form runat="server">

    <p><b> <ASP:Label id="lb_msg" runat="server"/> </b></p>

    <table>

        <tr>
            <td>Anrede:</td>
            <td>
                <ASP:RadioButtonList id="rb_title" runat="server"
                RepeatColumns="2">
                    <ASP:ListItem text="Herr" selected="true"/>
                    <ASP:ListItem text="Frau"/>
                </ASP:RadioButtonList>
            </td>
        </tr>

        <tr>
            <td>Vorname:</td>

```

```

<td>
  <ASP:TextBox id="tb_fn" runat="server"/>
  <ASP:RequiredFieldValidator runat="server"
    ControlToValidate="tb_fn"
    ErrorMessage="Bitte geben Sie Ihren Vornamen an"
    Text="***/>
</td>
</tr>

<tr>
  <td>Nachname:</td>
  <td>
    <ASP:TextBox id="tb_ln" runat="server"/>
    <ASP:RequiredFieldValidator runat="server"
      ControlToValidate="tb_ln"
      ErrorMessage="Bitte geben Sie Ihren Nachnamen an"
      Text="***/>
  </td>
</tr>

<tr>
  <td>Email-Adresse:</td>
  <td>
    <ASP:TextBox id="tb_email" runat="server"/>
    <ASP:RequiredFieldValidator runat="server"
      Display="Dynamic"
      ControlToValidate="tb_email"
      ErrorMessage="Bitte geben Sie Ihre Email-Adresse an"
      Text="***/>
    <ASP:RegularExpressionValidator runat="server"
      ControlToValidate="tb_email"
      ValidationExpression="^[\\w\\.\\-]+@[\\w\\.\\-]+\\.\\s*"
      ErrorMessage="Bitte geben Sie Ihre korrekte Email-Adresse an"
      Text="***/>
  </td>
</tr>

</table>

<ASP:ValidationSummary runat="server"/>

<p>
  <ASP:Button runat="server" text="Absenden" onclick="bt_click"/>
</p>

</form>

```

Schenkt man dem Eingabeformular keine Beachtung, so reduziert sich die Seite auf die Behandlung des Button-Ereignisses. Hier wird zunächst überprüft, ob bereits ein Datensatz mit der angegebenen Email-Adresse existiert. Dies ist notwendig, da ein eindeutiger Index für dieses Tabellenfeld eingerichtet wurde und doppelte Einträge somit zu einem Fehler führen. Es hätte auch wenig Sinn, einen Newsletter doppelt zu versenden. Im Anschluss an eine negativ ausgefallene Überprüfung wird ein neuer Datensatz angelegt und eine Meldung im Browser ausgegeben. Das war's.

Abmeldung

Im gezeigten Frontend ist eine spätere Abmeldung nicht vorgesehen. Diesen Missstand soll die zweite Version beheben. Hier wurde ein weiterer Button eingefügt, dessen Ereignisbehandlung den Datensatz mit der angegebenen Email-Adresse löscht. Damit der Benutzer nicht unnötigerweise seine persönlichen Daten noch einmal angeben muss, wurde die Validierung für den zweiten Button deaktiviert.

Listing 16.10 Newsletter2.aspx

```
...

void unsubscribe_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "DELETE * FROM Newsletter WHERE Email=@Email;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Email", tb_email.Text);

    if(cmd.ExecuteNonQuery() == 1)
        lb_msg.Text = "Sie wurden erfolgreich abgemeldet.";
    else
        lb_msg.Text = "Die angegebene Email-Adresse wurde nicht gefunden.";

    conn.Close();
}

...

<ASP:Button runat="server" text="Abmelden"
    onclick="unsubscribe_click" CausesValidation="false"/>

...
```

Das Listing zeigt einen Trick. Die Methode `ExecuteNonQuery` liefert die Anzahl geänderter Datensätze zurück. Wenn der Wert 0 ist, wurde der entsprechende Datensatz offensichtlich nicht gefunden.

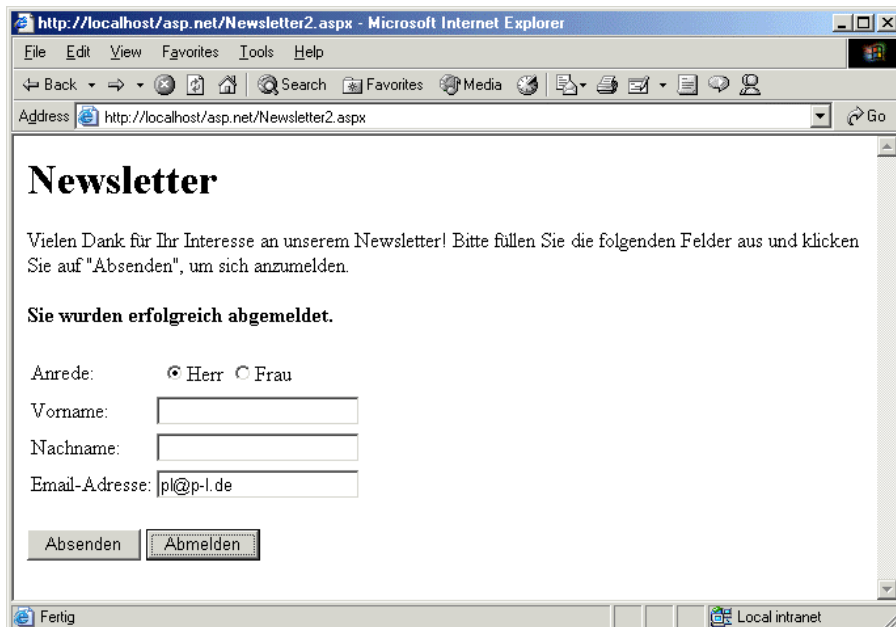


Abbildung 16.9 Die spätere Abmeldung ist genauso einfach wie die Anmeldung.



Um den Benutzern eine einfache Abmeldung zu ermöglichen, sollten Sie am Ende eines jeden versandten Newsletters auf diese Möglichkeit hinweisen. Als zusätzlichen Komfort können Sie einen personalisierten Link aufnehmen, der die Email-Adresse direkt in das entsprechende Feld einträgt:

```
http://localhost/asp.net/Newsletter2.aspx?email=pl@p-l.de
```

Die zur Auswertung notwendige Erweiterung des Beispiels sieht wie folgt aus:

```
void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack && Request.QueryString["email"] != null)
        tb_email.Text = Request.QueryString["email"];
}
```

Überprüfung der Email-Adresse

Mit Newslettern wird viel Schindluder getrieben. Oftmals weiß der Empfänger gar nichts von der Anmeldung und hält die Emails gar für Spam. Eine andere Quelle für fehlerhafte Datensätze sind Tippfehler des Benutzers. Beides vermindert die Qualität der Datenbankinhalte und erschwert deren Pflege.

Eine Überprüfung der Email-Adresse kann mitunter viel Arbeit und Ärger ersparen. Der neu angelegte Datensatz erhält dazu zunächst einen Zwischenstatus; die Anmeldung wird noch nicht freigeschaltet. An die angegebene Adresse wird eine Email mit einem speziellen Passwort geschickt. Nur mit diesem kann die Anmeldung tatsächlich aktiviert werden. Das System ist genauso einfach wie effektiv, denn falsche Adressangaben sind so ausgeschlossen.

Ich habe diese Möglichkeit bei der Anlage der Datenbanktabelle `Newsletter` bereits berücksichtigt. Dem Feld `AuthenticationCode` wird ein automatisch erzeugtes Passwort zugewiesen. Ich verwende eine Klasse aus dem Rezept „... ein Passwort erstellen?“ aus dem Kapitel „Sicherheit“. Das zweite boolesche Feld `Authenticated` gibt den Bestätigungsstatus an und wird nach Eingabe des Passworts auf `true` gesetzt.

Ich habe die Behandlung des Anmelde-Buttons erweitert. Nach der Anlage des Datensatzes wird hier nun eine Email generiert. Diese enthält unter anderem auch das zuvor in die Datenbank geschriebene Passwort. Die Generierung erfolgt auf Basis einer `MailParser`-Klasse aus dem Rezept „... eine Email-Vorlage erstellen?“ aus dem Kapitel „Eingabeformulare“. Über zwei `PlaceHolder-Controls` wird anschließend ein neues Formular zur Eingabe des Passwortes angezeigt.

Listing 16.11 Newsletter3.aspx – Anlegen des Datensatzes

```
void bt_click(object sender, EventArgs e)
{
    ...
    SQL = "INSERT INTO Newsletter (Title, Firstname, Lastname, Email,
        AuthenticationCode) VALUES (@Title, @Firstname, @Lastname, @Email,
        @AuthenticationCode)";
    cmd = new OleDbCommand(SQL, conn);
    cmd.CommandText = SQL;
    cmd.Parameters.Add("@Title", rb_title.SelectedItem.Value);
    cmd.Parameters.Add("@Firstname", tb_fn.Text);
    cmd.Parameters.Add("@Lastname", tb_ln.Text);
    cmd.Parameters.Add("@Email", tb_email.Text);
    cmd.Parameters.Add("@AuthenticationCode",
        PasswordGenerator.GetMnemonic(8));
    cmd.ExecuteNonQuery();

    string filename = Server.MapPath("confirm.txt");
```

```
NameValueCollection variables = new NameValueCollection();
foreach(OleDbParameter p in cmd.Parameters)
    variables.Add(p.ParameterName, p.Value.ToString());

MailMessageExt mail = new MailMessageExt(filename, variables);
SmtpMail.Send(mail);

ph1.Visible = false;
ph2.Visible = true;

tb_email2.Text = tb_email.Text;
}
```

Die Abbildung zeigt die erzeugte Email. Das dort angegebene Passwort muss in das zweite Formular eingegeben werden. Die Umschaltung erfolgt – wie beschrieben – über zwei Placeholder-Controls, die über die Visible-Eigenschaft gegensätzlich ein- beziehungsweise ausgeblendet werden.

Listing 16.12 Newsletter3.aspx – Formular zur Bestätigung

```
<ASP:Placeholder id="ph2" runat="server">

<p> Vielen Dank für Ihre Anmeldung. Um diese endgültig zu
    bestätigen, geben Sie bitte das Passwort ein, das Sie per Email
    erhalten haben. </p>

<p><b> <ASP:Label id="lb_msg2" runat="server"/> </b></p>

<table>

    <tr>
        <td>Email-Adresse:</td>
        <td> <ASP:TextBox id="tb_email2" runat="server"/> </td>
    </tr>

    <tr>
        <td>Bestätigungscode:</td>
        <td> <ASP:TextBox id="tb_code" runat="server"/> </td>
    </tr>

</table>

<p>
    <ASP:Button runat="server" text="Bestätigen"
        onclick="confirm_click"/>
</p>

</ASP:Placeholder>
```

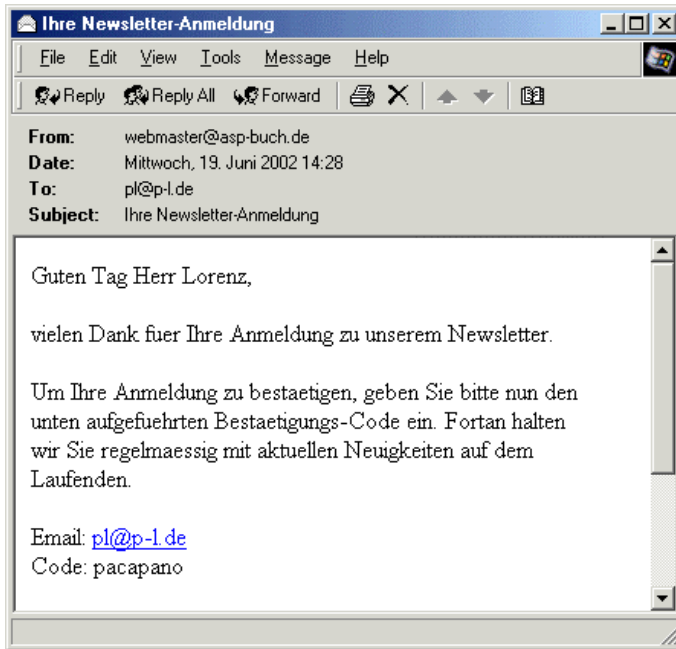



Abbildung 16.10 Die Email enthält das Passwort zur Freischaltung.

Das vom Benutzer eingegebene Passwort wird nach einem Klick auf den Bestätigen-Button überprüft. Ist es richtig, erfolgt die endgültige Freischaltung durch Setzen des Feldes `Authenticated`. Die Anmeldung war erfolgreich, die Email-Adresse ist überprüft.

Listing 16.13 Newsletter3.aspx – Überprüfen der Eingaben

```
void confirm_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "UPDATE Newsletter SET Authenticated=-1 WHERE
        Email=@Email AND AuthenticationCode=@AuthenticationCode;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Email", tb_email2.Text);
    cmd.Parameters.Add("@AuthenticationCode", tb_code.Text);

    if(cmd.ExecuteNonQuery() == 1)
        lb_msg2.Text = "Ihre Anmeldung wurde bestätigt, vielen Dank!";
    else
```

```
lb_msg2.Text = "Der eingegebene Code ist nicht korrekt, oder die  
Email-Adresse wurde nicht gefunden.";

conn.Close();
}
```

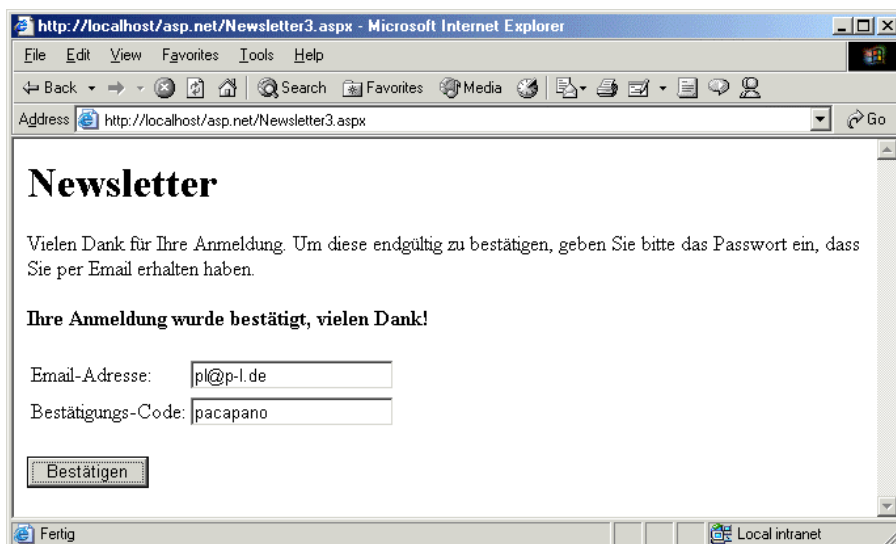


Abbildung 16.11 Erst nach Eingabe des Passworts wird die Anmeldung freigeschaltet.

Das Backend

Die Gestaltung des Backends zum Versand eines Newsletters gestaltet sich als recht einfach. Das Formular enthält drei Eingabefelder für Absender, Betreff und Text des Emailings. Hier können auch Platzhalter verwendet werden, die mit dem entsprechenden Inhalt der Datenbank ersetzt werden und so eine Personalisierung des Newsletters ermöglichen.

Listing 16.14 Backend1.aspx

```
<script runat="server">

void form_PreRender(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        HtmlForm form = (HtmlForm) sender;
```

```

        form.Attributes["onsubmit"] += "return(confirm('Sind Sie sicher,
            dass Sie fortfahren möchten?'))";
    }
}

void btsend_click(object sender, EventArgs e)
{
    ph1.Visible = false;
    Server.ScriptTimeout = 1000;
    Response.Write("<p><b> Der Newsletter wird versandt. Bitte warten!
        </b></p>");
    Response.Flush();

    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "SELECT * FROM Newsletter WHERE Authenticated=-1;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    OleDbDataReader reader = cmd.ExecuteReader();

    while(reader.Read())
    {
        MailMessage mail = new MailMessage();
        mail.To = (string) reader["Email"];
        mail.From = tb_from.Text;
        mail.Subject = tb_subject.Text;

        string body = tb_body.Text;
        for(int i=0; i<reader.FieldCount; i++)
            body = body.Replace("<" + reader.GetName(i) + ">",
                reader.GetValue(i).ToString());

        mail.Body = body;
        SmtpMail.Send(mail);

        Response.Write("&bull; " + reader["Email"] + "<br>");
        Response.Flush();
    }

    reader.Close();
    conn.Close();
}

</script>

<form runat="server" OnPreRender="form_PreRender">

```

```
<ASP:Placeholder id="ph1" runat="server">

<h1>Newsletter-Versand (Backend)</h1>

<p> Bitte geben Sie die notwendigen Daten zum Versand des
  Newsletters ein. Sie können die Platzhalter &lt;Title&gt;;,
  &lt;Firstname&gt;;, &lt;Lastname&gt;; und &lt;Email&gt;; verwenden.
</p>

<table>

  <tr>
    <td>Absender:</td>
    <td> <ASP:TextBox id="tb_from" runat="server"/> </td>
  </tr>

  <tr>
    <td>Betreff:</td>
    <td> <ASP:TextBox id="tb_subject" runat="server"/> </td>
  </tr>

  <tr>
    <td valign="top">Text:</td>
    <td>
      <ASP:TextBox id="tb_body" runat="server"
        TextMode="Multiline"
        Width="400"
        Height="200"/>
    </td>
  </tr>

</table>

<p>
  <ASP:Button runat="server" text="Absenden" onclick="btsend_click"/>
</p>

</ASP:Placeholder>

</form>
```

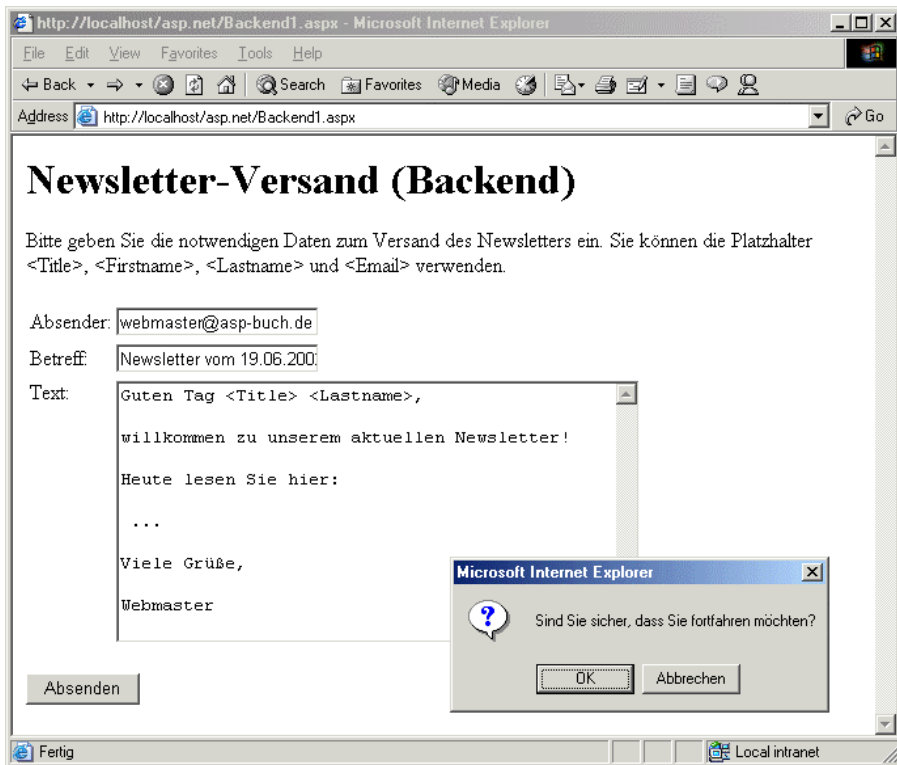


Abbildung 16.12 Das Backend ermöglicht das Versenden von Newslettern.

Ein Klick auf den Absende-Button fördert eine Sicherheitsabfrage zutage. Diese wurde entsprechend dem Rezept „... das Absenden eines Formulars bestätigen lassen?“ aus dem Kapitel „Eingabeformulare“ implementiert. Wird die Nachfrage bestätigt, erfolgt der Versand des Emailings. Per ADO.NET wird eine OleDbDataReader-Instanz auf alle bestätigten Anmeldungen „scharf“ gemacht.

In einer Schleife wird pro Datensatz eine Instanz der Klasse `MailMessage` angelegt und mit den Daten des Formulars ausgestattet. Bevor der Text zugewiesen wird, werden noch die dort hinterlegten Platzhalter mit den korrespondierenden Inhalten des aktuellen Datensatzes versehen. Fertig! Die Abbildung zeigt eine personalisierte Email.

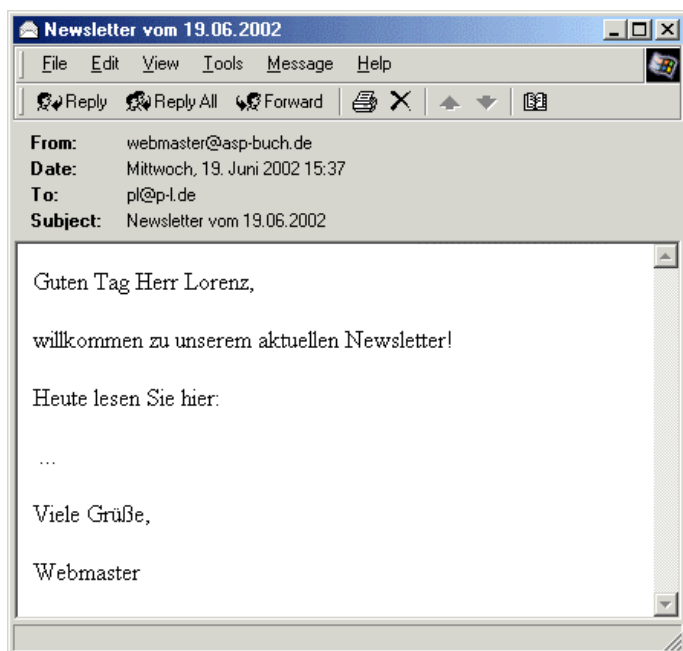


Abbildung 16.13 Diese Email wurde über das Backend erzeugt.

Das vorgestellte System bietet sich insbesondere zur Verwendung bei kleineren Datenmengen an. Bei einer größeren Anzahl von Empfängern sollte der Timeout-Wert des Scripts erhöht werden. Alternativ bietet sich die Verwendung einer spezialisierten Desktop-Software an, in die die Daten zuvor importiert werden. Aus eigener Erfahrung kann ich den combit address manager empfehlen. Informationen hierzu finden Sie unter folgender Adresse:

<http://www.combit.net>

Bevor Sie mit dem System in den Realbetrieb gehen, sollten Sie unbedingt noch eine Testfunktion implementieren, über die Sie den erstellten Newsletter vorab an eine eigene Adresse versenden können. Es empfiehlt sich, dass Sie eventuelle Tippfehler erkennen, bevor es Ihre Kunden tun. Meine Korrektorin, Frau Gottmann, weiß davon ein Lied zu singen.

Web Service

Zu guter Letzt möchte ich Ihnen noch vorstellen, wie Sie die gezeigte Funktionalität in einen Web Service kapseln können. Dies bietet sich durchaus an, denn so können Sie das System nicht nur in verschiedenen Webseiten (auf mehreren Servern) be-

nutzen, sondern auch Partner-Websites ermöglichen, Benutzer in Ihren Newsletter einzutragen. Selbstverständlich wird auch hier eine Überprüfung der Email-Adresse vorgenommen um einen Missbrauch zu verhindern. Ein zusätzlicher Schutz des Dienstes per Windows Authentication bietet sich an.

Listing 16.15 newsletter1.asmx

```
<% @WebService Language="C#" Debug="true" Class="Newsletter" %>

using System;
using System.Collections;
using System.Collections.Specialized;
using System.Data;
using System.Data.OleDb;
using System.Web.Mail;
using System.Web.Services;

public class Newsletter : WebService
{
    [WebMethod]
    public bool Subscribe(string title, string firstname, string
        lastname, string email)
    {
        OleDbConnection conn =
            new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
        conn.Open();

        string SQL = "SELECT ID FROM Newsletter WHERE Email=@Email;";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        cmd.Parameters.Add("@Email", email);

        bool ret = false;

        if(cmd.ExecuteScalar() == null)
        {
            SQL = "INSERT INTO Newsletter (Title, Firstname, Lastname, Email,
                AuthenticationCode) VALUES (@Title, @Firstname, @Lastname, @Email,
                @AuthenticationCode)";
            cmd = new OleDbCommand(SQL, conn);
            cmd.CommandText = SQL;
            cmd.Parameters.Add("@Title", title);
            cmd.Parameters.Add("@Firstname", firstname);
            cmd.Parameters.Add("@Lastname", lastname);
            cmd.Parameters.Add("@Email", email);
            cmd.Parameters.Add("@AuthenticationCode",
                PasswordGenerator.GetMnemonic(8));
            cmd.ExecuteNonQuery();
        }
    }
}
```

```
string filename = Server.MapPath("confirm.txt");

NameValueCollection variables = new NameValueCollection();
foreach(OleDbParameter p in cmd.Parameters)
    variables.Add(p.ParameterName, p.Value.ToString());

MailMessageExt mail = new MailMessageExt(filename, variables);
SmtpMail.Send(mail);

ret = true;
}

conn.Close();
return(ret);
}

[WebMethod]
public bool Confirm(string email, string code)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "UPDATE Newsletter SET Authenticated=-1 WHERE
        Email=@Email AND AuthenticationCode=@AuthenticationCode;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Email", email);
    cmd.Parameters.Add("@AuthenticationCode", code);

    bool ret = (cmd.ExecuteNonQuery() == 1);

    conn.Close();

    return(ret);
}

[WebMethod]
public bool Unsubscribe(string email)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "DELETE * FROM Newsletter WHERE Email=@Email;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Email", email);
```



```

bool ret = (cmd.ExecuteNonQuery() == 1);

conn.Close();

return(ret);
}
}

```

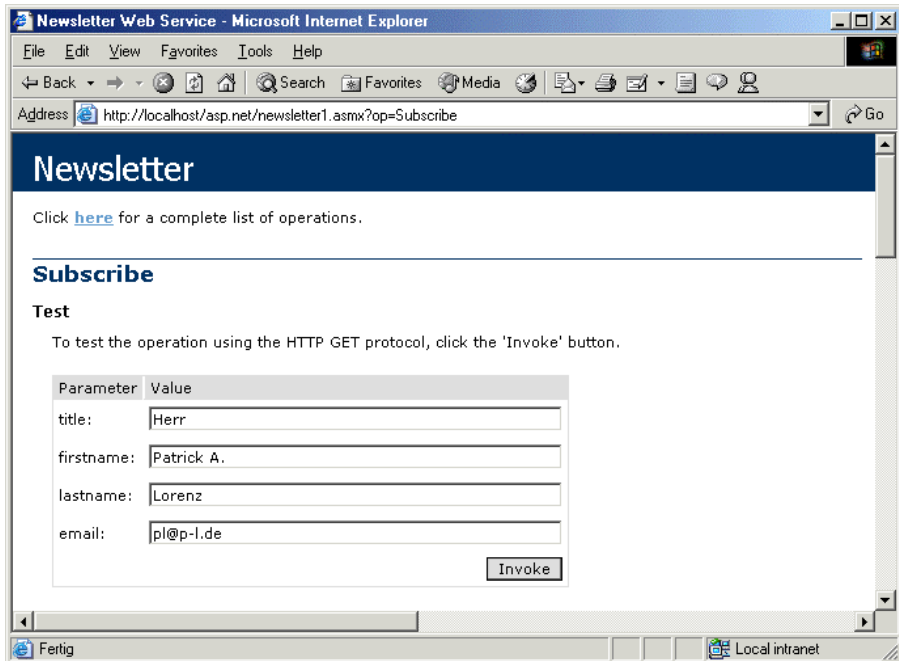


Abbildung 16.14 Auch über den Web Service ist eine Anmeldung möglich.

Der Web Service bietet drei Methoden an:

- `Subscribe` zum Anmelden an den Newsletter
- `Confirm` zum Bestätigen der Anmeldung
- `Unsubscribe` zum Abmelden

Die drei Methoden liefern allesamt einen booleschen Wert zurück, der über den Erfolg der gewünschten Aktion informiert.

16.4 ... eine Link-Sammlung erstellen?

Angebote wie das Open Directory Project (ODP) haben bewiesen, dass redaktionell betreute Verzeichnisdienste eine wichtige und sinnvolle Ergänzung zu den altbewährten Suchmaschinen wie Google darstellen. Viele Betreiber von Websites, Portalen und Communities bieten zudem eigene Verzeichnisse an, die ganz speziell auf die Zielgruppe zugeschnitten und durch einen kleineren Umfang deutlich übersichtlicher sind. Mit Hilfe von ASP.NET können Sie ein derartiges Link-System mit einer beliebig verschachtelten Kategoriehierarchie ruck, zuck erstellen. Dieses Rezept zeigt, wie das geht.

Datenbank

Natürlich soll auch das Link-Verzeichnis auf einer Datenbank basieren; keine Frage. Dabei gilt es zwischen der Kategorisierung und den eigentlichen Links zu differenzieren. Es müssen also zwei Tabellen `Links_Groups` und `Links` aufgenommen.

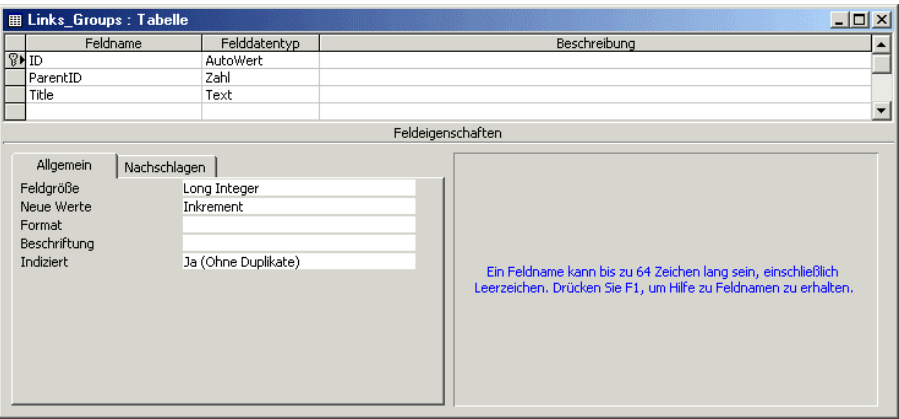


Abbildung 16.15 Die Struktur der Tabelle `Links_Groups`

In der ersten Tabelle `Links_Groups` sind die Kategorien des Verzeichnisses abgelegt, pro Datensatz eine. Ausgehend von der Hauptübersicht mit der ID 1 baut sich die Hierarchie auf. Dabei wird die jeweils übergeordnete Kategorie über das Feld `ParentID` referenziert. Es ergibt eine Quasi-Relation innerhalb einer Tabelle. Die Abbildung zeigt noch einmal das Entwurfsfenster.

Die zweite Tabelle enthält die eigentlichen Links. Neben dem Titel, der Beschreibung und der URL ist insbesondere die Zuordnung zu einer Gruppe über das Feld `GroupID` wichtig.

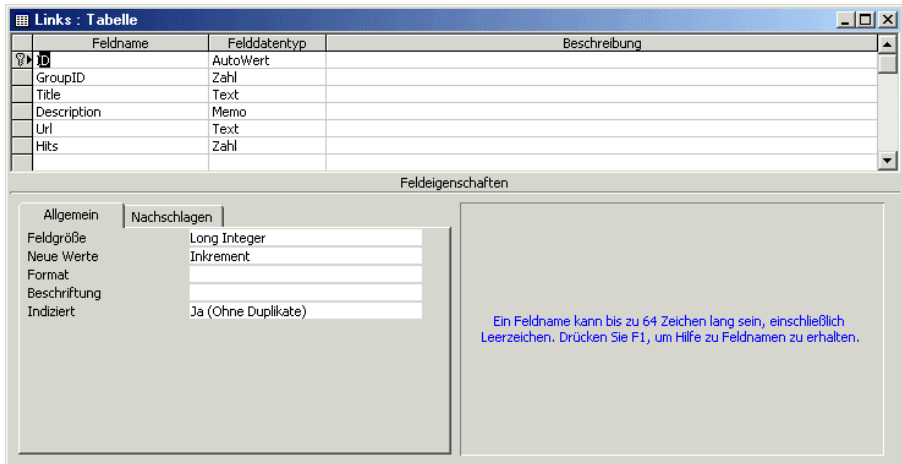


Abbildung 16.16 Die Struktur der Tabelle Links

Das gezeigte Feld `Hits` dient später als inkrementeller Zähler, der die Beliebtheit eines Links anzeigt.

Das Link-Verzeichnis

Nachdem die Datenbank erstellt ist, gilt es nun, das Verzeichnis selbst zu implementieren. An die ASP.NET-Seite werden folgende Anforderungen gestellt:

- Anzeige von (Unter-)Kategorien sowie Navigation
- Rücksprung zur nächsthöheren Kategorie
- Anzeige von Links sowie Aufruf und Zählung von Hits

Zur Anzeige bietet sich die Verwendung von zwei `DataList`-Controls an. Eine Methode übernimmt die Binding an die beiden unterschiedlichen Datenquellen. Dazu wird dieser die ID der aktuellen Kategorie übergeben. Die Controls enthalten jeweils `ItemTemplate`-Vorlagen mit einem `LinkButton`-Control zur Auswahl der Kategorie beziehungsweise des Links. Über die `ItemCommand`-Ereignisse wird die Benutzerinteraktion behandelt. Gegebenenfalls wird bei Auswahl einer neuen Kategorie die Datenbindung neu durchgeführt. That's it – bevor das Listing kommt, vorab ein Vorgeschmack auf das Ergebnis.



Abbildung 16.17 Die Hauptauswahl zeigt nur Kategorien an.

Listing 16.16 Links1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        BindData(1);
}

void BindData(int groupID)
{
    OleDbConnection conn1 =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn1.Open();

    OleDbConnection conn2 =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn2.Open();

    string SQL_group = string.Format("SELECT * FROM Links_Groups
        WHERE ID={0};", groupID);
    OleDbCommand cmd_group = new OleDbCommand(SQL_group, conn1);
    OleDbDataReader reader_group = cmd_group.ExecuteReader();
    reader_group.Read();

    lb_cat.Text = (string) reader_group["Title"];

    bt_back.Visible = (groupID != 1);
    bt_back.CommandArgument = reader_group["ParentID"].ToString();
    reader_group.Close();
}
```

```

string SQL1 = string.Format("SELECT * FROM Links_Groups
    WHERE ParentID={0};", groupID);
OleDbCommand cmd1 = new OleDbCommand(SQL1, conn1);
groups.DataSource = cmd1.ExecuteReader();

string SQL2 = string.Format("SELECT * FROM Links WHERE GroupID={0};",
    groupID);
OleDbCommand cmd2 = new OleDbCommand(SQL2, conn2);
links.DataSource = cmd2.ExecuteReader();

DataBind();
conn2.Close();
conn1.Close();
}

void groups_ItemCommand(object sender, DataListCommandEventArgs e)
{
    if(e.CommandName == "selectgroup")
        BindData(int.Parse(e.CommandArgument.ToString()));
}

void back_ItemCommand(object sender, CommandEventArgs e)
{
    if(e.CommandName == "selectgroup")
        BindData(int.Parse(e.CommandArgument.ToString()));
}

void links_ItemCommand(object sender, DataListCommandEventArgs e)
{
    if(e.CommandName == "selectlink")
    {
        OleDbConnection conn =
            new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=C:\inetpub\wwwroot\aspnet\community.mdb");
        conn.Open();

        int id = int.Parse(e.CommandArgument.ToString());

        string SQL = string.Format("UPDATE Links SET Hits=Hits+1 WHERE
            ID={0};", id);
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        cmd.ExecuteNonQuery();

        cmd.CommandText = string.Format("SELECT Url FROM Links WHERE
            ID={0};", id);
        string url = (string) cmd.ExecuteScalar();

        conn.Close();
    }
}

```

```
Response.Redirect(url);
}
}

</script>

<form runat="server">

<h1> Katetorie: <ASP:Label id="lb_cat" runat="server"/> </h1>

<p>
<ASP:LinkButton id="bt_back" runat="server"
Text="Zurück"
OnCommand="back_ItemCommand"
CommandName="selectgroup"/>
</p>

<ASP:DataList id="groups" runat="server"
RepeatColumns="2"
OnItemCommand="groups_ItemCommand">

<HeaderTemplate>
<h2>Bitte wählen Sie die gewünschte Unterkategorie aus</h2>
</HeaderTemplate>

<ItemTemplate>
<ASP:LinkButton runat="server"
Text=<%=# DataBinder.Eval(Container.DataItem, "Title") %>
CommandName="selectgroup"
CommandArgument=<%=# DataBinder.Eval(Container.DataItem, "ID") %>/>
</ItemTemplate>

</ASP:DataList>

<ASP:DataList id="links" runat="server"
RepeatColumns="2"
OnItemCommand="links_ItemCommand">

<HeaderTemplate>
<h2>Bitte wählen Sie den gewünschten Link aus</h2>
</HeaderTemplate>

<ItemTemplate>
<ASP:LinkButton runat="server"
Text=<%=# DataBinder.Eval(Container.DataItem, "Title") %>
CommandName="selectlink"
```

```

        CommandArgument=<%=# DataBinder.Eval(Container.DataItem, "ID") %>/>
        (<%=# DataBinder.Eval(Container.DataItem, "Hits") %> Hits)<br>
        <%=# DataBinder.Eval(Container.DataItem, "Description") %>
    </ItemTemplate>

</ASP:DataList>

</form>

```

Auch wenn es auf den ersten Blick nach einem langen Listing aussieht, vor dem Hintergrund der enthaltenen Funktionalität ist es eher kurz geraten; ASP.NET sei Dank.

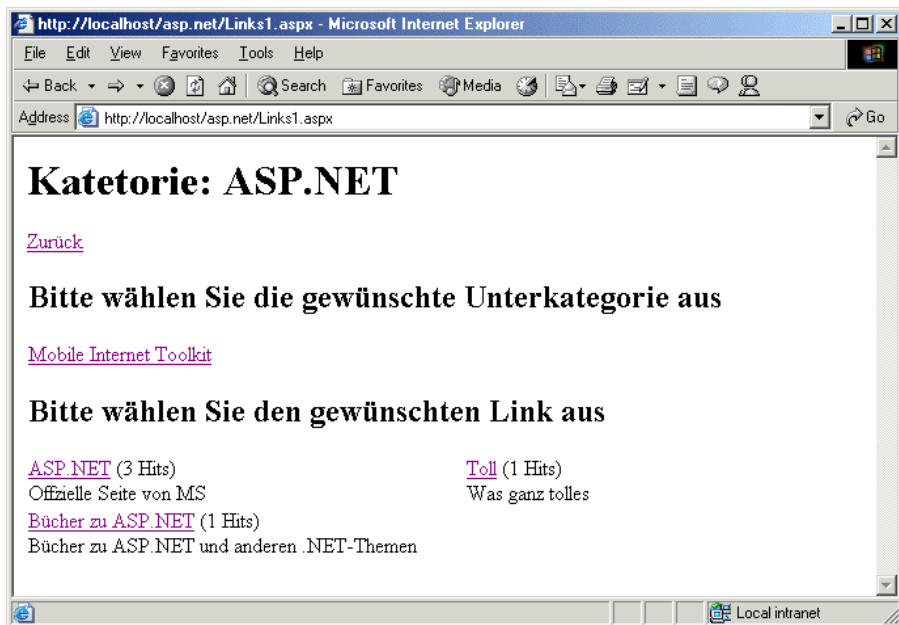


Abbildung 16.18 Nach der Auswahl werden die Unterkategorien und Links angezeigt.

Die Abbildung zeigt das Verzeichnis noch einmal im Einsatz. Innerhalb der Kategorienhierarchie wurde die Gruppe „ASP.NET“ ausgewählt. Es existieren eine Untergruppe „Mobile Internet Toolkit“ sowie eine Reihe von Links. Über diese können Sie zu den beschriebenen Seiten navigieren. Dabei erfolgt zunächst einPostBack zum Server, um den Link inkrementell zu zählen. Anschließend erfolgt ein Redirect. Wenn Sie die Zielseite in einem separaten Fenster anzeigen möchten, beachten Sie bitte das Rezept „... einen Redirect für einen anderen Frame durchführen?“ im Kapitel „Basics“.

16.5 ... einen zufälligen Link anzeigen?

Das im vorherigen Rezept gezeigte Link-Verzeichnis bringt mich auf eine weitere Idee, die in diesem Zusammenhang sicherlich des Öfteren realisiert werden soll: die Anzeige eines zufällig ausgewählten Links. Das Listing zeigt dies anhand der zuvor beschriebenen Datenbank sowie des Rezepts „... einen zufälligen Datensatz abfragen?“ aus dem Kapitel „Datenbanken“. Hierüber wird ein Datensatz aus der Tabelle `Links` zufällig ausgewählt und im Browser dargestellt. Über einen zusätzlichen Button kann der Benutzer einen neuen Link anfordern, sofern ihm der aktuelle nicht zusagt.

Listing 16.17 RandomLink1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        UpdateRandomLink();
}

void bt_Click(object sender, EventArgs e)
{
    UpdateRandomLink();
}

void UpdateRandomLink()
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "SELECT MIN(ID) FROM Links;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    int min = (int) cmd.ExecuteScalar();

    SQL = "SELECT MAX(ID) FROM Links;";
    cmd = new OleDbCommand(SQL, conn);
    int max = (int) cmd.ExecuteScalar();

    Random rnd = new Random();
    OleDbDataReader reader;

    do
    {
        int randomID = rnd.Next(min, max);
        SQL = string.Format("SELECT * FROM Links WHERE ID={0}", randomID);
```



```

cmd = new OleDbCommand(SQL, conn);
reader = cmd.ExecuteReader();
} while(!reader.Read());

hl.NavigateUrl = (string) reader["Url"];
hl.Text = (string) reader["Title"];
lb.Text = (string) reader["Description"];

conn.Close();
}

</script>

<h1> Zufallslink </h1>

<form runat="server">

  <p> Der folgende Link wurde zufällig aus unserem Link-Verzeichnis für
  Sie ausgewählt. </p>

  <p>
    <b> <ASP:Hyperlink id="hl" runat="server"/> </b> <br>
    <ASP:Label id="lb" runat="server"/>
  </p>

  <p> <ASP:Button runat="server" text="Neuer Link" onclick="bt_Click"/>

</form>

```

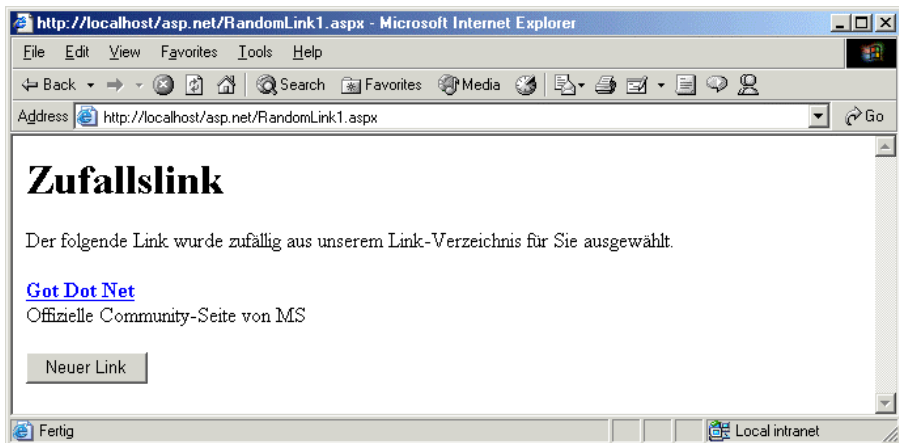


Abbildung 16.19 Bei jedem Aufruf wird ein zufälliger Link samt Beschreibung angezeigt.

16.6 ... eine elektronische Postkarte versenden?

Elektronische Postkarten oder auch E-Cards erfreuen sich großer Beliebtheit. Kein Wunder, sind sie doch für den Anbieter vergleichsweise einfach zu erstellen und für den Benutzer eine kurze, aber nette Abwechslung. Dieses Rezept zeigt, wie Sie mit einfachen Mitteln ein System für elektronische Postkarten entwerfen können. Das System wird aus drei Komponenten bestehen:

- Einer Datenbank im Hintergrund
- Einer Anlageseite für den Absender
- Einer Abfrageseite für den Empfänger

Datenbank

Vonseiten der Datenbank sind zwei Tabellen notwendig. Die erste mit dem Namen `ECards` enthält die Liste der möglichen Karten samt Titel und Dateinamen des anzuzeigenden Bildes.

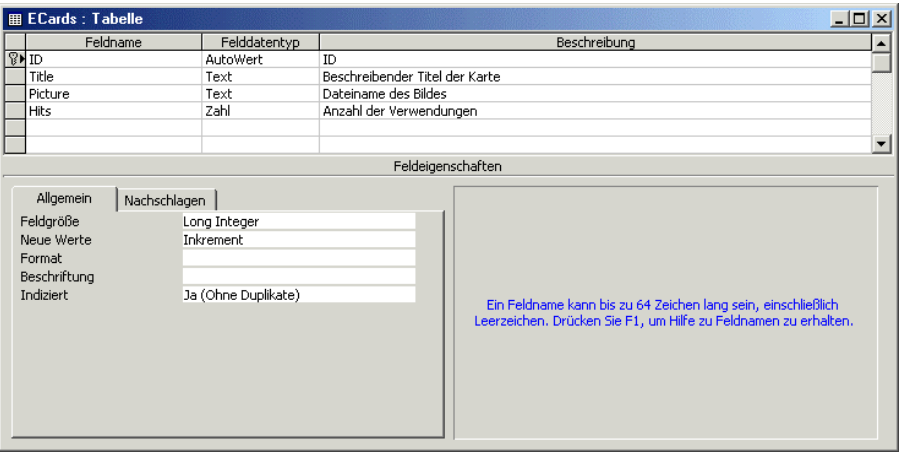


Abbildung 16.20 Die möglichen Kartenmotive werden in der Tabelle `ECards` hinterlegt.

In der zweiten Tabelle `ECards_Queue` werden die erstellten Karten zur Abholung bereitgehalten. Da zur späteren Identifizierung einer Karte die automatisch generierte ID verwendet werden soll, sollte diese zufällig und nicht *inkrementell* vergeben werden. Auf diese Weise lässt sich ein Missbrauch effektiv verhindern.

ECards_Queue : Tabelle			
	Feldname	Felddatentyp	Beschreibung
	ID	AutoWert	ID (Zufallswert)
	CardID	Zahl	ID der gewählten Karte
	SenderName	Text	Name des Absenders
	SenderEmail	Text	Email des Absenders
	RecipientName	Text	Name des Empfängers
	RecipientEmail	Text	Email des Empfängers
	Title	Text	Titel der Karte
	Text	Memo	Text der Karte

Feldeigenschaften	
Allgemein Nachschlagen Feldgröße: Long Integer Neue Werte: Zufall Format: Beschriftung: Indiziert: Ja (Ohne Duplikate)	Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Abbildung 16.21 Die Tabelle ECards_Queue nimmt die verfassten Karten auf.

Anlage der E-Card

Die Anlage einer neuen Grusskarte besteht aus zwei Schritten. Zunächst muss das gewünschte Motiv aus den hinterlegten Karten ausgesucht werden. Anschließend wird dieses personalisiert und mit den Daten des Empfängers versehen.

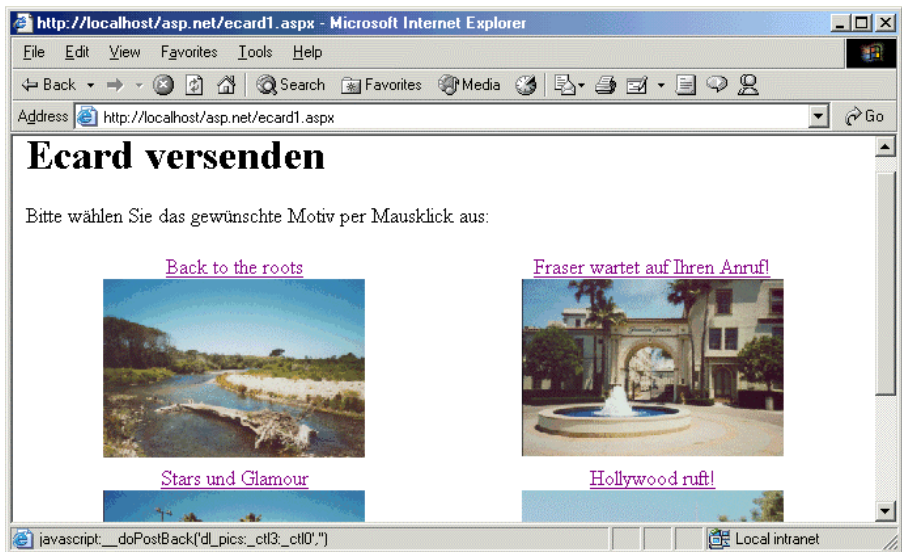


Abbildung 16.22 Die Auswahl des Kartenmotivs

Die Abbildung zeigt die Auswahl des später zu verwendenden Kartenmotivs. Ein Klick reicht aus, um zur zweiten Seite zu wechseln. Doch hier erst einmal das Listing, das dahinter steht.

Listing 16.18 ecard1.aspx – Auswahl des Motivs

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        OleDbConnection conn =
            new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
        conn.Open();

        string SQL = "SELECT * FROM ECards ORDER BY Hits DESC;";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        dl_pics.DataSource = cmd.ExecuteReader();
        dl_pics.DataBind();
        conn.Close();

        ph2.Visible = false;
        ph3.Visible = false;
    }
}

void dl_pic_ItemCommand(object sender, DataListCommandEventArgs e)
{
    if(e.CommandName == "selectimage")
    {
        int CardID = int.Parse(e.CommandArgument.ToString());
        ViewState["CardID"] = CardID;

        ph1.Visible = false;
        ph2.Visible = true;
    }
}

...

<h1> E-Card versenden </h1>

<form runat="server">

    <ASP:Placeholder id="ph1" runat="server">

        <p> Bitte wählen Sie das gewünschte Motiv per Mausklick aus: </p>
```

```

<ASP:DataList id="dl_pics" runat="server"
  OnItemCommand="dl_pic_ItemCommand"
  Width="100%"
  RepeatColumns="2">

  <ItemTemplate>
    <ASP:LinkButton runat="server"
      CommandName="selectimage"
      CommandArgument=<%=# DataBinder.Eval(Container.DataItem, "ID") %>>

    <p align="center">
      <%=# DataBinder.Eval(Container.DataItem, "Title") %> <br>
      <ASP:Image runat="server"
        ImageUrl=<%=# DataBinder.Eval(Container.DataItem, "Picture") %>
        Width="200"/>
    </p>

    </ASP:LinkButton>
  </ItemTemplate>

</ASP:DataList>

</ASP:Placeholder>

...

```

Beim Laden der Seite werden die verfügbaren Grußkarten ausgelesen. Über die Sortierung wird die Beliebtheit berücksichtigt. Die Anzeige erfolgt über ein gebundenes DataList-Control. Die Behandlung des enthaltenen LinkButton-Controls „merkt“ sich beim Klick auf eines der Bilder dessen ID. Die aktuelle Ansicht wird ausgeblendet und stattdessen eine zweite Formularseite zur Eingabe der persönlichen Daten angezeigt.

Listing 16.19 ecard1.aspx – Eingabe der persönlichen Daten

```

...

<ASP:Placeholder id="ph2" runat="server">

  <p> Bitte geben Sie nun Ihre persönlichen Daten, die des
    Empfängers und den gewünschten Kartentext ein. </p>

  <table>

    <tr>
      <td>Ihr Name:</td>
      <td> <ASP:TextBox id="tb_sendername" runat="server"/> </td>

```

```
</tr>

<tr>
  <td>Ihre Email-Adresse:</td>
  <td> <ASP:TextBox id="tb_senderemail" runat="server"/> </td>
</tr>

<tr>
  <td>Name des Empfängers:</td>
  <td> <ASP:TextBox id="tb_recipientname" runat="server"/> </td>
</tr>

<tr>
  <td>Email-Adresse des Empfängers:</td>
  <td> <ASP:TextBox id="tb_recipientemail" runat="server"/> </td>
</tr>

<tr>
  <td>Betreff:</td>
  <td> <ASP:TextBox id="tb_title" runat="server"/> </td>
</tr>

<tr>
  <td align="top">Grusstext:</td>
  <td>
    <ASP:TextBox id="tb_text" runat="server"
      TextMode="Multiline"
      Width="300"
      Height="100"/>
  </td>
</tr>

</table>

<p> <ASP:Button runat="server" text="Absenden"
  onclick="bt_submit_click"/>

</ASP:Placeholder>

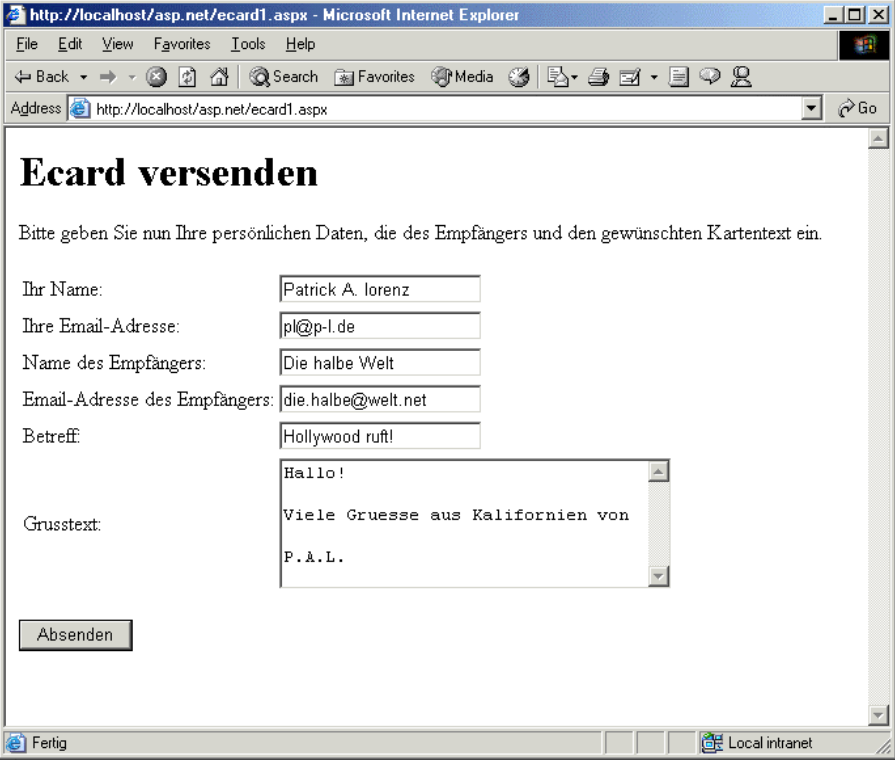
<ASP:Placeholder id="ph3" runat="server">

  <p><b> Vielen Dank! </b></p>
  <p> Ihre E-Card wurde gespeichert und der Empfänger per
    Email informiert.</p>

</ASP:Placeholder>

</form>
```

Das Formular verfügt aus Platzgründen bisher noch nicht über eine Validierung der Benutzereingaben. Diese sollte vor der weiteren Verwendung unbedingt noch nachgerüstet werden. Beispiele hierzu gibt es unter anderem im Rezept „... eine Newsletter-An- und -Abmeldung realisieren?“ weiter oben.



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/asp.net/ecard1.aspx`. The page title is "Ecard versenden". Below the title, a message reads: "Bitte geben Sie nun Ihre persönlichen Daten, die des Empfängers und den gewünschten Kartentext ein." The form contains several input fields: "Ihr Name:" with the value "Patrick A. lorenz", "Ihre Email-Adresse:" with "pl@p-l.de", "Name des Empfängers:" with "Die halbe Welt", "Email-Adresse des Empfängers:" with "die.halbe@welt.net", and "Betreff:" with "Hollywood ruft!". Below these is a text area for the message, containing "Hallo!" and "Viele Gruesse aus Kalifornien von P.A.L.". At the bottom left of the form is a button labeled "Absenden". The browser's status bar at the bottom shows "Fertig" and "Local intranet".

Abbildung 16.23 Im zweiten Schritt können die persönlichen Daten eingegeben werden.

Sind alle Daten eingegeben und die Grußkarte vollständig, kann diese über einen Button-Klick abgesendet werden. Es wird ein neuer Datensatz in der Tabelle `E-Cards_Queue` angelegt und mit den vom Benutzer eingegebenen Daten gefüllt. Auch die gewählte Karten-ID findet ihren Weg aus dem `ViewState-StateBag` in den Datensatz.

Anschließend wird die automatisch von der Engine vergebene ID abgefragt, wie im Rezept „... die automatische ID des zuletzt eingefügten Datensatzes abfragen?“ aus dem Kapitel „Datenbanken“ beschrieben. Zudem wird ein Passwort-Hashcode auf Basis der angegebenen Email-Adresse des Absenders erstellt. Diese Daten werden genutzt, um eine Email an den Empfänger der Karte zu senden. Wie sonst sollte dieser von deren Existenz erfahren? Verwendet wird der Mail-Parser aus dem Rezept „... eine Email-Vorlage erstellen?“, Kapitel „Eingabeformulare“.

Zu guter Letzt wird der Hit-Zähler des verwendeten Motivs inkrementiert, um die Beliebtheit bei der Anzeige der Karten auswerten zu können. Puh, das war's ...

Listing 16.20 ecard1.aspx – Absenden der Grußkarte

```
...

void bt_submit_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "INSERT INTO ECards_Queue(CardID, SenderName,
        SenderEmail, RecipientName, RecipientEmail, Title, [Text]) VALUES
        (@CardID, @SenderName, @SenderEmail, @RecipientName,
        @RecipientEmail, @Title, @Text);";

    OleDbCommand cmd = new OleDbCommand(SQL, conn);

    cmd.Parameters.Add("@CardID", (int) ViewState["CardID"]);
    cmd.Parameters.Add("@SenderName", tb_sendername.Text);
    cmd.Parameters.Add("@SenderEmail", tb_senderemail.Text);
    cmd.Parameters.Add("@RecipientName", tb_recipientname.Text);
    cmd.Parameters.Add("@RecipientEmail", tb_recipientemail.Text);
    cmd.Parameters.Add("@Title", tb_title.Text);
    cmd.Parameters.Add("@Text", tb_text.Text);
    cmd.ExecuteNonQuery();

    string filename = Server.MapPath("ecard.txt");

    NameValueCollection variables = new NameValueCollection();
    foreach(OleDbParameter p in cmd.Parameters)
        variables.Add(p.ParameterName, p.Value.ToString());

    SQL = "SELECT @@Identity;";
    cmd = new OleDbCommand(SQL, conn);

    variables.Add("@ID", cmd.ExecuteScalar().ToString());
    variables.Add("@Code", FormsAuthentication.
        HashPasswordForStoringInConfigFile(tb_senderemail.Text, "MD5"));

    MailMessageExt mail = new MailMessageExt(filename, variables);
    Smtplib.Send(mail);

    SQL = string.Format("UPDATE ECards SET Hits=Hits+1 WHERE ID={0};",
        ViewState["CardID"]);
```



```
new OleDbCommand(SQL, conn).ExecuteScalar();

conn.Close();

ph2.Visible = false;
ph3.Visible = true;
}

...
```

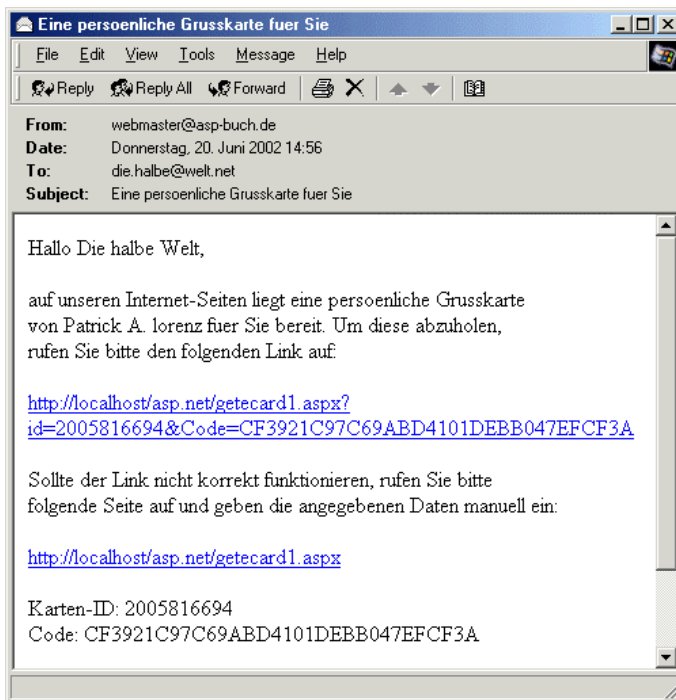


Abbildung 16.24 Der Empfänger wird per Email informiert.

Abfragen der E-Card

Hat der Empfänger die Email erhalten, will er natürlich seine Grußkarte einsehen. Die Abfrageseite enthält dazu zwei Placeholder-Controls. Das erste ist im Listing zu sehen und verfügt über ein Formular zur Eingabe der ID sowie des Codes aus der Email. Dies ist notwendig, da es je nach Email-Programm manchmal Probleme

bei längeren Links gibt. Alternativ kann der vorgegebene Link genutzt werden, vergleiche Abbildung. Werden die ID und der Code auf diese Weise an die Seite übertragen, werden die beiden Eingabefelder ausgefüllt und der Button ausgelöst.

Listing 16.21 getecard1.aspx – Eingabe der Abfragedaten

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        ph2.Visible = false;

    string id = Request.QueryString["id"];
    string code = Request.QueryString["code"];

    if(id != null && code != null)
    {
        tb_id.Text = id;
        tb_code.Text = code;
        bt_submit_click(null, EventArgs.Empty);
    }
}

...

<h1> E-Card abfragen </h1>

<form runat="server">

    <ASP:PlaceHolder id="ph1" runat="server">

        <p> Zur Abfrage einer Grusskarte geben Sie bitte die ID und den
            Code ein, den Sie per Email erhalten haben. </p>

        <p><b> <ASP:Label id="lb_msg" runat="server"/> </b></p>

        <table>
            <tr>
                <td>ID:</td>
                <td> <ASP:TextBox id="tb_id" runat="server"/> </td>
            </tr>
            <tr>
                <td>Code:</td>
                <td> <ASP:TextBox id="tb_code" runat="server" Width="300"/> </td>
            </tr>
        </table>
```

```

<asp:Button runat="server" text="Abfragen"
    onclick="bt_submit_click"/>

</asp:Placeholder>

```

Ein manueller oder „vorgetäuschter“ Klick auf den Button resultiert in einer Datenbankabfrage. Hier wird die zur ID passende Karte gesucht. Existiert diese, wird zur Sicherheit der eingegebene Hash-Code verglichen. Ist alles korrekt, werden die persönlichen Daten sowie das gewählte Bild ausgelesen und dem Benutzer über ein zweites Placeholder-Control angezeigt.

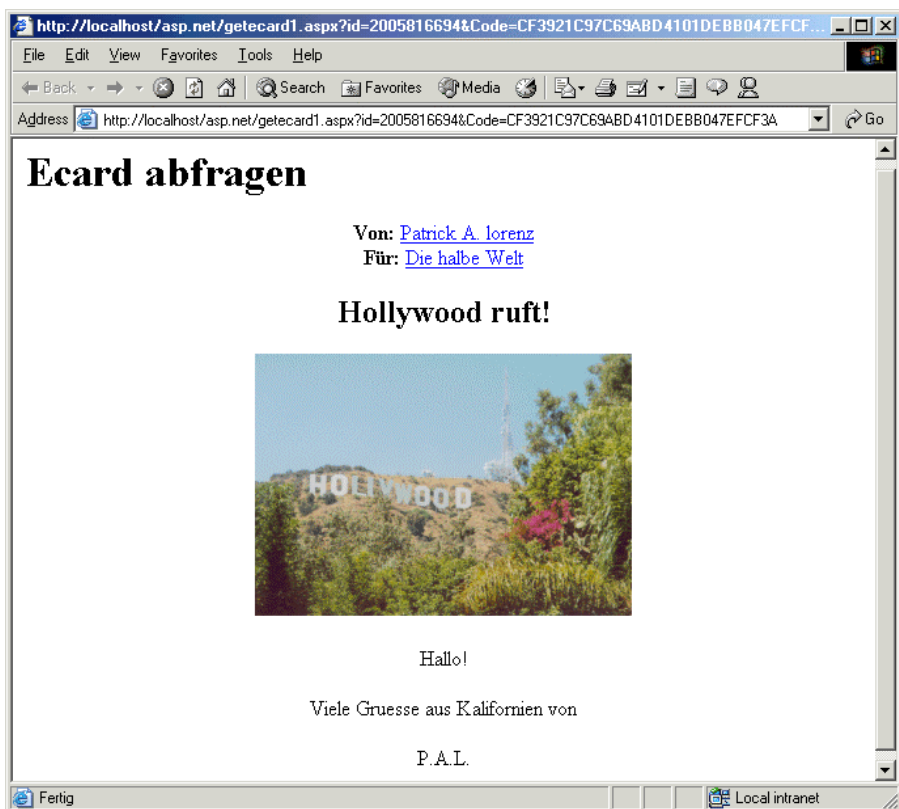


Abbildung 16.25 Die Grußkarte erstrahlt in vollem Glanz.

Listing 16.22 getecard1.aspx – Anzeige der Grusskarte

```
...

void bt_submit_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "SELECT * FROM ECards_Queue WHERE ID=@ID;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@ID", tb_id.Text);
    OleDbDataReader reader = cmd.ExecuteReader();

    if(reader.Read())
    {
        if(FormsAuthentication.HashPasswordForStoringInConfigFile((string)
            reader["SenderEmail"], "MD5") == tb_code.Text)
        {
            hl_sender.NavigateUrl = "mailto:" + reader["SenderEmail"];
            hl_sender.Text = (string) reader["SenderName"];

            hl_recipient.NavigateUrl = "mailto:" + reader["RecipientEmail"];
            hl_recipient.Text = (string) reader["RecipientName"];

            lb_title.Text = (string) reader["Title"];
            lb_text.Text = FormatText((string) reader["Text"]);

            string filename = Server.MapPath("getecard.txt");

            NameValueCollection variables = new NameValueCollection();
            for(int i=0; i<reader.FieldCount; i++)
                variables.Add("@ " + reader.GetName(i),
                    reader.GetValue(i).ToString());

            MailMessageExt mail = new MailMessageExt(filename, variables);
            SmtpMail.Send(mail);

            int CardID = (int) reader["CardID"];
            reader.Close();

            SQL = string.Format("SELECT Picture FROM ECards WHERE ID={0};",
                CardID);
            cmd = new OleDbCommand(SQL, conn);
            img.ImageUrl = (string) cmd.ExecuteScalar();

            ph1.Visible = false;
            ph2.Visible = true;
        }
    }
}
```

```

    }
    else
        lb_msg.Text = "Der eingegebene Code ist leider nicht richtig.";
    }
    else
        lb_msg.Text = "Die eingegebene Karte wurde leider nicht gefunden.";

    conn.Close();
}

string FormatText(string source)
{
    source = source.Replace("\r\n\r\n", "</p><p align=\"center\">");
    source = source.Replace("\r\n", "<br>");
    return(source);
}

...

<ASP:Placeholder id="ph2" runat="server">

    <p align="center">
        <b>Von:</b> <ASP:HyperLink id="hl_sender" runat="server"/> <br>
        <b>Für:</b> <ASP:HyperLink id="hl_recipient" runat="server"/>
    </p>

    <h2 align="center"> <ASP:Label id="lb_title" runat="server"/> </h2>

    <p align="center"> <ASP:Image id="img" runat="server"/> </p>

    <p align="center"> <ASP:Label id="lb_text" runat="server"/> </p>

</ASP:Placeholder>

</form>

```

Damit der Absender auch ja mitbekommt, dass seine Karte abgeholt wurde, erhält er automatisch eine Email, sobald der Empfänger die Karte anschaut. Auch hier ist wieder der bekannte Mail-Parser im Einsatz.

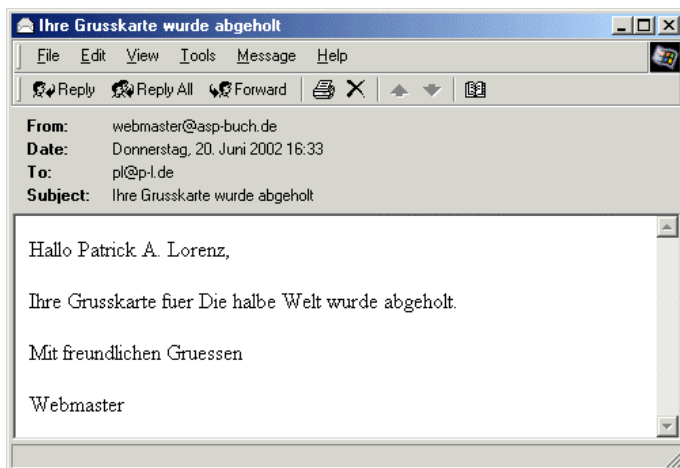


Abbildung 16.26 Der Absender wird automatisch informiert.

Schlussbemerkungen

Nachträglich ist mir eingefallen, dass eine Vorschaufunktion für die erstellte Grußkarte noch nett wäre. Na ja, die ist auf Basis der gelieferten Listings sicher in ein paar Minuten erstellt.

Darüber hinaus bietet es sich an, erstellte Karten nach einer gewissen Zeit – wie beispielsweise 14 Tage – zu löschen. Hierzu benötigen Sie lediglich ein Datumsfeld sowie einen Job, der beispielsweise einmal pro Tag alle alten Karten löscht. Der SQL-Server integriert eine solche Möglichkeit, bei Access können Sie ein kleines VBScript automatisch über den Task-Planer starten lassen.

16.7 ... ein Gästebuch erstellen?

Wenngleich ein Gästebuch nicht unbedingt zum gelungenen Inhalt einer Seite beiträgt, so hat es sich doch gerade bei kleineren Internet-Angeboten als nettes Goodie erwiesen. Mit ASP.NET ein eigenes Gästebuch zu erstellen, ist ziemlich einfach. Dieses Rezept verrät Ihnen, wie es geht. Dabei werden die drei wesentlichen Elemente vorgestellt:

- Die Datenbank als Basis
- Eine Seite zum Hinzufügen neuer Gästebucheinträge
- Eine Seite zum Anschauen des Gästebuches

Datenbank

Guestbook : Tabelle			
	Feldname	Felddatentyp	Beschreibung
ID		AutoWert	Eindeutige ID
Name		Text	Name des Besuchers
Email		Text	Email des Besuchers
Url		Text	Url des Besuchers
Text		Memo	Text des Eintrags
Date		Datum/Uhrzeit	Datum+Uhrzeit mit Standardwert

Feldeigenschaften	
Allgemein	Nachschlagen
Format	
Eingabeformat	
Beschriftung	
Standardwert	Jetzt()
Gültigkeitsregel	
Gültigkeitsmeldung	
Eingabe erforderlich	Nein
Indiziert	Nein

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Abbildung 16.27 Die Tabelle Guestbook

Die Abbildung zeigt die Tabelle, die dem Gästebuch als Datenspeicher dient. Sie ist relativ einfach aufgebaut und enthält im Wesentlichen Textfelder für die Angaben des Besuchers sowie den Text des Eintrags. Das Datumsfeld wird bei der Anlage automatisch mit der aktuellen Zeit versehen.

Hinzufügen eines neuen Eintrags

Das Hinzufügen eines neuen Gästebucheintrags erfolgt über ein einfaches Web-Formular. Für die gezeigten Datenbankfelder existieren hier korrespondierende Eingabefelder. Ein Button sorgt dafür, dass die Eingaben auf die übliche Weise in der Datenbank abgelegt werden. Über zwei Placeholder-Controls wird die Ansicht anschließend umgeschaltet.

Listing 16.23 gbadd1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        ph2.Visible = false;
}

void bt_submit_click(object sender, EventArgs e)
{

```

```
OleDbConnection conn =
    new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
        @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
conn.Open();

string SQL = "INSERT INTO Guestbook(Name, Email, Url, [Text])
    VALUES (@Name, @Email, @Url, @Text)";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
cmd.Parameters.Add("@Name", tb_name.Text);
cmd.Parameters.Add("@Email", tb_email.Text);
cmd.Parameters.Add("@Url", tb_url.Text);
cmd.Parameters.Add("@Text", tb_text.Text);
cmd.ExecuteNonQuery();

conn.Close();

ph1.Visible = false;
ph2.Visible = true;
}

</script>

<h1> Ins Gästebuch eintragen </h1>

<form runat="server">

    <p> Bitte geben Sie Ihre persönlichen Daten sowie
        Ihren Gästebuchtext ein. </p>

    <ASP:PlaceHolder id="ph1" runat="server">

        <table>

            <tr>
                <td>Ihr Name:</td>
                <td> <ASP:TextBox id="tb_name" runat="server"/> </td>
            </tr>

            <tr>
                <td>Ihre Email-Adresse:</td>
                <td> <ASP:TextBox id="tb_email" runat="server"/> </td>
            </tr>

            <tr>
                <td>Url:</td>
                <td> <ASP:TextBox id="tb_url" runat="server"/> </td>
            </tr>

            <tr>
```



```
<td align="top">Text:</td>
<td>
  <ASP:TextBox id="tb_text" runat="server"
    TextMode="Multiline"
    Width="300"
    Height="100"/>
</td>
</tr>

</table>

<p> <ASP:Button runat="server" text="Absenden"
  onclick="bt_submit_click"/>

</ASP:Placeholder>

<ASP:Placeholder id="ph2" runat="server">

  <p> Vielen Dank für Ihren Eintrag. Dieser ist
    sofort im Gästebuch sichtbar. </p>

  <p> <a href="gbshow1.aspx">Zum Gästebuch</a>... </p>

</ASP:Placeholder>

</form>
```

Aus Platzgründen wurden die üblichen Eingabevalidierungen im Listing nicht berücksichtigt. Diese lassen sich gegebenenfalls mit Hilfe der anderen Rezepte und Lösungen dieses Kapitels leicht nachrüsten.



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/asp.net/gbadd1.aspx`. The page title is "Ins Gästebuch eintragen". Below the title, a message reads: "Bitte geben Sie Ihre persönlichen Daten sowie Ihren Gästebuchtext ein." The form contains four input fields: "Ihr Name:" with the value "Patrick A. Lorenz", "Ihre Email-Adresse:" with the value "pl@p-l.de", "Url:" with the value "http://www.asp-buch.de", and a "Text:" area containing "Moin!", "Ich wollte nur mal 'Hallo' sagen.", and "Viele Gruesse,". A "Absenden" button is located at the bottom left of the form. The browser's status bar at the bottom shows "Fertig" and "Local intranet".

Abbildung 16.28 Das Anlegen eines neuen Eintrags geht ganz schnell.

Anzeige der Einträge

Was wäre ein Gästebuch ohne die Anzeige der eingetragenen Besucher-Statements. Im dritten und letzten Schritt möchte ich Ihnen die eigentliche Hauptseite vorstellen, die die vorhandenen Einträge dem interessierten Besucher präsentiert. Es handelt sich um eine einfache Abfrage der Tabelle `Guestbook` über einen `OleDbDataReader` sowie die Bindung an ein `DataList-Control`. Die Abfrage erfolgt mittels des `ORDER BY`-Befehls umgekehrt chronologisch, so dass der neueste Eintrag zuerst angezeigt wird.

Listing 16.24 gbshow1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        OleDbConnection conn =
            new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
```

```

conn.Open();

string SQL = "SELECT * FROM Guestbook ORDER BY Date DESC;";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
dl.DataSource = cmd.ExecuteReader();
DataBind();

conn.Close();
}
}

string FormatText(string source)
{
    source = source.Replace("\r\n\r\n", "</p><p>");
    source = source.Replace("\r\n", "<br>");
    return(source);
}

</script>

<h1> Gästebuch </h1>

<p> We proudly present our guestbook. Wollen Sie sich auch eintragen?
<a href="gbadd1.aspx">Hier geht's lang</a>! </p>

<ASP:DataList id="dl" runat="server"
    Width="100%"
    RepeatLayout="Flow">

    <ItemTemplate>

        <table border="1" width="100%">
            <tr>
                <td>
                    <b>
                        <ASP:HyperLink runat="server"
                            NavigateUrl=<## "mailto:" +
                                DataBinder.Eval(Container.DataItem, "Email") %>
                            Text=<## DataBinder.Eval(Container.DataItem, "Name") %>/></b>,

                        <ASP:HyperLink runat="server"
                            NavigateUrl=<## DataBinder.Eval(Container.DataItem, "Url") %>
                            Text=<## DataBinder.Eval(Container.DataItem, "Url") %>/> am

                        <## ((DateTime) DataBinder.Eval(Container.DataItem,
                            "Date")).ToString("d") %>
                    </td>
                </tr>
            <tr>

```

```
<td>
    <%# FormatText((string) DataBinder.Eval(Container.DataItem,
        "Text")) %>
</td>
</tr>
</table>

</ItemTemplate>

</ASP:DataList>
```

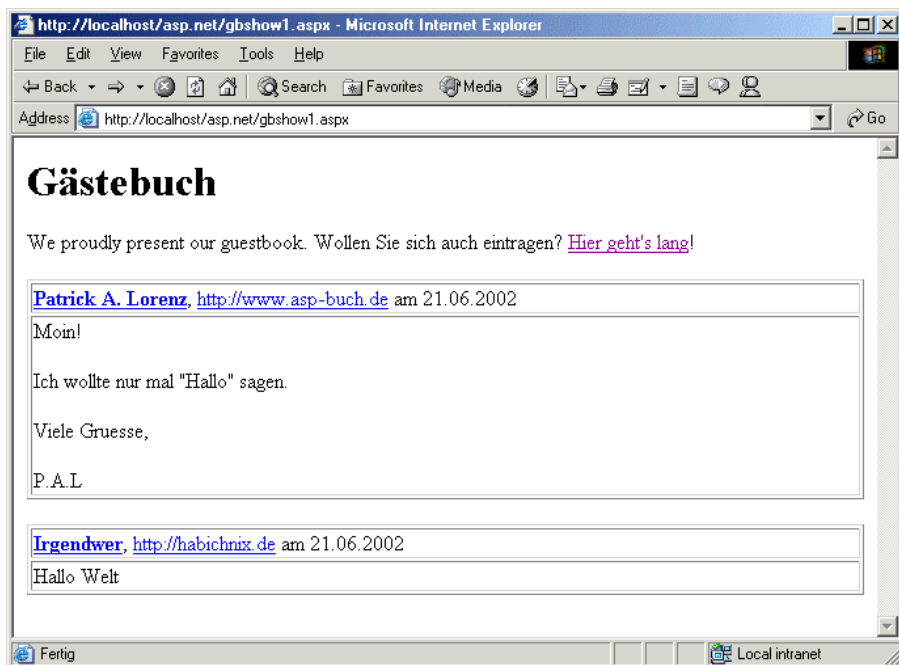


Abbildung 16.29 Alle Einträge werden umgekehrt chronologisch angezeigt.

Schlussbemerkungen

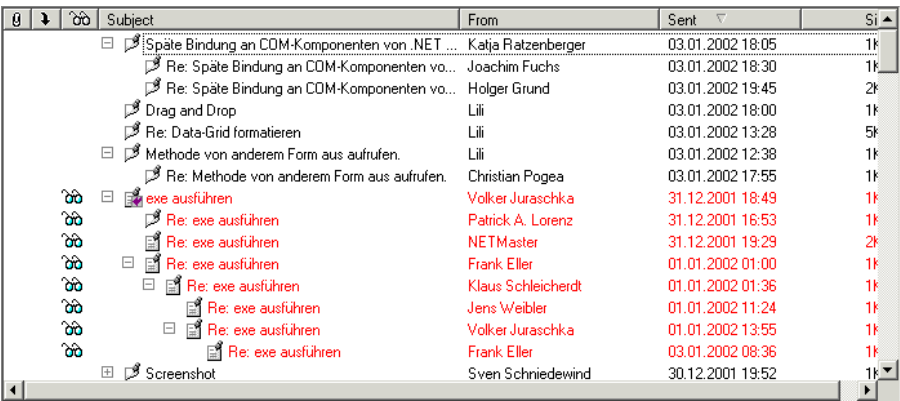
Bei der vorgestellten Implementierung nicht berücksichtigt wurden Features wie die Administration zum Löschen von Einträgen sowie zum Anfügen von Kommentaren des Betreibers. Auch könnte dieser zum Beispiel per Email über neue Einträge informiert werden.

16.8 ... ein Forum erstellen?

Ein interaktives Forum ist eine interessante Möglichkeit zum Austausch von Teilnehmern Ihrer Community untereinander. Üblicherweise können Benutzer Beiträge zur Verfügung stellen, und andere Benutzer antworten auf diese. So ergibt sich ein hierarchisches Diskussionsthema, genannt Thread. Das System ist mit einer Newsgroup vergleichbar.

Ein Forum zu entwickeln bedarf einiger Überlegungen. Zunächst gilt es, einen Blick auf die Anforderungen zu richten. Für eine minimale Lösung kommen auf den Entwickler immerhin folgende Aspekte zu:

- Schreiben von Beiträgen
- Betrachten einer Gesamtübersicht mit der Hierarchie der Threads
- Lesen eines Beitrags beziehungsweise eines Threads
- Antworten auf Beiträge



	Subject	From	Sent	Si
[-]	Späte Bindung an COM-Komponenten von .NET ...	Katja Ratzenberger	03.01.2002 18:05	1K
[+]	Re: Späte Bindung an COM-Komponenten vo...	Joachim Fuchs	03.01.2002 18:30	1K
[+]	Re: Späte Bindung an COM-Komponenten vo...	Holger Grund	03.01.2002 19:45	2K
[+]	Drag and Drop	Lili	03.01.2002 18:00	1K
[+]	Re: Data-Grid formatieren	Lili	03.01.2002 13:28	5K
[-]	Methode von anderem Form aus aufrufen.	Lili	03.01.2002 12:38	1K
[+]	Re: Methode von anderem Form aus aufrufen.	Christian Poguea	03.01.2002 17:55	1K
[+]	exe ausführen	Volker Juraschka	31.12.2001 18:49	1K
[+]	Re: exe ausführen	Patrick A. Lorenz	31.12.2001 16:53	1K
[+]	Re: exe ausführen	NETMaster	31.12.2001 19:29	2K
[-]	Re: exe ausführen	Frank Eller	01.01.2002 01:00	1K
[+]	Re: exe ausführen	Klaus Schleicherdt	01.01.2002 01:36	1K
[+]	Re: exe ausführen	Jens Weibler	01.01.2002 11:24	1K
[+]	Re: exe ausführen	Volker Juraschka	01.01.2002 13:55	1K
[+]	Re: exe ausführen	Frank Eller	03.01.2002 08:36	1K
[+]	Screenshot	Sven Schniedewind	30.12.2001 19:52	1K

Abbildung 16.30 Die Ansicht einer Newsgroup mit Outlook Express

Datenbank

Ein einzelnes Forum kommt mit einer Tabelle in der Community-Datenbank aus. Die öffentlichen Felder eines Beitrags sind schnell umrissen: drei Textfelder für Absendername, Email-Adresse sowie den Betreff des Beitrags und ein Memofeld für diesen selbst.

Das Problem kommt an einer anderen Stelle, nämlich der Anzeige der Thread-Übersicht. Hier sollen alle Threads umgekehrt chronologisch aufgelistet werden. Soweit kein Problem, doch es sollen natürlich auch die Hierarchie der Threads und somit die hinterlegten Antworten in der korrekten Reihenfolge angegeben werden.

Die Abbildung weiter oben zeigt eine Microsoft-Newsgroup in Outlook Express. Hier ist die notwendige Sortierung zu erkennen. Doch wie soll diese bitte mit SQL realisiert werden, ohne eine endlose Anzahl von Unterabfragen zu benötigen? Mit ein paar Tricks kein Problem.

Feldname	Felddatentyp	Beschreibung
ID	AutoWert	Eindeutige ID
ThreadID	Zahl	ID des Threads = eindeutige ID des ersten Beitrags
ParentID	Zahl	Eindeutige des übergeordneten Beitrags oder 0
ChildCount	Zahl	Anzahl der untergeordneten Beiträge
OrderID	Text	Feld zur Sortierung der Beiträge
Name	Text	Name des Autoren
Email	Text	Email-Adresse des Autoren
Subject	Text	Betreff des Beitrags
Text	Memo	Text des Beitrags
Date	Datum/Uhrzeit	Datum des Beitrags

Feldereigenschaften

Allgemein | Nachschlagen

Feldgröße: Long Integer
 Neue Werte: Inkrement
 Format:
 Beschriftung:
 Indiziert: Ja (Ohne Duplikate)

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Abbildung 16.31 Die Forumtabelle enthält zahlreiche interne Felder.

Die Abbildung zeigt die Struktur der Forumtabelle. Neben den beschriebenen öffentlichen Feldern sind insgesamt fünf interne Spalten zu erkennen. Sie stellen das Herzstück der Tabelle dar:

- ID ist die eindeutige ID des Beitrags
- ThreadID ist die eindeutige ID des obersten Beitrags im Thread
- ParentID ist die eindeutige ID des in der Hierarchie direkt übergeordneten Beitrags beziehungsweise 0 für den ersten Beitrag eines Threads
- ChildCount ist die Anzahl der untergeordneten Beiträge
- OrderID ist ein speziell zur Sortierung verwendetes Feld. Hier wird die Hierarchie in Form der IDs der übergeordneten Beiträge jeweils separiert mit einem Punkt hinterlegt.

Die Tabelle zeigt ein Beispiel für die Verwendung der Felder. Gegeben sind zwei Threads „Neuer Beitrag 1“ und neuer „Beitrag 2“. Sie haben die ThreadID 1 beziehungsweise 3. Diese ergibt sich aus der automatisch vergebenen ID des jeweils obersten Beitrags. Für den ersten Beitrag existieren zwei direkte Antworten (2 und

4). Zudem existiert eine Antwort auf Antwort 2. Die Hierarchie ist hier deutlich erkennbar. Widergespiegelt wird diese Hierarchie durch das Feld `OrderID`, in der die IDs der einzelnen übergeordneten Datensätze abgelegt sind.

Tabelle 16.1 Beispiel für die internen Felder mehrerer Beiträge

ID	T.ID	P.ID	Child	OrderID	Subject
3	3	0	1	3	Neuer Beitrag 2
6	3	3	0	3.1	Re: Neuer Beitrag 2
1	1	0	2	1	Neuer Beitrag 1
2	1	1	1	1.1	Re: Neuer Beitrag 1
5	1	2	0	1.1.1	Re: Re: Neuer Beitrag 1
4	1	1	0	1.2	Re: Neuer Beitrag 1

Die Eleganz dieser Variante liegt in zwei wesentlichen Punkten. Erstens ist die Sortierung der Beiträge für die Gesamtübersicht auf diese Weise trivial. Folgende Query reicht für die korrekte Sortierung aus:

```
SELECT * FROM Forum ORDER BY ThreadID DESC, OrderID
```

Zweitens muss beim Antworten auf einen Beitrag neben dem neu angelegten Datensatz nur ein weiterer bearbeitet werden. Dies ist der jeweils in der Hierarchie direkt übergeordnete Datensatz. Hier muss das Feld `ChildCount` inkrementiert werden.

Anlegen eines neues Beitrags/Threads

Die Anlage eines neuen Beitrags ist relativ einfach. Die ASP.NET-Seite enthält die benötigten Felder für Name, Email-Adresse, Betreff sowie den Beitrag selbst. Über einen Button werden die Eingaben in die Datenbank geschrieben. Nach der Neuanlage muss unbedingt die von der Engine automatisch vergebene `ID` abgefragt und den Feldern `ThreadID` und `OrderID` zugewiesen werden. Die beiden anderen Felder `ParentID` und `ChildCount` behalten ihren initiellen Wert 0.

Listing 16.25 forumadd1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        ph2.Visible = false;
}
```

```
void bt_submit_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "INSERT INTO Forum(Name, Email, Subject, [Text]) VALUES
        (@Name, @Email, @Subject, @Text)";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Name", tb_name.Text);
    cmd.Parameters.Add("@Email", tb_email.Text);
    cmd.Parameters.Add("@Subject", tb_subject.Text);
    cmd.Parameters.Add("@Text", tb_text.Text);
    cmd.ExecuteNonQuery();

    SQL = "SELECT @@Identity;";
    cmd = new OleDbCommand(SQL, conn);
    int id = (int) cmd.ExecuteScalar();

    SQL = string.Format("UPDATE Forum SET ThreadID={0}, OrderID='{0}'
        WHERE ID={0};", id);
    cmd = new OleDbCommand(SQL, conn);
    cmd.ExecuteNonQuery();

    conn.Close();

    ph1.Visible = false;
    ph2.Visible = true;
}
</script>

<h1> Neuen Beitrag schreiben </h1>

<form runat="server">

    <p> Bitte geben Sie Ihre persönlichen Daten sowie Betreff
        und Text des Beitrags ein. </p>

    <ASP:Placeholder id="ph1" runat="server">

        <table>

            <tr>
                <td>Ihr Name:</td>
                <td> <ASP:TextBox id="tb_name" runat="server"/> </td>
            </tr>
```



```
<tr>
  <td>Ihre Email-Adresse:</td>
  <td> <ASP:TextBox id="tb_email" runat="server"/> </td>
</tr>

<tr>
  <td>Betreff:</td>
  <td> <ASP:TextBox id="tb_subject" runat="server"/> </td>
</tr>

<tr>
  <td valign="top">Text:</td>
  <td>
    <ASP:TextBox id="tb_text" runat="server"
      TextMode="Multiline"
      Width="300"
      Height="100"/>
  </td>
</tr>

</table>

<p> <ASP:Button runat="server" text="Absenden"
  onclick="bt_submit_click"/> </p>

</ASP:Placeholder>

<ASP:Placeholder id="ph2" runat="server">

  <p> Vielen Dank für Ihren Beitrag. Dieser ist sofort
    im Forum sichtbar. </p>

  <p> <a href="forumshow1.aspx">Zum Forum</a>... </p>

</ASP:Placeholder>

</form>
```

http://localhost/asp.net/forumadd1.aspx - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Search Favorites Media Print Mail News Groups

Address http://localhost/asp.net/forumadd1.aspx Go

Neuen Beitrag schreiben

Bitte geben Sie Ihre persönlichen Daten sowie Betreff und Text des Beitrags ein.

Ihr Name:

Ihre Email-Adresse:

Betreff:

Text:

Moin!

Das ist der erste Beitrag in diesem ach so neuen Forum.

Viel Spaß,

Fertig Local intranet

Abbildung 16.32 Die Eingabe eines neuen Beitrags ist ganz einfach.

Anzeige der Thread-Übersicht

Die Hauptanzeige der Threads und Beiträge ist sicherlich ein interessanter Aspekt eines Forums. Hier gilt es, die vorhandenen Themen übersichtlich und in der korrekten Hierarchiefolge zu präsentieren. Die dazu notwendigen Grundlagen habe ich bereits weiter oben im Zusammenhang mit der Datenbankbeschreibung erklärt. Bei der Anzeige lassen sich die dort vorgestellten Techniken konkret anwenden. Diese beruht auf einem DataGrid-Control und einer einfachen SQL-Query:

```
SELECT * FROM Forum ORDER BY ThreadID DESC, OrderID;
```

Die so gewonnenen Daten werden an das DataGrid gebunden. Hier existieren drei Spalten für Betreff, Autor und Datum. Die beiden ersten werden über eine TemplateColumn-Spalte angelegt. Um eine Übersicht zu garantieren, sollen hierarchisch untergeordnete Beiträge wie im Newsreader eingerückt erscheinen. Hierzu werden die im Feld `OrderID` enthaltenen Punkte gezählt und entsprechend viele unbedingte Leerzeichen (HTML-Sonderzeichen ` `) ausgegeben. Grundlage bilden die beiden Rezepte „... das Vorkommen eines Zeichens in einer Zeichenkette zählen?“ und „... eine Zeichenkette aus wiederholenden Zeichen/Zeichenketten zusammensetzen?“ aus dem Kapitel „Zeichenketten“.

Listing 16.26 forumshow1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        OleDbConnection conn =
            new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
        conn.Open();

        string SQL = "SELECT * FROM Forum ORDER BY ThreadID DESC, OrderID;";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        dg.DataSource = cmd.ExecuteReader();
        DataBind();

        conn.Close();
    }
}

int CountChar(string str, char c)
{
    int count = 0;
    for(int i=0; i<str.Length; i++)
        if(str[i] == c) count++;
    return(count);
}

string RepeatString(string str, int count)
{
    string result = string.Empty;

    for(int i=0; i<count; i++)
        result += str;

    return(result);
}

</script>

<h1> Forum </h1>

<p>
    Bitte wählen Sie den gewünschten Beitrag per Mausklick aus.<br>
    Sie können auch einen <a href="forumadd1.aspx">neuen Beitrag</a>
    verfassen.
</p>
```

```
<ASP:DataGrid id="dg" runat="server"
AutoGenerateColumns="false"
Width="100%">

<Columns>

<ASP:TemplateColumn
HeaderText="Betreff"
ItemStyle-Width="60%">

<ItemTemplate>
<%# RepeatString("&nbsp;", CountChar((string)
DataBinder.Eval(Container.DataItem, "OrderID"), '.')) %>
<ASP:Hyperlink runat="server"
Text=<%# DataBinder.Eval(Container.DataItem, "Subject") %>
NavigateUrl=<%# "forumshowarticle1.aspx?id=" +
DataBinder.Eval(Container.DataItem, "ID") %>/>
</ItemTemplate>

</ASP:TemplateColumn>

<ASP:TemplateColumn
HeaderText="Autor"
ItemStyle-Width="20%">

<ItemTemplate>
<ASP:Hyperlink runat="server"
Text=<%# DataBinder.Eval(Container.DataItem, "Name") %>
NavigateUrl=<%# "mailto:" + DataBinder.Eval(Container.DataItem,
"Email") %>/>
</ItemTemplate>

</ASP:TemplateColumn>

<ASP:BoundColumn
HeaderText="Datum"
DataField="Date"
DataFormatString="{0:d}"
ItemStyle-Width="20%" />

</Columns>

</ASP:DataGrid>
```

Um der Abbildung ein wenig mehr Leben einzuhauchen, habe ich über die Datenbank zwei weitere Beiträge angelegt. Es handelt sich um einen neuen Thread sowie eine Antwort auf den ersten Beitrag. Die Abbildung zeigt nun die korrekte Anordnung der Beiträge und der Threads. Die Antwort ist dabei zur Kennzeichnung der Hierarchie leicht eingerückt.

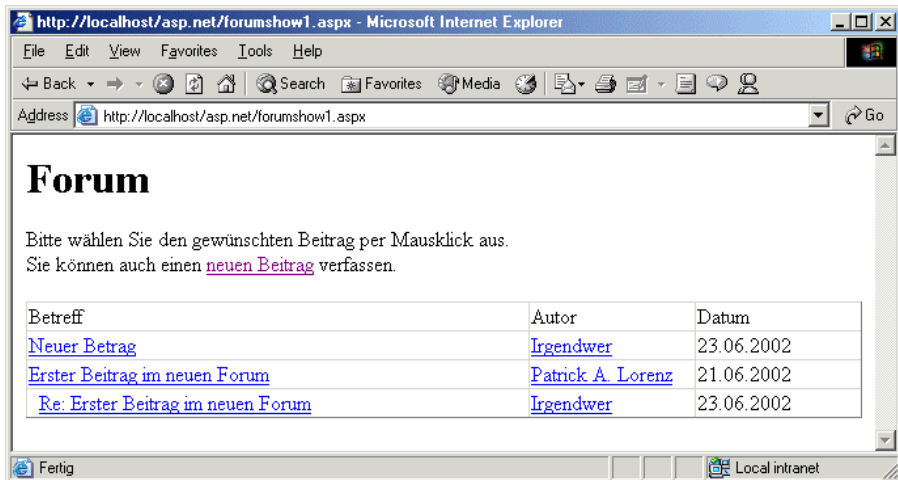


Abbildung 16.33 Die Seite zeigt eine Übersicht aller Threads und Beiträge.

Die Betreffspalte besteht aus einem Link. Über diesen kann auf die Seite `forum-showarticle1.aspx` zugegriffen werden, die einen ausgewählten Beitrag anzeigt. Dessen ID wird über den Query-String weitergegeben.

Anzeige eines einzelnen Beitrags

Die Anzeige eines einzelnen Beitrags ist ausgesprochen einfach zu realisieren. Vier Label-Controls nehmen die öffentlichen Felder des korrespondierenden Datensatzes auf. Über ein HyperLink-Control kann auf den Beitrag geantwortet werden. Hierzu wird die ID des aktuellen Beitrags im Query-String an die Seite `forumadd1.aspx` übergeben.

Listing 16.27 forumadd1.aspx

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
```

```
OleDbConnection conn =
    new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
        @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
conn.Open();

string SQL = "SELECT * FROM Forum WHERE ID=@ID;";
OleDbCommand cmd = new OleDbCommand(SQL, conn);
cmd.Parameters.Add("@ID", Request.QueryString["id"]);
OleDbDataReader reader = cmd.ExecuteReader();

if(reader.Read())
{
    lb_subject.Text = (string) reader["Subject"];
    hl_author.Text = (string) reader["Name"];
    hl_author.NavigateUrl = "mailto:" + reader["Email"];
    lb_date.Text = ((DateTime) reader["Date"]).ToString("d");
    lb_text.Text = FormatText((string) reader["Text"]);
    hl_reply.NavigateUrl = "forumadd1.aspx?reply=" + reader["ID"];
}

conn.Close();
}
}

string FormatText(string source)
{
    source = source.Replace("\r\n\r\n", "</p><p>");
    source = source.Replace("\r\n", "<br>");
    return(source);
}

</script>

<h1> Forum </h1>

<h2> <ASP:Label id="lb_subject" runat="server"/> </h2>

<p> <ASP:HyperLink id="hl_author" runat="server"/>
    am <ASP:Label id="lb_date" runat="server"/> </p>

<p> <ASP:Label id="lb_text" runat="server"/> </p>

<p> <ASP:HyperLink id="hl_reply" runat="server" text="Antworten"/>
</p>
```



Abbildung 16.34 Die Seite zeigt einen einzelnen Beitrag an.

Antworten auf einen Beitrag

Die Seite `forumadd1.aspx` habe ich Ihnen schon weiter oben im Zuge der Neuanlage eines Beitrags vorgestellt. Die Beantwortung eines existierenden Beitrags ist im Grunde nichts anderes, denn auch hier wird ein neuer Datensatz mit den entsprechenden Benutzereingaben angelegt. Zusätzlich muss eine Verknüpfung mit dem bestehenden Beitrag angelegt werden. Aufgrund dieser Ähnlichkeiten bietet sich die Implementierung innerhalb einer Seite mehr als an. Ich habe die oben gezeigte Version daher erweitert.

Die Änderungen müssen an zwei Stellen vorgenommen werden. Zunächst muss beim Öffnen der Seite der übergeordnete Beitrag abgefragt und die hinterlegten Eingaben für Betreff und Text müssen ausgelesen werden. Diese Daten werden als Grundlage für den Vorschlagswert der beiden korrespondierenden Eingabefelder genutzt. Die ID des übergeordneten Beitrags wird über eine Eigenschaft `ReplyID` aus dem Query-String ausgelesen und im ViewState-StateBag der Seite abgespeichert.

Die zweite Änderung betrifft das Speichern des neuen Beitrags. Es gilt, eine Verknüpfung zum bestehenden Datensatz herzustellen. Hier müssen die Felder `ThreadID`, `ParentID` und `OrderID` anders als bei der Anlage eines neuen Threads gesetzt

werden. Über die beschriebene Eigenschaft `ReplyID` wird dazu der übergeordnete Beitrag abgefragt und die Anzahl der zugeordneten Unterbeiträge inkrementiert. Auf diese Weise lässt sich auch die benötigte `OrderID` zur korrekten Sortierung der Beiträge in der Gesamtübersicht anlegen.

Listing 16.28 forumadd1.aspx – Erweiterte Version

```
<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        ph2.Visible = false;

    if((!IsPostBack) && (this.ReplyID != 0))
    {
        OleDbConnection conn =
            new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
                @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
        conn.Open();

        string SQL = "SELECT Subject, Text FROM Forum WHERE ID=@ID;";
        OleDbCommand cmd = new OleDbCommand(SQL, conn);
        cmd.Parameters.Add("@ID", this.ReplyID);
        OleDbDataReader reader = cmd.ExecuteReader();
        if(reader.Read())
        {
            tb_subject.Text = "Re: " + reader["Subject"];
            tb_text.Text = RepeatString("\r\n", 5) + "> " +
                ((string) reader["Text"]).Replace("\r\n", "\r\n> ");
        }
    }
}

int ReplyID
{
    get
    {
        if(Request.QueryString["reply"] != null) ViewState["reply"] =
            int.Parse(Request.QueryString["reply"]);
        if(ViewState["reply"] != null) return((int) ViewState["reply"]);
        return(0);
    }
}

string RepeatString(string str, int count)
{
    string result = string.Empty;
```



```
for(int i=0; i<count; i++)
    result += str;

return(result);
}

void bt_submit_click(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\community.mdb");
    conn.Open();

    string SQL = "INSERT INTO Forum(Name, Email, Subject, [Text])
        VALUES (@Name, @Email, @Subject, @Text)";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    cmd.Parameters.Add("@Name", tb_name.Text);
    cmd.Parameters.Add("@Email", tb_email.Text);
    cmd.Parameters.Add("@Subject", tb_subject.Text);
    cmd.Parameters.Add("@Text", tb_text.Text);
    cmd.ExecuteNonQuery();

    SQL = "SELECT @@Identity";
    cmd = new OleDbCommand(SQL, conn);
    int ID = (int) cmd.ExecuteScalar();

    int ThreadID = ID;
    int ParentID = 0;
    string OrderID = ID.ToString();

    if(this.ReplyID != 0)
    {
        SQL = "UPDATE Forum SET ChildCount=ChildCount+1 WHERE ID=@ReplyID";
        cmd = new OleDbCommand(SQL, conn);
        cmd.Parameters.Add("@ReplyID", this.ReplyID);
        if(cmd.ExecuteNonQuery() == 1)
        {
            SQL = "SELECT ThreadID, OrderID, ChildCount FROM Forum WHERE
                ID=@ReplyID";
            cmd = new OleDbCommand(SQL, conn);
            cmd.Parameters.Add("@ReplyID", this.ReplyID);
            OleDbDataReader reader = cmd.ExecuteReader();
            if(reader.Read())
            {
                ThreadID = (int) reader["ThreadID"];
                ParentID = this.ReplyID;
                OrderID = reader["OrderID"] + "." +
                    reader["ChildCount"].ToString();
            }
        }
    }
}
```

```
}  
reader.Close();  
}  
}  
  
SQL = "UPDATE Forum SET ThreadID=@ThreadID, ParentID=@ParentID,  
OrderID=@OrderID WHERE ID=@ID;";  
cmd = new OleDbCommand(SQL, conn);  
cmd.Parameters.Add("@ThreadID", ThreadID);  
cmd.Parameters.Add("@ParentID", ParentID);  
cmd.Parameters.Add("@OrderID", OrderID);  
cmd.Parameters.Add("@ID", ID);  
cmd.ExecuteNonQuery();  
  
conn.Close();  
  
ph1.Visible = false;  
ph2.Visible = true;  
}  
</script>  
...
```

The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/asp.net/forumadd1.aspx?reply=2`. The page title is "Neuen Beitrag schreiben". Below the title, there is a prompt: "Bitte geben Sie Ihre persönlichen Daten sowie Betreff und Text des Beitrags ein." The form contains four input fields: "Ihr Name:" with the value "Patrick A. Lorenz", "Ihre Email-Adresse:" with the value "pl@p-l.de", "Betreff:" with the value "Re: Erster Beitrag im neuen", and "Text:" with the value "Hallo, toll, dass man jetzt auch antworten kann ;-)) Viele Grüße,". A "Absenden" button is located at the bottom left of the form. The status bar at the bottom shows "Fertig" and "Local intranet".

Neuen Beitrag schreiben

Bitte geben Sie Ihre persönlichen Daten sowie Betreff und Text des Beitrags ein.

Ihr Name:

Ihre Email-Adresse:

Betreff:

Text:

Abbildung 16.35 Auf einen Beitrag lässt sich ganz einfach antworten.

Die drei Abbildungen zeigen das Antworten auf einen bestehenden Beitrag, eine aktualisierte Abbildung der Gesamtübersicht mit weiteren, hierarchisch angeordneten Artikeln sowie die dahinter liegende Tabelle in Access.

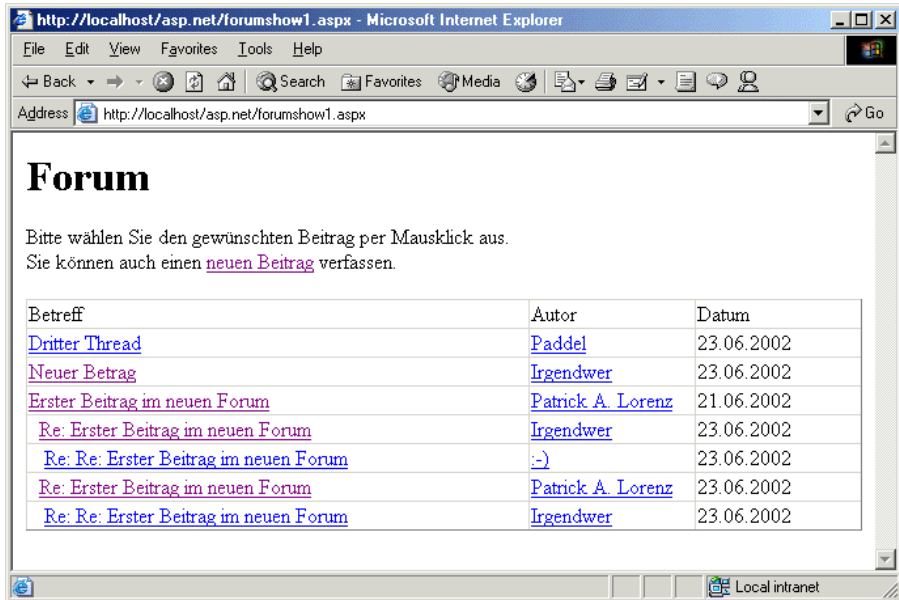


Abbildung 16.36 Die Hierarchie der Artikel ist deutlich zu erkennen.

Forum : Tabelle								
	ID	ThreadID	ParentID	ChildCount	OrderID	Name	Email	Subject
▶	2	2	0	2	2	Patrick A. Lorenz	pl@p-l.de	Erster Beitrag im neuen Forum
	3	2	2	1	2.1	Irgendwer	irgendwer@	Re: Erster Beitrag im neuen Forum
	4	4	0	0	4	Irgendwer	irgendwer@	Neuer Betrag
	8	2	2	1	2.2	Patrick A. Lorenz	pl@p-l.de	Re: Erster Beitrag im neuen Forum
	9	2	8	0	2.2.1	Irgendwer	irgendwer@	Re: Re: Erster Beitrag im neuen Forum
	10	2	3	0	2.1.1	:-)	:-)	Re: Re: Erster Beitrag im neuen Forum
	11	11	0	0	11	Paddel	pl@p-l.de	Dritter Thread
* Wert)	0	0	0	0				

Datensatz: 1 von 7

Abbildung 16.37 Die gezeigten Beiträge lassen sich in der Datenbank wiederfinden.

Schlussbemerkungen

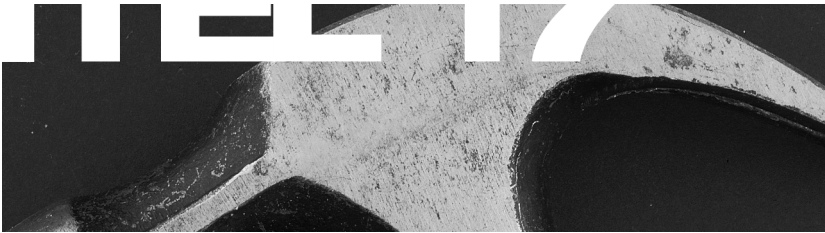
Wie eingangs erwähnt, handelt es sich bei dieser Implementierung um eine minimale Version. Sie können diese noch in vielerlei Hinsicht erweitern, verbessern und Ihren individuellen Anforderungen anpassen. Mögliche Aspekte sind hierbei:

- Administration
- Archivierung älterer Beiträge
- Suche
- Unterstützung für mehrere Foren/Boards
- Und vieles mehr ...

Viel Spaß dabei!

17

E-Commerce



Wie kann ich ...

17 E-Commerce

E-Commerce ist ein absolut wichtiges Thema im Internet. Viele Entwickler stehen immer wieder vor der Aufgabe, Web-Shops zu implementieren oder auch nur einzelne Funktionen zu ergänzen. In diesem Kapitel finden Sie zahlreiche Hilfsmittel für wiederkehrende Problemfälle.

Für dieses Kapitel existiert eine spezielle Beispieldatenbank mit dem Namen `shop.mdb`. Diese wird von allen vorgestellten Beispielen genutzt.

17.1 ... eine hierarchische Produktübersicht erzeugen?

Produktdaten lassen sich schlecht pauschal beschreiben. Zu unterschiedlich sind die individuellen Anforderungen an das System. In diesem Rezept möchte ich Ihnen eine einfache Variante vorstellen, die Sie mit meist wenigen Handgriffen an Ihre jeweiligen Bedürfnisse anpassen können. Das System ist angelehnt an das Rezept „... eine Link-Sammlung erstellen?“ aus dem Kapitel „Community“, und tatsächlich ähneln sich die Anforderungen ziemlich stark.

Datenbank

Natürlich wird auch oder sogar gerade eine Produktübersicht durch die zugrunde liegende Datenbank bestimmt. Analog zur angesprochenen Link-Sammlung existieren auch in diesem Fall zwei Tabellen. Während die erste eine in sich relationale Anordnung der Produktgruppen beinhaltet, werden in der zweiten die eigentlichen Produktinformationen abgelegt. Dazu gehören der Name des Produktes, eine Beschreibung, der Nettopreis und der Umsatzsteuersatz.



Abbildung 17.1 Die Tabelle nimmt die einzelnen Produkte auf.

Die Beispieldatensätze entsprechen den von mir geschriebenen Buchtiteln zu .NET, es könnte sich aber auch um jede andere Form von Produkten handeln. Wäre ich ein Al Bundy-Fan, wären es wohl Schuhe ...

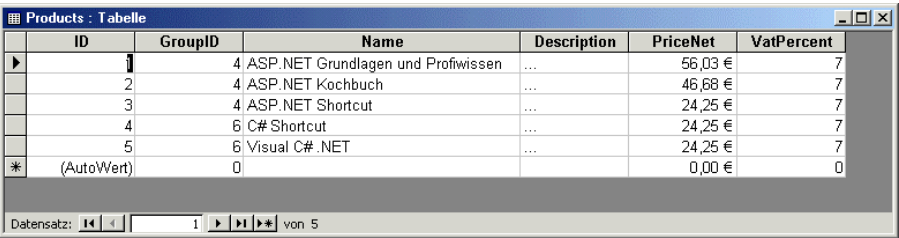


Abbildung 17.2 Die hinterlegten Datensätze

Produktübersicht

Die Produktübersicht besteht aus einer einzelnen ASP.NET-Seite. Diese enthält zwei DataList-Controls zur Auswahl der Produktgruppe sowie zur Anzeige der darin enthaltenen Produkte. Die hinterlegten Vorlagen werden mit Hilfe zweier SQL-Abfragen zum Leben erweckt. Zu beachten ist die Berechnung des Bruttopreises auf Basis des Nettopreises innerhalb der Query.

Listing 17.1 Products1.aspx

```

<script runat="server">

void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        BindData(1);
}

void BindData(int groupID)
{
    OleDbConnection conn1 =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\shop.mdb");
    conn1.Open();

    OleDbConnection conn2 =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\shop.mdb");
    conn2.Open();

    string SQL_group = string.Format("SELECT * FROM Products_Groups
        WHERE ID={0};", groupID);
    OleDbCommand cmd_group = new OleDbCommand(SQL_group, conn1);
    OleDbDataReader reader_group = cmd_group.ExecuteReader();
    reader_group.Read();

    lb_cat.Text = (string) reader_group["Title"];

    bt_back.Visible = (groupID != 1);
    bt_back.CommandArgument = reader_group["ParentID"].ToString();
    reader_group.Close();

    string SQL1 = string.Format("SELECT * FROM Products_Groups
        WHERE ParentID={0};", groupID);
    OleDbCommand cmd1 = new OleDbCommand(SQL1, conn1);
    groups.DataSource = cmd1.ExecuteReader();

    string SQL2 = string.Format("SELECT *, (PriceNet +
        (PriceNet/100*VatPercent)) AS PriceGross FROM Products
        WHERE GroupID={0};", groupID);
    OleDbCommand cmd2 = new OleDbCommand(SQL2, conn2);
    products.DataSource = cmd2.ExecuteReader();

    DataBind();
    conn2.Close();
    conn1.Close();

    groups.Visible = (groups.Items.Count > 0);
}

```

```
products.Visible = (products.Items.Count > 0);
}

void groups_ItemCommand(object sender, DataListCommandEventArgs e)
{
    if (e.CommandName == "selectgroup")
        BindData(int.Parse(e.CommandArgument.ToString()));
}

void back_ItemCommand(object sender, CommandEventArgs e)
{
    if (e.CommandName == "selectgroup")
        BindData(int.Parse(e.CommandArgument.ToString()));
}

</script>

<form runat="server">

    <h1> Kategorie: <ASP:Label id="lb_cat" runat="server"/> </h1>

    <p>
        <ASP:LinkButton id="bt_back" runat="server"
            Text="Zurück"
            OnCommand="back_ItemCommand"
            CommandName="selectgroup"/>
    </p>

    <ASP:DataList id="groups" runat="server"
        RepeatColumns="2"
        OnItemCommand="groups_ItemCommand">

        <HeaderTemplate>
            <h2>Bitte wählen Sie die gewünschte Produktgruppe aus</h2>
        </HeaderTemplate>

        <ItemTemplate>
            <ASP:LinkButton runat="server"
                Text=<%=# DataBinder.Eval(Container.DataItem, "Title") %>
                CommandName="selectgroup"
                CommandArgument=<%=# DataBinder.Eval(Container.DataItem, "ID") %>/>
            </ItemTemplate>

    </ASP:DataList>

    &nbsp;

    <ASP:DataList id="products" runat="server"
        RepeatColumns="2">
```

```

<HeaderTemplate>
  <h2>Bitte wählen Sie das gewünschte Produkt aus</h2>
</HeaderTemplate>

<ItemTemplate>
  <ASP:HyperLink runat="server"
    Text=<%=# DataBinder.Eval(Container.DataItem, "Name") %>
    NavigateUrl=<%=# "product.aspx?id=" +
      DataBinder.Eval(Container.DataItem, "ID") %>/>
  <br>
  <%=# DataBinder.Eval(Container.DataItem, "Description") %>
  Preis: <%=# DataBinder.Eval(Container.DataItem, "PriceGross",
    "{0:0.00} €") %><br>
</ItemTemplate>

</ASP:DataList>
</form>

```

Die Namen der einzelnen Produkte werden über HyperLink-Controls ausgegeben. Diese verweisen auf die – nicht vorhandene – Seite `product.aspx`. Die Datensatz-ID des Produktes wird im Query-String übergeben. Die Seite könnte optional weitere Informationen zu dem gewählten Artikel anzeigen.

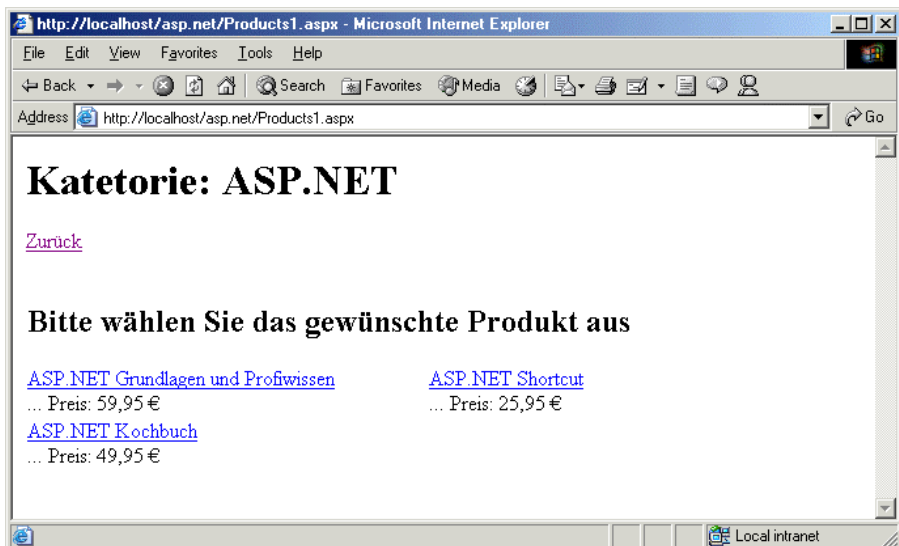


Abbildung 17.3 Eine einfache Produktübersicht mit beliebigen Hierarchien

17.2 ... ermitteln, ob ein Land zur EU gehört?

Für die in einem Shop angezeigten Informationen ist es mitunter wichtig, ob das Zielland innerhalb der EU liegt oder nicht. Dem Benutzer kann vor dem Eintreten in den Shop eine Auswahl seines Landes angezeigt werden, beziehungsweise diese kann bereits auf Basis des entsprechenden HTTP-Kopfzeileneintrags vorselektiert werden (vergleiche Rezept „... die Sprache des Besuchers erkennen?“ im Kapitel „Basics“). Ist ein Land ausgewählt, kann mit Hilfe einer einfachen Methode dessen Zugehörigkeit zur EU ermittelt werden.

Listing 17.2 IsEUCountry1.aspx

```
<script language="C#" runat=server>

void Page_Load(object sender, EventArgs e)
{
    OleDbConnection conn =
        new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data Source=c:\inetpub\wwwroot\aspnet\countries.mdb");
    conn.Open();

    string SQL = "SELECT ID,TextDE FROM Countries ORDER BY TextDE;";
    OleDbCommand cmd = new OleDbCommand(SQL, conn);
    countrylist.DataSource = cmd.ExecuteReader();

    if(!IsPostBack)
    {
        countrylist.DataBind();
        cl_SelectedIndexChanged(countrylist, EventArgs.Empty);
    }

    conn.Close();
}

void cl_SelectedIndexChanged(object sender, EventArgs e)
{
    string country = countrylist.SelectedItem.Value;
    lb.Text = (this.IsEUCountry(country) ? "Ja" : "Nein");
}

bool IsEUCountry(string country)
{
    string[] EUCountries = {"D", "B", "DK", "FIN", "F", "GR", "IRL", "I",
        "L", "NL", "A", "P", "S", "E", "GB"};
    return(Array.IndexOf(EUCountries, country.ToUpper()) != -1);
}

</script>
```

```
<form runat="server">

  <p> Bitte wählen Sie Ihr Land aus:
    <ASP:DropDownList id="countrylist" runat="server"
      DataValueField="ID"
      DataTextField="TextDE"
      AutoPostBack="true"
      OnSelectedIndexChanged="cl_SelectedIndexChanged"/>
  </p>

  <p> EU-Land?
    <ASP:Label id="lb" runat="server"/>
  </p>

</form>
```

Das Beispiel realisiert ein DropDownList-Control zur Auswahl des gewünschten Landes. In einem Label darunter wird ausgegeben, ob das Land Mitglied der EU ist oder nicht.

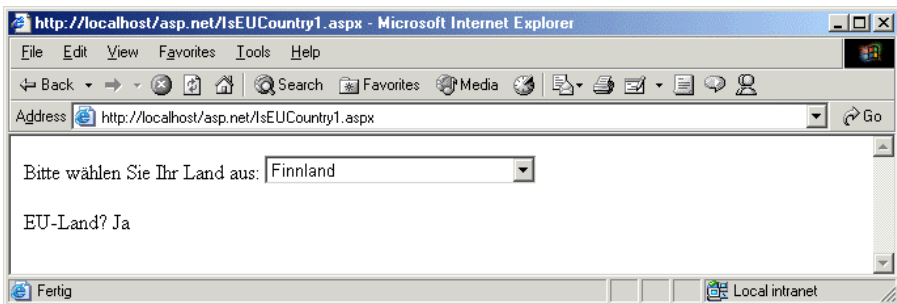


Abbildung 17.4 Gute Handys kommen aus der EU.

Beachten Sie, dass diese Abfrage ausschließlich echte Mitgliedsstaaten der Europäischen Gemeinschaft berücksichtigt. Assoziierte Länder sowie Gebiete der EU, die außerhalb von Europa liegen (Guadeloupe, Martinique etc.) sind nicht eingeschlossen. Der Grund hierfür ist nicht Faulheit, vielmehr entspricht dies dem geläufigen Einsatz der Abfrage.

17.3 ... ermitteln, ob ein Land den Euro als Zahlungsmittel verwendet?

Sehr ähnlich wie die Abfrage der Mitgliedschaft in der EU aus dem vorherigen Rezept kann auch ermittelt werden, ob ein Land den Euro als offizielles Zahlungsmittel akzeptiert. Berücksichtigt werden die zwölf Mitgliedsstaaten.

Listing 17.3 IsEuroCountry1.aspx

```
...  
  
bool IsEuroCountry(string country)  
{  
    string[] EuroCountries = {"16:46 28.06.2002D", "B", "FIN", "F", "GR",  
        "IRL", "I", "L", "NL", "A", "P", "E"};  
    return(Array.IndexOf(EuroCountries, country.ToUpper()) != -1);  
}  
  
...
```

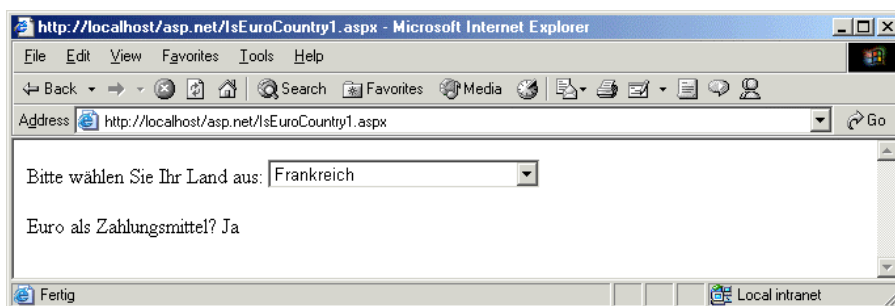


Abbildung 17.5 Viva la révolution!

17.4 ... ermitteln, ob ein Land umsatzsteuerpflichtig ist?

Prinzipiell muss die Umsatzsteuer nur zahlen, wer innerhalb von Deutschland bestellt. Bei Lieferungen ins Ausland wird lediglich der Nettopreis berechnet. Das gilt sowohl für Mitgliedsländer der EU als auch die restliche Welt. Bei Ländern der EU muss aber eine gültige Umsatzsteueridentifikationsnummer angegeben werden. Ist dies nicht der Fall, muss die Steuer trotz Lieferung ins Ausland berechnet werden. Dies gilt insbesondere für Privatpersonen.

Bei der Abfrage, ob ein Land umsatzsteuerpflichtig ist, sollte dabei immer berücksichtigt werden, ob eine so genannte Vat-ID vorhanden ist oder nicht. Diese sollte selbstverständlich im Verlaufe der Bestellung noch überprüft, der Standardwert kann aber anhand der Zielgruppe festgelegt werden. Bei Geschäftskunden sollte vom Vorhandensein ausgegangen werden, bei Privatpersonen eher nicht.

Das folgende Beispiel zeigt eine Abfrage entsprechend dem ausgewählten Land sowie dem Besitz einer Vat-ID. Neben einer `DropDownList` existiert dazu eine `CheckBox`, die vorausgefüllt wird. Dabei werden als Zielgruppe Geschäftskunden angenommen. Das zusätzliche `Label` gibt an, ob im vorliegenden Fall die Steuer berechnet werden muss oder nicht.

Listing 17.4 `IsVatCountry1.aspx`

```
<script language="C#" runat=server>

...

void cl_SelectedIndexChanged(object sender, EventArgs e)
{
    string country = countrylist.SelectedItem.Value;
    bool IsEU = IsEUCountry(country);
    cb.Checked = (IsEU && country.ToUpper() != "D");
    cb.Enabled = IsEU;
    lb.Text = (this.IsVatCountry(country, cb.Checked) ? "Ja" : "Nein");
}

void cb_CheckedChanged(object sender, EventArgs e)
{
    string country = countrylist.SelectedItem.Value;
    lb.Text = (this.IsVatCountry(country, cb.Checked) ? "Ja" : "Nein");
}

bool IsVatCountry(string country, bool hasVatID)
{
    bool IsEU = IsEUCountry(country);
    return((country.ToUpper() == "D") || (IsEU && !hasVatID));
}

bool IsEUCountry(string country)
{
    string[] EUCountries = {"D", "B", "DK", "FIN", "F", "GR", "IRL",
        "I", "L", "NL", "A", "P", "S", "E", "GB"};
    return(Array.IndexOf(EUCountries, country.ToUpper()) != -1);
}

</script>
```

```
<form runat="server">

<p> Bitte wählen Sie Ihr Land aus:
<ASP:DropDownList id="countrylist" runat="server"
    DataValueField="ID"
    DataTextField="TextDE"
    AutoPostBack="true"
    OnSelectedIndexChanged="cl_SelectedIndexChanged"/>
</p>

<p>
<ASP:CheckBox id="cb" runat="server"
    Text="Hat Vat-ID"
    AutoPostBack="true"
    OnCheckedChanged="cb_CheckedChanged"/>
</p>

<p> Umsatzsteuerpflichtig?
<ASP:Label id="lb" runat="server"/>
</p>

</form>
```

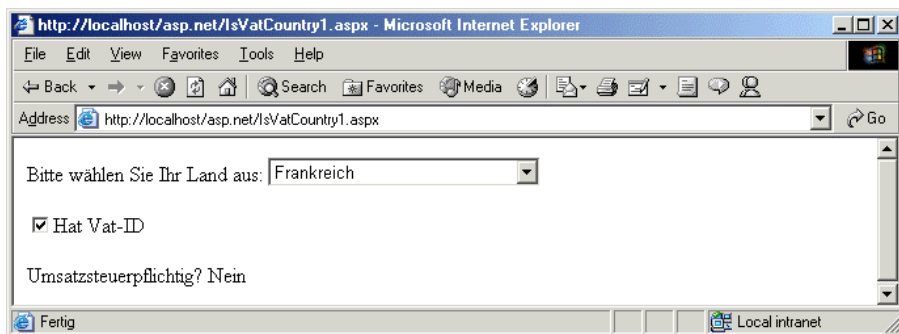


Abbildung 17.6 Franzosen mit gültiger Vat-ID können steuerfrei einkaufen.



Hinweise zur Überprüfung einer Umsatzsteueridentifikationsnummer finden Sie in den Rezepten „... eine Umsatzsteuer-ID syntaktisch überprüfen?“ sowie „... eine Umsatzsteuer-ID online beim nationalen Finanzamt überprüfen?“.

Beachten Sie bitte, dass nach neuer Gesetzeslage auf ausgestellten Rechnungen in das EU-Ausland, bei denen auf die Berechnung der Umsatzsteuer verzichtet wird, sowohl die Vat-ID des Rechnungsstellers als auch die des -empfängers angegeben werden muss.

17.5 ... Preise mit und ohne Umsatzsteuer anzeigen?

Das vorherige Beispiel hat gezeigt, auf welcher Basis über die Berechnung der Umsatzsteuer entschieden werden sollte. Entsprechend dieser Information sollte die Anzeige der Preise gleich korrekt, also entweder mit oder ohne Umsatzsteuer erfolgen. Ich habe dazu die Seite aus dem Rezept „... eine hierarchische Produktübersicht erzeugen?“ ein klein wenig angepasst. Zusätzlich ist nun eine Auswahl des Landes sowie der Vat-ID möglich. Aufgrund der geänderten Ausgangsposition musste ich das Listing ein wenig ändern, so dass nun zwei Eigenschaften `ShowVat` und `CurrentGroupID` informieren, ob die Preise mitsamt Umsatzsteuer angezeigt werden und welche ID die aktuelle Produktgruppe hat.

Listing 17.5 Products2.aspx

```
<script runat="server">

bool ShowVat
{
    get
    {
        if(countrylist.SelectedItem == null) return(true);
        string country = countrylist.SelectedItem.Value;
        return(this.IsVatCountry(country, cb.Checked));
    }
}

int CurrentGroupID
{
    get
    {
        if(ViewState["ID"] == null) return(1);
        return((int) ViewState["ID"]);
    }
    set { ViewState["ID"] = value; }
}

...

if(this.ShowVat)
    SQL = string.Format("SELECT *, (PriceNet +
        (PriceNet/100*VatPercent)) AS Price FROM Products WHERE
        GroupID={0};", this.CurrentGroupID);
else
    SQL = string.Format("SELECT *, PriceNet AS Price FROM Products
        WHERE GroupID={0};", this.CurrentGroupID);

...
```

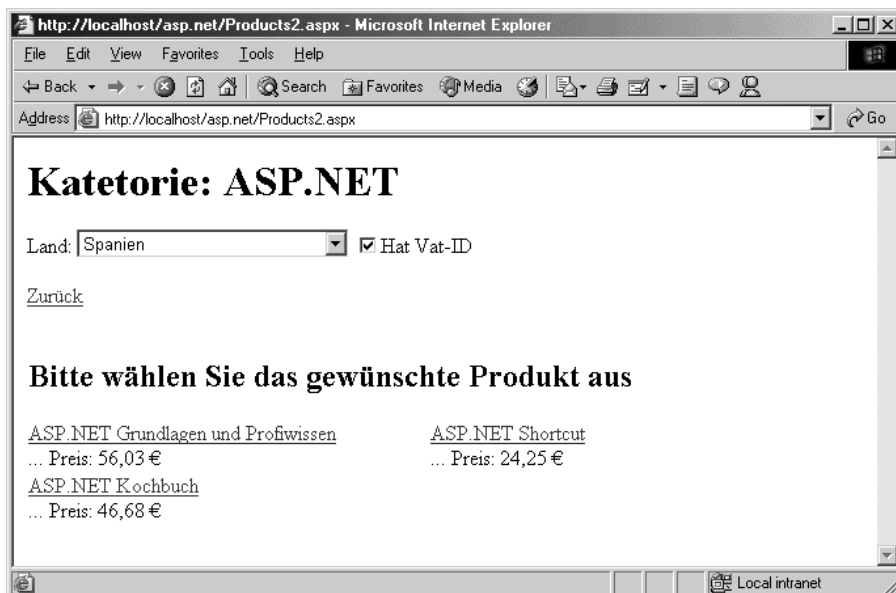


Abbildung 17.7 Die Produktpreise werden für Spanien ohne Umsatzsteuer angezeigt¹.

17.6 ... eine Umsatzsteuer-ID syntaktisch überprüfen?

Die Umsatzsteueridentifikationsnummer entspricht in jedem Land einem bestimmten Aufbau. Dieser lässt sich beispielsweise mit Hilfe von regulären Ausdrücken überprüfen. Das Listing zeigt die Verwendung. Nach Eingabe einer ID reicht ein Button-Klick zur syntaktischen Überprüfung. Verwendet wird die statische Methode `IsMatch` der Klasse `Regex` innerhalb einer `switch`-Abfrage auf Basis des gewählten Landes.

Listing 17.6 CheckVatID1.aspx

```
<script runat="server">

void bt_click(object sender, EventArgs e)
{
    bool IsVatIDOK = CheckVatID(ddl.SelectedItem.Value, tb.Text);
}
```

¹ Dem „geneigten“ Leser mag es nicht entgangen sein, dass Schuhe doch das bessere Produkt gewesen wäre, da Bücher oftmals einem besonders geschützten Preisverfahren unterliegen, so existiert bei uns in Deutschland die Buchpreisbindung. Insofern ist die Angabe ohne Umsatzsteuer genau genommen irreführend. Aber wie gesagt, ich bin kein Fan von Al Bundy ...

```

lb.Text = "Die eingegebene Vat-ID ist syntaktisch ";
lb.Text += (IsVatIDOK ? "OK" : "nicht OK");
}

bool CheckVatID(string country, string vatID)
{
    switch(country.ToUpper())
    {
        case "D":
            return(Regex.IsMatch(vatID, @"^DE\d{9}$"));

        case "B":
            return(Regex.IsMatch(vatID, @"^BE\d{9}$"));

        case "DK":
            return(Regex.IsMatch(vatID, @"^DK\d{8}$"));

        case "FIN":
            return(Regex.IsMatch(vatID, @"^FI\d{8}$"));

        case "F":
            return(Regex.IsMatch(vatID, @"^FR\w{2}\d{9}$"));

        case "GR":
            return(Regex.IsMatch(vatID, @"^EL\d{8,9}$"));

        case "IRL":
            return(Regex.IsMatch(vatID, @"^IE(\d\w)(\d{5})[A-Za-z]$"));

        case "I":
            return(Regex.IsMatch(vatID, @"^IT\d{11}$"));

        case "L":
            return(Regex.IsMatch(vatID, @"^LU\d{8}$"));

        case "NL":
            return(Regex.IsMatch(vatID, @"^NL(\d{9})B(\d{2})$"));

        case "A":
            return(Regex.IsMatch(vatID, @"^ATU(\d{8})$"));

        case "P":
            return(Regex.IsMatch(vatID, @"^PT\d{9}$"));

        case "S":
            return(Regex.IsMatch(vatID, @"^SE(\d{10})01$"));

        case "E":

```

```

    return(Regex.IsMatch(vatID, @"^ES([A-Za-z]\d{8})|(\d{8}[A-Za-z])|([A-Za-z]\d{7}[A-Za-z])$"));

    case "GB":
        return(Regex.IsMatch(vatID, @"(^GB\d{9}$)|(^GB\d{12}$)"));
    }
    return(false);
}

</script>

<form runat="server">

    <p> Land:
    <ASP:DropDownList id="ddl" runat="server">
        <ASP:ListItem Value="D" Text="Deutschland" Selected="true"/>
        <ASP:ListItem Value="B" Text="Belgien"/>
        <ASP:ListItem Value="DK" Text="Dänemark"/>
        <ASP:ListItem Value="FIN" Text="Finnland"/>
        <ASP:ListItem Value="F" Text="Frankreich"/>
        <ASP:ListItem Value="GR" Text="Griechenland"/>
        <ASP:ListItem Value="IRL" Text="Irland"/>
        <ASP:ListItem Value="I" Text="Italien"/>
        <ASP:ListItem Value="L" Text="Luxemburg"/>
        <ASP:ListItem Value="NL" Text="Niederlande"/>
        <ASP:ListItem Value="A" Text="Österreich"/>
        <ASP:ListItem Value="P" Text="Portugal"/>
        <ASP:ListItem Value="S" Text="Schweden"/>
        <ASP:ListItem Value="E" Text="Spanien"/>
        <ASP:ListItem Value="GB" Text="Großbritannien"/>
    </ASP:DropDownList>
    </p>

    <p> Umsatzsteueridentifikationsnummer:
    <ASP:TextBox id="tb" runat="server"/>
    </p>

    <p> <ASP:Label id="lb" runat="server"/> </p>

    <ASP:Button runat="server" Text="Überprüfen" OnClick="bt_click"/>

</form>

```

In diesem Beispiel wird ausschließlich der syntaktische Aufbau der Vat-ID sowie deren Zugehörigkeit zu einem ausgewählten Land überprüft. Jede ID beginnt mit einem zweistelligen Landeskürzel. Die deutsche Variante besteht beispielsweise aus dem einleitenden Kürzel „DE“ und neun darauf folgenden Ziffern.

Die Überprüfung berücksichtigt *nicht* die Prüfziffer, die in jeder ID enthalten ist. Die Mechanismen hierzu sind relativ einfach, würden den Umfang des Buches jedoch sprengen. Auf Wunsch stelle ich eventuell eine entsprechend erweiterte Version im Internet zur Verfügung. Ansonsten finden Sie die notwendigen Informationen jedoch auch online unter folgender Adresse:

<http://www.pruefzifferberechnung.de>

Weitere Informationen stellt auch das Bundesamt für Finanzen zur Verfügung:

<http://www.bff-online.de>



Beachten Sie bitte, dass eine syntaktische Überprüfung der Vat-ID der Sorgfaltspflicht, die sich aus § 6a Absatz 4 UStG ergibt, nicht genügt.

Diese verlangt nach einer qualifizierten Prüfung auf Basis von Name und Anschrift des Unternehmens. Eine derartige Überprüfung kann man durch das BfF (Außenstelle Saarlouis) beispielsweise schriftlich, per Email oder auch telefonisch vornehmen lassen. Weitere Informationen hierzu finden Sie auf der genannten Website. Beachten Sie in diesem Zusammenhang bitte auch das nachfolgende Rezept.

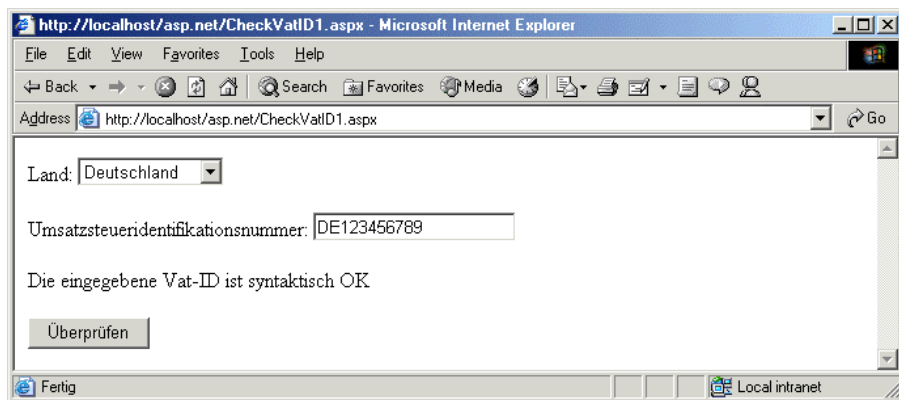


Abbildung 17.8 Die eingegebene Vat-ID ist syntaktisch korrekt.

Die Überprüfung einer Vat-ID kann alternativ auch über ein Validation Control erfolgen. Entweder verwenden Sie hierzu das CustomValidator-Control oder ein eigenes Objekt. Weitere Informationen zu diesem Thema finden Sie im Rezept „... überprüfen, ob eine Eingabe einem individuellen Schema entspricht?“ im Kapitel „Eingabeformulare“.

17.7 ... eine Umsatzsteuer-ID online beim nationalen Finanzamt überprüfen?

Das Bundesamt für Finanzen (BfF) ermöglicht die Integration einer Online-Überprüfung für Umsatzsteueridentifikationsnummern in eigene Web-Applikationen. Dem Online-Dienst wird die eigene (deutsche) sowie die zu überprüfende ID übergeben, die *nicht* in Deutschland vergeben wurde. Der Dienst leitet die Anfrage an die jeweils zuständige nationale Behörde weiter. Diese überprüft die Existenz der Nummer und meldet das Ergebnis an das BfF und darüber an den Aufrufer mit.

Die Übergabe der beiden Parameter erfolgt über den Query-String. Das Ergebnis wird in Form eines WDDX-Streams (Web Distributed Data Exchange) geliefert, was nichts anderes als XML ist. Die Weiterbearbeitung der zurückgelieferten Daten kann also mit Hilfe der .NET XML-Klassen erfolgen.

Die Adresse zur Abfrage der Daten sieht so aus:

```
http://wddx.bff-online.de/ustid.php
?eigene_id=<Ihre UstID>
&abfrage_id=<Zu prüfende UstID>
```

Das WDDX-Ergebnis der Abfrage sieht im Fall einer korrekten ID beispielsweise wie folgt aus:

```
<?xml version='1.0'?>
<!DOCTYPE wddxPacket SYSTEM 'wddx_0100.dtd'>
<wddxPacket version='1.0'>
  <header/>
  <data>
    <struct>
      <var name='abfrage_id'><string>...</string></var>
      <var name='uhrzeit'><string>11:49:57</string></var>
      <var name='datum'><string>29.06.2002</string></var>
      <var name='eigene_id'><string>...</string></var>
      <var name='fehler_text'><string>Die angefragte USt-IdNr.
        ist zum o. g. Zeitpunkt gueltig.</string></var>
      <var name='fehler_code'><string>200</string></var>
    </struct>
  </data>
</wddxPacket>
```

Im Beispiel sehen Sie zwei Eingabefelder für die eigene sowie die zu überprüfende Vat-ID. Ein Button-Klick resultiert in einer Abfrage, die mit Hilfe der Klasse `WebClient` aus dem Namespace `System.Net` durchgeführt wird. Der von der Methode `OpenRead` gelieferte Stream wird direkt der Methode `Load` einer neu angelegten

XmlDocument-Instanz übergeben. Auf dieser Basis kann nun das Element „fehler_code“ abgefragt werden. War die Anfrage erfolgreich, enthält dieses Element den Statuscode „200“. Alternativ könnte die URL auch direkt einer Überladung der Methode `XmlDocument.Load` übergeben werden.

Listing 17.7 CheckVatID2.aspx

```
<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.Xml" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    bool IsVatIDOK = CheckVatID(tb1.Text, tb2.Text);

    lb.Text = "Die eingegebene Vat-ID ist syntaktisch ";
    lb.Text += (IsVatIDOK ? "OK" : "nicht OK");
}

bool CheckVatID(string vatID, string vatIDToCheck)
{
    string url = string.Format("http://wddx.bff-online.de/
        ustid.php?eigene_id={0}&abfrage_id={1}", vatID, vatIDToCheck);

    WebClient webclient = new WebClient();
    XmlDocument doc = new XmlDocument();
    doc.Load(webclient.OpenRead(url));

    XmlElement root = doc.DocumentElement;
    XmlNode node = root.SelectSingleNode(
        "data/struct/var[@name='fehler_code']/string");

    return (node.InnerText == "200");
}

</script>

<form runat="server">

<p> Ihre Umsatzsteueridentifikationsnummer:
    <ASP:TextBox id="tb1" runat="server"/>
</p>

<p> Zu prüfende Umsatzsteueridentifikationsnummer:
    <ASP:TextBox id="tb2" runat="server"/>
</p>

<p> <ASP:Label id="lb" runat="server"/> </p>
```

```
<ASP:Button runat="server" Text="Überprüfen" OnClick="bt_click"/>  
</form>
```

Damit das Beispiel wie in der Abbildung gezeigt funktioniert, muss eine (leere) XML Document Type Definition (DTD) im System32-Verzeichnis abgelegt werden. Der Grund ist die relative Angabe der DTD im zurückgelieferten XML- beziehungsweise WDDX-Stream. Die Klasse `XmlDocument` kommt daher auf die Idee, die DTD lokal zu suchen, was in einer `IOException` resultiert, solange nicht eine leere Datei im genannten Verzeichnis abgelegt wird.

Sollte in Ihrem Fall der Zugriff auf das Verzeichnis nicht möglich sein, so können Sie den zurückgelieferten `Stream` zuvor mittels der Klasse `StreamReader` aus dem Namespace `System.IO` in eine Zeichenkette umwandeln und die Angabe der DTD vor der Übergabe an die Klasse `XmlDocument` mittels `Replace` entfernen. Beachten Sie, dass Sie in diesem Fall nicht die Methode `Load`, sondern die Methode `LoadXml` verwenden müssen.

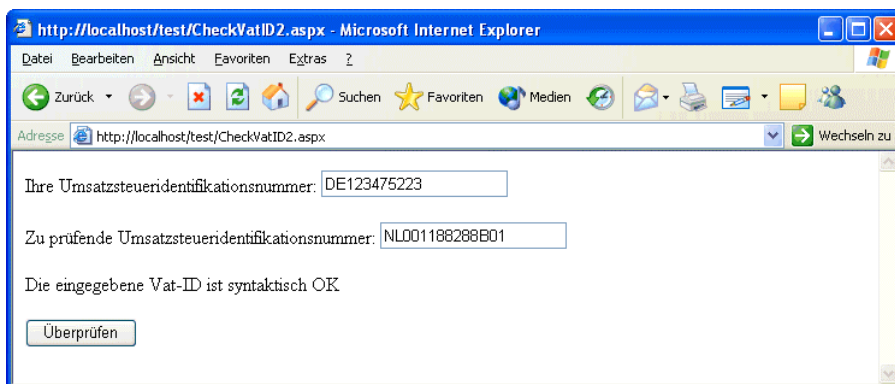


Abbildung 17.9 Die eingegebenen Nummern sind korrekt.

Da eine derartige Funktionalität sicherlich häufig benötigt wird, habe ich einen kleinen Web Service geschrieben, der diese in .NET-Manier kapselt. Der Methode `CheckOnline` werden die zwei bekannten Parameter übergeben. Zurückgeliefert wird eine Instanz der Struktur `VatIDCheckResult`, in der die Inhalte des gelieferten XML-Streams als öffentliche Felder angeboten werden. Selbstverständlich werden die Werte hier in den korrekten Datentypen angeboten.

Listing 17.8 CheckVatID.asmx

```

<% @WebService Class="VatIDCheck" Language="C#" Debug="true" %>

using System;
using System.Web.Services;
using System.Xml;
using System.IO;

[WebService]
public class VatIDCheck : WebService
{
    [WebMethod]
    public VatIDCheckResult CheckOnline(string vatID,
        string vatIDToCheck)
    {
        string url = string.Format("http://wddx.bff-online.de/
            ustid.php?eigene_id={0}&abfrage_id={1}", vatID, vatIDToCheck);

        XmlDocument doc = new XmlDocument();
        doc.Load(url);
        XmlElement root = doc.DocumentElement;

        VatIDCheckResult result = new VatIDCheckResult();
        result.VatID = root.SelectSingleNode(
            "data/struct/var[@name='eigene_id']/string").InnerText;

        result.VatIDToCheck = root.SelectSingleNode(
            "data/struct/var[@name='abfrage_id']/string").InnerText;

        string date = root.SelectSingleNode(
            "data/struct/var[@name='datum']/string").InnerText;
        string time = root.SelectSingleNode(
            "data/struct/var[@name='uhrzeit']/string").InnerText;
        result.CheckTime = DateTime.Parse(string.Format("{0} {1}",
            date, time));

        result.ErrorText = root.SelectSingleNode(
            "data/struct/var[@name='fehler_text']/string").InnerText;

        result.ErrorCode = int.Parse(root.SelectSingleNode(
            "data/struct/var[@name='fehler_code']/string").InnerText);

        result.IsOK = (result.ErrorCode == 200);

        return(result);
    }
}

public struct VatIDCheckResult

```

```
{  
    public string VatID;  
    public string VatIDToCheck;  
    public DateTime CheckTime;  
    public string ErrorText;  
    public int ErrorCode;  
    public bool IsOK;  
}
```

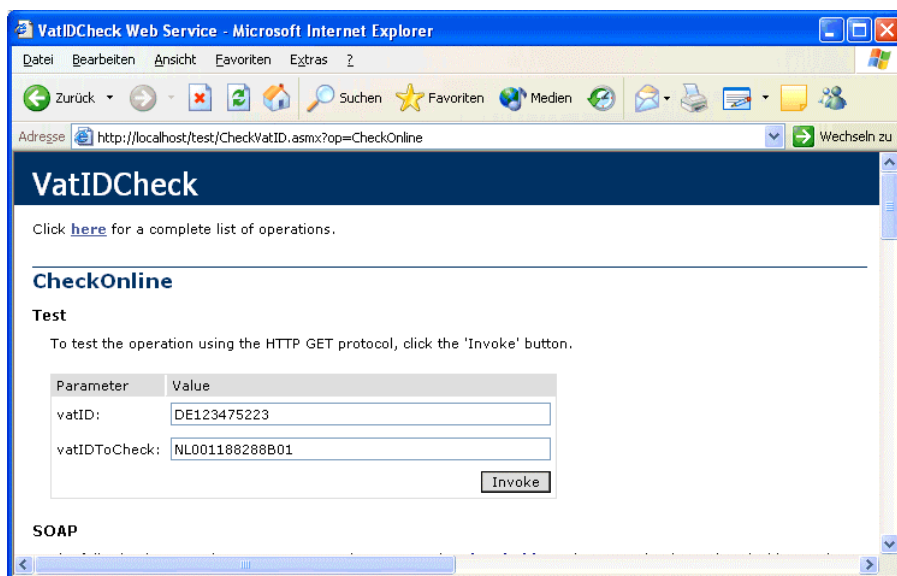


Abbildung 17.10 Die Eingabemaske des Web Services

Die beiden Abbildungen zeigen den Web Service im Einsatz. Dieser kann wie gewohnt mittels `wsdl.exe` referenziert und in eigene Web-Applikationen eingebunden werden.

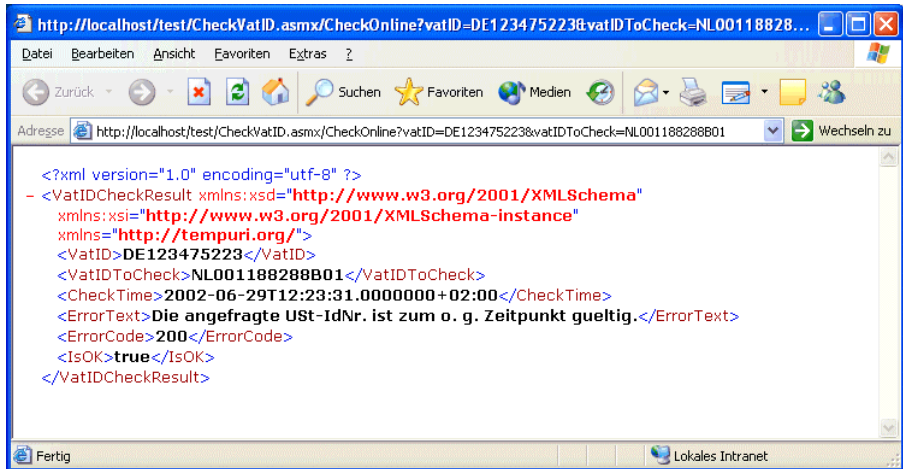


Abbildung 17.11 Das Ergebnis wird in Form einer Struktur geliefert.

Zusätzliche Hinweise

Eine Liste der möglichen Fehlercodes und deren Ursache erhalten Sie auf der Website des BfF unter folgender Adresse:

<http://www.bff-online.de>

Der Dienst wird Ihnen kostenlos durch das Bundesamt für Finanzen zur Verfügung gestellt. Bei Verwendung erkennen Sie dessen Nutzungsbedingung an. Der Service ist laut Angaben des BfF außer bei Wartungsarbeiten täglich zwischen 5.00 und 23.00 Uhr verfügbar. Weitere Informationen finden Sie auf der oben genannten Website. Der Autor des Buches bietet den Dienst nicht an und ist für dessen Verwendbarkeit nicht verantwortlich.

Beachten Sie bitte, dass eine syntaktische Überprüfung der Vat-ID der Sorgfaltpflicht, die sich aus § 6a Absatz 4 UstG ergibt, nicht genügt. Diese verlangt nach einer qualifizierten Prüfung auf Basis von Name und Anschrift des Unternehmens. Eine derartige Überprüfung kann man durch das BfF (Außenstelle Saarlouis) beispielsweise schriftlich, per Email oder auch telefonisch vornehmen lassen. Weitere Informationen hierzu finden Sie auf der genannten Website.

17.8 ... aktuelle Währungsinformationen erhalten?

Es gibt zahlreiche Möglichkeiten und Methoden, aktuelle Daten für eine Online-Währungsumrechnung zu erhalten. Alle vorzustellen wäre müßig und vermutlich noch nicht einmal möglich. Ich habe daher eine Variante herausgepickt, die Sie zum Testen direkt und ohne Anmeldung nutzen können. Die Rede ist von Yahoo. Das Unternehmen bietet auf seinem Finanzportal ständig aktualisierte Währungsinformationen an. Rein theoretisch können Sie die Daten intern abfragen und zur eigenen Verwendung auswerten. Die Abbildung zeigt die entsprechende Seite des Portals.

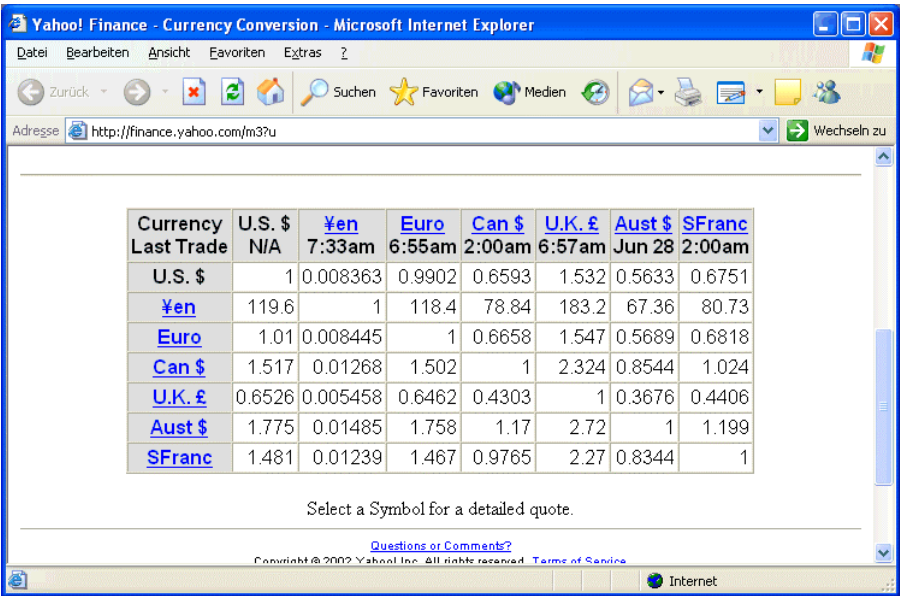


Abbildung 17.12 Yahoo liefert ständig aktualisierte Währungskurse.

Sie finden die gezeigte Seite unter folgender Adresse:

http://finance.yahoo.com/m3?u

Die nachfolgende Beispielseite enthält eine Währungsumrechnung. Dazu sind zwei DropDownList-Controls zur Auswahl der Quell- und Zielwährung sowie ein Eingabefeld für den Umrechnungsbetrag enthalten. Ein Button-Klick fragt die gezeigte Seite von Yahoo ab, liest den entsprechenden Kurs aus und präsentiert das Umrechnungsergebnis in einem Label.

Die interne Abfrage der Seite ist relativ einfach und geschieht mit Hilfe der Klasse WebClient aus dem Namespace System.Net. Detailinformationen finden Sie im

Rezept „... Inhalte anderer Seiten scrapen?“ im Kapitel „Content-Management“. Mit Hilfe von Zeichenkettenoperationen wird die in der Abbildung gezeigte Tabelle ausgeschnitten und anschließend in ein HTML Server Control umgewandelt. Auf diese Weise sparen Sie das lästige manuelle Parsen der Tabelle. Auch hier finden Sie im genannten Kapitel im Rezept „... eine Tabelle parsen?“ weitere Hintergrundinformationen.

Listing 17.9 GetCurrencies1.aspx

```
<% @Import Namespace="System.Globalization" %>
<% @Import Namespace="System.Net" %>
<% @Import Namespace="System.IO" %>

<script runat="server">

void bt_click(object sender, EventArgs e)
{
    if(Cache["CurrencyTable"] == null)
    {
        WebClient webclient = new WebClient();
        StreamReader reader = new
            StreamReader(webclient.OpenRead("http://finance.yahoo.com/m3?u"));
        string rawdata = reader.ReadToEnd();

        int pstart = rawdata.IndexOf("<table border=1");
        int pend = rawdata.IndexOf("</table>", pstart);
        rawdata = rawdata.Substring(pstart, pend-pstart+8);

        rawdata = rawdata.Replace("<table ", "<table runat=\"server\" ");
        HtmlTable table = (HtmlTable)
            this.ParseControl(rawdata).Controls[0];

        Cache["CurrencyTable"] = table;
    }

    HtmlTable CurrencyTable = (HtmlTable) Cache["CurrencyTable"];

    int x = (ddl1.SelectedIndex+1);
    int y = (ddl2.SelectedIndex+1);
    HtmlTableCell cell = CurrencyTable.Rows[y].Cells[x];

    decimal value1 = decimal.Parse(tb1.Text);
    decimal rate = decimal.Parse(cell.InnerText,
        new CultureInfo("en-us"));
    decimal value2 = (value1 * rate);
    lb.Text = value2.ToString();
}
```

```
</script>

<form runat="server">

  <h1> Währungsumrechnung </h1>

  <p>
    <ASP:DropDownList id="ddl1" runat="server">
      <ASP:ListItem Text="U.S. Dollar $" />
      <ASP:ListItem Text="Yen ¥" />
      <ASP:ListItem Text="Euro €" Selected="true" />
      <ASP:ListItem Text="Kanadische Dollar $" />
      <ASP:ListItem Text="Englische Pfund £" />
      <ASP:ListItem Text="Australische Dollar $" />
      <ASP:ListItem Text="Schweizer Franken" />
    </ASP:DropDownList>

    <ASP:TextBox id="tb1" runat="server" Text="1" />
  </p>

  <p>
    <ASP:DropDownList id="ddl2" runat="server">
      <ASP:ListItem Text="U.S. Dollar $" Selected="true" />
      <ASP:ListItem Text="Yen ¥" />
      <ASP:ListItem Text="Euro €" />
      <ASP:ListItem Text="Kanadische Dollar $" />
      <ASP:ListItem Text="Englische Pfund £" />
      <ASP:ListItem Text="Australische Dollar $" />
      <ASP:ListItem Text="Schweizer Franken" />
    </ASP:DropDownList>

    <ASP:Label id="lb" runat="server" />
  </p>

  <ASP:Button runat="server" Text="Umrechnen" OnClick="bt_click" />

</form>
```

Die Abbildung zeigt das Ergebnis. Nach Auswahl der Quell- und Zielwährung sowie Eingabe des Umrechnungsbetrages zaubert ein einziger Button-Klick das Ergebnis zutage. Die Abfrage der Yahoo-Seite erfolgt aus Gründen der Performance nicht bei jeder Umrechnung. Stattdessen wird das erzeugte HTML Server Control im Cache abgelegt. Der Zugriff auf die einzelnen Kurse erfolgt über das Objektmodell des Controls basierend auf dem Index der beiden DropDownList-Controls. Die eingegebenen Werte werden mittels `decimal.Parse` in den gleichnamigen Datentyp umgewandelt. Da die Seite von Yahoo die englische Notation und somit einen Punkt als Dezimaltrennzeichen verwendet, wird der Parse-Methode

ein `FormatProvider` in Form einer Instanz der Klasse `CultureInfo` übergeben. Diese wurde auf Basis der Locale-ID „en-us“ angelegt und arbeitet somit ebenfalls mit der englischen Schreibweise.



Abbildung 17.13 Der Euro gegenüber dem U.S.-Dollar hat fast unbemerkt deutlich zugelegt.



Ich gehe davon aus, dass Sie an entsprechender Stelle um Erlaubnis fragen, bevor Sie die Kurse zu eigenen Zwecken weiterverwenden. Dies gehört nicht nur zum guten Stil, sondern sichert Sie auch in rechtlicher Hinsicht ab. Das hier vorgestellte Listing stellt nur eine prinzipielle technische Erklärung bereit.

Web Service

Wenn ich es mir recht überlege, dann ist eine derartige Funktionalität optimal für einen Web Service ausgelegt, finden Sie nicht? O.K., dann mal los. Das Prinzip bleibt bestehen, die Daten werden bei Yahoo abgefragt, als `HtmlTable-Control` geparkt und von dort aus umgerechnet. Die Unterschiede liegen mehr im Detail.

Zunächst einmal habe ich zwei Web-Methoden vorgesehen. `GetExchangeRate` liefert einen Umrechnungskurs für zwei gegebene Quell- und Zielwährungen. Dieser Methode obliegt es auch, die Daten abzufragen und aufzubereiten. Die zweite Methode `CurrencyConversion` berechnet hingegen einen Betrag auf Basis zweier Währungen und greift dabei auf die Möglichkeiten von `GetExchangeRate` zurück. Die Übergabe der gewünschten Währungen erfolgt über die neue Enumeration `Currency` mit den sieben zur Verfügung stehenden Einheiten.

Listing 17.10 Currency.asmx

```
<% @WebService Class="CurrentCurrencies"
    Language="C#" Debug="true" %>

using System;
using System.Globalization;
using System.IO;
using System.Net;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.Services;

[WebService]
public class CurrentCurrencies : WebService
{
    [WebMethod]
    public Decimal GetExchangeRate(Currency sourceCurrency,
        Currency destinationCurrency)
    {
        if(Context.Cache["CurrencyTable"] == null)
        {
            WebClient webclient = new WebClient();
            StreamReader reader = new
                StreamReader(webclient.OpenRead("http://finance.yahoo.com/m3?u"));
            string rawdata = reader.ReadToEnd();

            int pstart = rawdata.IndexOf("<table border=1");
            int pend = rawdata.IndexOf("</table>", pstart);
            rawdata = rawdata.Substring(pstart, pend-pstart+8);

            rawdata = rawdata.Replace("<table ", "<table runat=\"server\" ");
            HtmlTable table = (HtmlTable)
                (new Page()).ParseControl(rawdata).Controls[0];

            Context.Cache["CurrencyTable"] = table;
        }

        HtmlTable CurrencyTable = (HtmlTable)
            Context.Cache["CurrencyTable"];

        int x = ((int) sourceCurrency) + 1;
        int y = ((int) destinationCurrency) + 1;
        HtmlTableCell cell = CurrencyTable.Rows[y].Cells[x];

        return(decimal.Parse(cell.InnerText, new CultureInfo("en-us")));
    }
}
```



```

[WebMethod]
public Decimal CurrencyConversion(decimal value,
    Currency sourceCurrency, Currency destinationCurrency)
{
    decimal rate = this.GetExchangeRate(sourceCurrency,
        destinationCurrency);
    return(value * rate);
}
}

public enum Currency
{
    USDollar,
    Yen,
    Euro,
    CanadianDollar,
    EnglishPound,
    AustralianDollar,
    SwissFranc
}

```

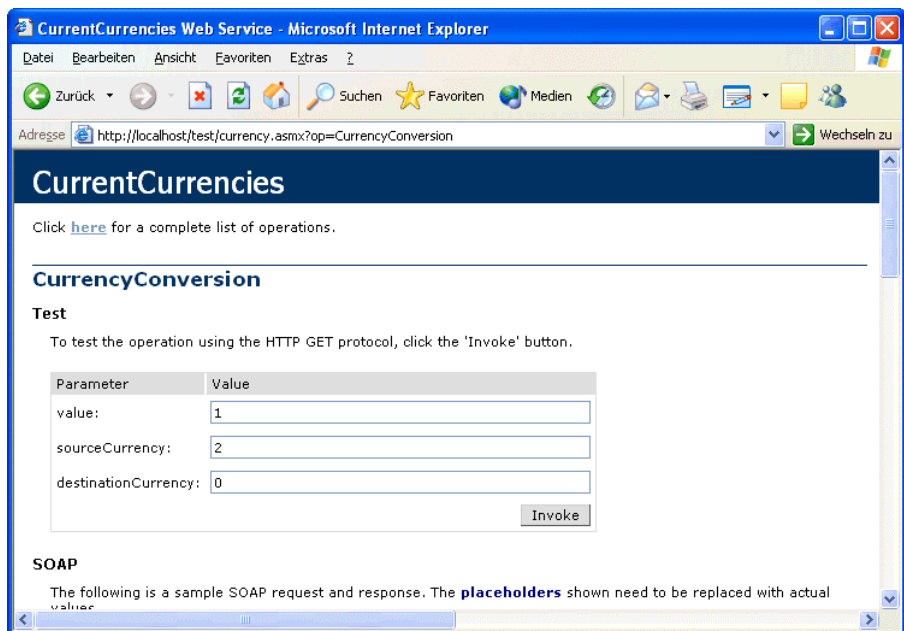


Abbildung 17.14 Der Web Service ermöglicht eine einfache Währungsumrechnung.

Zwei Dinge fallen gegenüber der zuvor gezeigten Implementierung auf. Zum einen steht die Klasse `Cache` nicht direkt zur Verfügung, sondern muss über `Context` angefordert werden. Zum anderen handelt es sich um einen Web Service und nicht um eine reguläre ASP.NET-Seite. Die für die Methode `ParseControl` benötigte Instanz der Klasse `Page` steht daher nicht zur Verfügung. Auch dies ist kein Problem, denn die Klasse lässt sich ohne Weiteres zu diesem Zweck neu instanziiieren.

That's it, der Web Service steht und kann nach Einbindung mittels `wsdl.exe` direkt verwendet werden. Denken Sie aber bitte unbedingt an die rechtliche Situation, auf die ich weiter oben hingewiesen habe.

17.9 ... Preise in verschiedenen Währungen anzeigen?

Nachdem Sie im vorherigen Rezept gesehen haben, wie einfach sich aktuelle Währungsinformationen abfragen lassen, möchte ich Ihnen nun eine abgewandelte Version der Produktübersicht zeigen, die die Preise in den unterschiedlichen Währungen bereithält. Die Basis bildet der vorgestellte Web Service, so dass die eigentliche Abfrage der Kursdaten ausgelagert wird.

Zunächst muss der Web Service eingebunden werden. Dies geschieht wie üblich über das Kommandozeilenprogramm `wsdl.exe`. Die so erzeugte Quellcode-Datei mit dem Namen `CurrentCurrencies.cs` wird über die `@Assembly`-Direktive eingebunden. Die Umrechnung selbst erfolgt auf Basis der Benutzerauswahl innerhalb eines `DropDownList`-Controls über die Methode `GetPrice`. Der Umrechnungskurs wird von der Eigenschaft `Rates` abgefragt, die ein `decimal`-Array liefert. Die enthaltenen Daten stammen aus dem `Cache` und werden nur bei Bedarf über den Web Service angefordert.

Listing 17.11 Products3.aspx

```
<% @Assembly Src="CurrentCurrencies.cs" %>

...

decimal GetPrice(decimal euroValue)
{
    int current = ddl_currency.SelectedIndex;
    return(euroValue * this.Rates[current]);
}

decimal[] Rates
{
    get
    {
        if (Cache["Rates"] == null)
```

```

{
    CurrentCurrencies cc = new CurrentCurrencies();
    decimal[] rates = new decimal[7];
    for(int i=0; i<7; i++)
    {
        rates[i] = cc.GetExchangeRate(Currency.Euro, (Currency) i);
    }
    Cache["Rates"] = rates;
}
return((decimal[]) Cache["Rates"]);
}
}

...

<ASP:DataList id="products" runat="server"
    RepeatColumns="2">

    <HeaderTemplate>
        <h2>Bitte wählen Sie das gewünschte Produkt aus</h2>
    </HeaderTemplate>

    <ItemTemplate>
        <ASP:HyperLink runat="server"
            Text=<%=# DataBinder.Eval(Container.DataItem, "Name") %>
            NavigateUrl=<%=# "product.aspx?id=" +
                DataBinder.Eval(Container.DataItem, "ID") %>/>
        <br>
        <%=# DataBinder.Eval(Container.DataItem, "Description") %>
        Preis: <%=# GetPrice((decimal) DataBinder.Eval(Container.DataItem,
            "Price")).ToString("0.00") %><br>
    </ItemTemplate>

</ASP:DataList>

</form>

```

Das Listing zeigt nur die geänderten Ausschnitte gegenüber der einfachen Produktübersicht aus dem Rezept „... eine hierarchische Produktübersicht erzeugen?“.



Abbildung 17.15 Die Preise werden in U.S.-Dollar ausgegeben.



Index

•

.NET Reflector 140

@

@@Identity 405

@Control

 ClassName-Attribut 364, 370

 Inherits-Attribut 344

 Src-Attribut 344

@Implements 373

@Import 98

@Page

 Culture-Attribut 71

 Inherits-Attribut 89

 Language-Attribut 85

 Src-Attribut 88

€

€ 69, 771

A

Access Control List 230

ACL 230

AGB 206

alert 243

Algorithmen 535

Allgemeine Geschäftsbedingungen 206

ALTER TABLE 400

Apache 608

Array

 Contains 112

 Element enthalten 112

 inline erstellen 118

 sortieren 114

ArrayList

 Contains 112

as 110

ASP.NET-Engine

 anderen Server verwenden 608

 Neustart konfigurieren 606

 registrieren 607

aspnet_regiis.exe 607

Assembly

 KeyFile 105

 strongly named 101, 103

 Version 105

Ausführung unterbrechen 30

Ausgabe verzögern 60

AutoGenerateColumns 276

AutoPostBack 240

B

Base64 647

BaseDataList 274

Baseline Security Analyser 605

BaseValidator 207

Batch

 kompilieren 93

Besucher

- Eingabe erzwingen 199
- IP-Adresse ermitteln 609
- Sprache erkennen 72
- Betriebssystem abfragen 601
- Bitmap 546
 - GetThumbnailImage 549
 - RotateFlip 555
- BoundColumn 313
 - individuell formatieren 279
- Breakpoint 29
- Broken Link
 - abfangen 77
 - bei Bildern 81
 - finden 79
 - Xenu Link Sleuth 80
- Browser
 - Frames 50
 - MessageBox anzeigen 243
 - Möglichkeiten ermitteln 46
 - Seiten optimieren 48
- Browsercaps
 - aktualisieren 48
- Button
 - in DataGrid 285
- ButtonColumn 313

C

- C#
 - as 110
 - is 110
 - IsDate 156
 - IsNumeric 154
 - params 119, 122
 - using 125
 - With 124
- Calendar 183
- CausesValidation 200
- Celsius 190
- char
 - IsNumber 163
- CheckBox
 - in DataGrid 303

- TextAlign 129
- validieren 206
- CheckBoxList
 - sortieren 256
- CheckBoxValidator 207
- Class Browser 141
- CLR
 - Version abfragen 602
- Code Behind
 - automatisch ableiten 94
 - globale Seitenvorlage 35
 - kompilieren 88
 - mehrere Ableitungen 38
 - User Control 344, 370
 - Web Service 627
- Collection
 - eigene erstellen 142
 - strongly typed 142
- CollectionBase 142
- COM
 - Excel verwenden 664
 - Komponente einbinden 570
 - mit visual Studio .NET verwenden 572
- combit
 - address manager 715
 - List & Label 670
- Community 688
 - Forum 746
 - Gästebuch 739
 - Link-Sammlung 719
 - neue Inhalte markieren 696
 - Newsletter 701
 - News-System 688
 - Postkarten/eCards versenden 727
 - Zufallslink anzeigen 725
- Compare 117
- CompareTo 115
- CompareValidator 200
- Compiler 88
- Computername 596
- Connection

- lokal und beim Provider 392
- Content-Management
 - Inhalte übernehmen 676
 - Linkzähler 683
 - PDF-Datei erzeugen 670
 - Tabelle parsen 679
- Content-Types
 - Übersicht 83
- Context
 - aus Klasse zugreifen 150
 - Current 150
 - Handler 55
 - RewritePath 534
- ControlPropertiesValid 207
- Cookie
 - Annahme überprüfen 43
 - Session ohne 498
- COUNT 412, 438
- CREATE TABLE 397
- CreateChildControls 259, 361
- CreateObject 570
- CryptoStream 536
- csc.exe 88
 - Optionen ändern 100
- CultureInfo 70
- CustomValidator 213
- D**
- DAS 535
- DataBinder.Eval 271
 - mit Web Service Proxy verwenden 645
- DataColumn
 - dynamisch anlegen 283
- DataField 279
- DataFields 320
- DataFormatString 279, 320
- DataGrid
 - AllowCustomPaging 331
 - auf Click-Ereignisse reagieren 285
 - auf TextBox zugreifen 294
 - AutoGenerateColumns 276
 - automatische Spalten anpassen 276
 - Bearbeitungsmodus verlassen 289
 - Button 285
 - CheckBox 303
 - CurrentPageIndex 327
 - Datensatz-ID mitführen 273
 - editieren 291, 294
 - Eingaben validieren 301
 - gebundene Spalte formatieren 279
 - hierarchischen Datensatz anzeigen 321, 426
 - ID mitführen 273
 - ItemCreated 277
 - ItemDataBound 278, 284
 - mehrere Felder in einer Spalte 281, 317
 - mehrzeilige TextBox 300
 - neue Spaltentypen entwickeln 317
 - PageSize 329
 - seitenweise darstellen 327, 330
 - Selektion aufheben 288
 - sortieren 324
 - Spalten dynamisch anfügen 313
 - TemplateColumn 281
 - TextBox formatieren 299
 - Vorlage dynamisch laden 374
 - Zahlen und Dati formatieren 271
- GridColumn 317
- DataItem 271
- DataKeyField 274
- DataKeys 274
- DataList
 - Vorlage dynamisch laden 374
 - Zahlen und Dati formatieren 271
- DataReader
 - GetSchemaTable 396
- DataRelation 321, 423
- DataRow
 - GetChildRows 425, 429
- DataSet
 - filtern 433
 - in Session ablegen 503

- per Web Service übertragen 639
- Relationen herstellen 423
- sortieren 422
- strongly typed 384
- DataTable
 - DefaultView 422
 - Spalte dynamisch anlegen 283
- DataTextFormatString 238
- DataTypeCheck 200
- DataGridView
 - filtern 433
 - Sort 422
- Datei
 - binär erstellen oder ergänzen 465
 - binär lesen 463
 - binäre Daten im Web Service übertragen 647
 - Download verhindern 528
 - Existenz prüfen 458
 - exklusiv sperren 489
 - kurzen Namen abfragen 482
 - mit Wildcards suchen 456
 - packen und entpacken 470
 - PDF downloaden 676
 - PDF erzeugen 670
 - per Email versenden 479
 - per Web Service übertragen 647
 - serverseitig herunterladen 611
 - temporäre erstellen 483
 - Text auslesen 460
 - Text erstellen oder ergänzen 462
 - umbenennen 459
 - verknüpftes Icon anzeigen 455
 - verschlüsseln 535
 - Version ermitteln 487
 - vor direktem Zugriff schützen 528
 - vorhandene anzeigen 453
 - ZIP 470
 - zum Client senden 457
- Dateisystem
 - aktuelles Verzeichnis ermitteln 449
 - Änderungen überwachen 466
 - binäre Datei auslesen 463
 - binäre Datei erstellen oder ergänzen 465
 - Datei umbenennen 459
 - Datei zum Client senden 457
 - Dateien anzeigen 453
 - Dateien komprimieren 470
 - Dateien mit Wildcards suchen 456
 - Datei-Icon anzeigen 455
 - Elemente eines Pfades ermitteln 485
 - Existenz von Datei oder Verzeichnis prüfen 458
 - Pfadelemente verbinden 485
 - Programm starten 576
 - temporäre Datei erstellen 483
 - Textdatei auslesen 460
 - Textdatei erstellen oder ergänzen 462
 - Verzeichnis löschen 486
 - Verzeichnisliste erstellen 450
 - virtuellen Pfad umwandeln 448
 - ZIP-Datei 470
- Datenbank
 - Aggregatfunktion 437
 - Anzahl Datensätze ermitteln 412
 - Autowert abfragen 405
 - Datensatz per SQL aktualisieren 407
 - Datensatz per SQL anlegen 402
 - Datensatz per SQL löschen 410
 - Datensätze sortieren 419
 - hierarchischen Datensatz anzeigen 426
 - ID eines neuen Datensatzes abfragen 405
 - Memo-Feld darstellen 438
 - relational verknüpfte Datensätze anlegen 430
 - Relationen herstellen 423
 - Relationen visualisieren 426
 - Schema abfragen 395
 - Spalte per SQL anlegen 399
 - Tabelle per SQL anlegen 397

- vor SQL Injection schützen 415
- zufälligen Datensatz abfragen 441

Datensätze sortieren 419

DateTime

- DayOfWeek 182
- DaysInMonth 186
- Hour 181
- Now 178
- ToString 178

Datum

- aktuelle Zeit ermitteln 178
- Anzahl Tage im Monat ermitteln 186
- der letzten Änderung 189
- in der Zukunft 203
- in Vorlagen formatieren 271
- Kalenderwoche ermitteln 183
- Schaltjahr abfragen 187
- validieren 202
- Wochentag ermitteln 181

DayOfWeek 182

Debugging 26

- mit Visual Studio .NET 29

DELETE 410

DES 535

DESCryptoServiceProvider 536

DetailColumn 321

Dictionary

- eigenes erstellen 145

DictionaryBase 145

DIN

- EN ISO 3166-1 1998-04 220

Directory

- Exists 458
- GetDirectories 450
- GetFiles 453, 456, 565

Dispose 126

DllImport-Attribut 456, 522, 573

Dns

- Resolve 610

DropDownList

- Eintrag über Inhalt selektieren 262
- mehrere Felder anzeigen 238

- mehrstufige Auswahl 216

- Selektion erhalten 263

- sortieren 256

DSN 388

E

E-Card 727

E-Commerce

- EU-Land 769

- Euro 771

- Preise in verschiedenen Währungen 791

- Produktübersicht 764

- Umsatzsteuer-ID online überprüfen 779

- Umsatzsteuer-ID syntaktisch überprüfen 775

- Umsatzsteuerpflicht 771

- variable Preisanzeige 774

- Währungsinformationen abfragen 785

EditCommandColumn 294, 313

EditItemTemplate 304

Eigenschaften

- alle abfragen 131

- indizierte abfragen oder setzen 136
- per Reflection abfragen oder setzen 133

Email

- auf Echtheit überprüfen 514, 708

- Datei uploaden und versenden 230

- Emailing versenden 711

- Formular weiterleiten 221

- Linkzähler für Emailing 683

- Newsletter 701

- personalisieren 711

- Postkarten versenden 727

- validieren 205

- verschlüsseln 540

- Vorlage erstellen 234

- ZIP-Datei erstellen und versenden 479

- Enum
 - IsDefined 130
- Enumeration
 - alle Werte auflisten 130
 - einzelne Elemente abfragen 127
 - Flags 128
 - IsDefined 130
 - Wert aus Zeichenkette ermitteln 129
- Enums
 - GetNames 130
- Environment
 - GetEnvironmentVariable 596, 599
 - GetFolderPath 597
 - MachineName 596
 - OSVersion 601
 - SpecialFolder 597
 - TickCount 598
 - Version 602
- Ereignisprotokoll
 - anzeigen 589
 - Eintrag anlegen 595
- Euro 771
 - Zeichen ausgeben 69
- EvaluateIsValid 207
- EventArgs 354
- EventLog
 - GetEventLogs 589
 - WriteEntry 595
- Excel 664
- Execute 576
- ExecuteNonQuery
 - Anzahl betroffener Datensätze 414
 - Tabelle per SQL anlegen 397
- F**
- Fahrenheit 190
- Fehlerbehandlung 40
- File
 - AppendText 462
 - Close 466
 - Create 465
 - CreateText 462
 - Exists 458
 - GetLastWriteTime 189
 - Move 460
 - OpenRead 463
 - OpenText 460
 - OpenWrite 465
- FileStream 463
 - Lock 489
 - Unlock 489
- FileSystemWatcher 466
- FileVersionInfo 488
- Fill 414
- FindByValue 262
- FindControl 379
- Forms Authentication
 - alle Dateien schützen 526
 - Impersonation 524
 - mit Windows überprüfen 522
- FormsAuthentication
 - HashPasswordForStoringInConfigFile 521
- Formular
 - Absenden bestätigen lassen 248
 - aus Zeichenkette parsen 377
 - CheckBox validieren 206
 - Datei-Upload 225
 - editierbares DataGrid erstellen 291
 - Eingabe erzwingen 199
 - Eingabefeld schreibschützen 215
 - Eingabefelder in User Controls 350
 - Eingabelänge begrenzen 270
 - Eingaben in DataGrid validieren 301
 - Email validieren 205
 - formatiertes Eingabefeld 254
 - JavaScript vor Absenden ausführen 245
 - Kontaktformular erstellen 221
 - Länderliste anzeigen 219
 - Listen sortieren 256
 - mehrere Felder in Liste anzeigen 238
 - mehrstufige Auswahl 216
 - per Email weiterleiten 221

- Selektion behalten 263
- TextBox mit Farbverlauf 268
- Upload ergibt immer null 266
- validieren 221
- Forum 746
- Frames 50
- G**
- GAC
 - Assembly ablegen 103
 - Assembly einbinden 101
 - Assembly löschen 107
 - Sprache registrieren 86
- Gargabe Collection
 - using 126
- Gästebuch 739
- GetLastError 41
- GetObjectData 509
- GetSchemaTable 396
- GetShortPathName (Win32 API) 482
- GetTickCount 599
- GetType 111
- GetVersionInfo 488
- GetWeekOfYear 183
- Global Assembly Cache
 - Assembly ablegen 103
 - Assembly einbinden 101
 - Assembly löschen 107
 - Sprache registrieren 86
- Globalization 70
- Grafik
 - Auflösung ermitteln 548
 - dynamisch erstellen 556
 - einlesen und ausgeben 546
 - Format konvertieren 548
 - Größe ermitteln 548
 - per Web Service generieren 649
 - rotieren 555, 563
 - Text ausgeben 558
 - Text formatieren 561
 - Text rotieren 563
 - Textur verwenden 562
 - Thumbnail zwischenspeichern 550
 - Thumbnails erzeugen 549
 - uploaden und speichern 553
 - vor Übernahme schützen 534
- Graphics 556
 - Clear 556
 - DrawString 560
 - FromImage 556
 - RotateTransform 563
- GregorianCalendar 183
- GUID 75
- H**
- HashCode 520
- Hilfsfunktionen
 - global hinterlegen 147
- HtmlInputFile 225, 266, 553
- HtmlTable 680
- HtmlTextWriter 65, 253
- HTTP 30
- HttpBrowserCapabilities 46
- HttpContext siehe Context
- HttpModule 531
- HttpPostedFile 226
- HttpRequest siehe Request
- HttpResponse siehe Response
- HTTPS 513
- HttpServerUtility siehe Server
- HttpSessionState siehe Session
- HttpWebRequest
 - Create 613
- HyperLinkColumn 313
- I**
- IANA 83
- ICollection 142, 145
- IComparable 115
- IComparer 117, 259
- IDictionary 145
- IDisposable 126
- IEnumerable 142
 - string 160

if

- inline ausführen 119

IIf 121**IIS**

- Fehler 404 konfigurieren 78
- Lockdown Tool 605
- Version abfragen 603
- vor Viren schützen 604
- Windows Authentication konfigurieren 636

IList 142

- Contains 112

In/Out 578**InitializeAsUserControl** 366**inline**

- Array erstellen 119
- if-Abfrage 119

InputStream 554**INSERT INTO** 402**InstantiateIn** 376**int**

- Parse 201

Internet Explorer

- Proxy 30
- Web Service Behaviour verwenden 625

IP-Adresse 609

- auflösen 610

IPHostEntry 610**is** 110**IsDate** 156**ISerializable** 509**IsNumeric** 154**ISO**

- 3166-1 1997 220

IsPostBack 265**ItemCreated** 276**ItemDataBound** 276**ITemplate** 376**ItemTemplate** 272, 690**J****JavaScript**

- alert 243, 245
- Formular bestätigen 248
- Redirect 53
- Unterstützung überprüfen 44
- vor Absenden eines Formulars ausführen 245

K**Kalenderwoche** 183**Klasse**

- auf ASP-NET-Objekte zugreifen 150
- in Session ablegen 506
- individuell serialisieren 509

Kommunikation belauschen 30**Kompilieren**

- Batch 93
- Code Behind 88
- Optionen ändern 100

Komponenten

- Detailinformation ermitteln 140

Kopfzeilen

- auslesen 32
- Content-Disposition 458, 530
- ContentType 458, 530, 547
- HTTP_REFERER 531
- Redirect 52
- Refresh 61
- UserAgent 46

Kryptografie 535**Kurse** 785**L****Länderliste** 219**Library**

- kompilieren 89

Link Sleuth 80**Link-Sammlung** 719**Linux** 609**List & Label** 670**ListBox**

- Eintrag über Inhalt selektieren 262
- mehrere Felder anzeigen 238
- mehrstufige Auswahl 216
- Selektion erhalten 263
- sortieren 256
- ListItemCollection
 - FindByValue 262
- LoadControl 367
- LoadTemplate 374
- Lockdown Tool 605
- LogonUser 522, 573

M

- machine.config
 - Engine-Neustart konfigurieren 606
 - ProcessModel konfigurieren 581, 588
 - Sprache ändern 85
 - Web Service Hilfeseite ändern 655
- MailMessage 230, 234
- MailMessageExt 234
- MapPoint .NET Web Services 655
- Math
 - Ceiling 193
 - Even 195
 - Floor 193
 - Max 194
 - Min 194
 - Odd 195
 - Round 192
- MaximumValue 204
- MaxLength 270
- MD5 521, 535
- MessageBox 243
- META-Tags 64
- Methoden
 - global hinterlegen 147
 - per Reflection aufrufen 138
- MethodInfo
 - Invoke 138
- Microsoft Baseline Security Analyser 605

- Microsoft.Data.Odbc 391
- MinimumValue 204
- Mnemonisches Passwort 517
- Mono-Projekt 608
- MultipleFieldsColumn 317

N

- Namespaces
 - importieren 97
- NamingContainer 376
- NETBIOS-Name 596
- NetworkCredential 637
- Netzwerk
 - Datei herunterladen 611
 - Host-Namen auflösen 610
 - IP-Adresse ermitteln 609
 - Ping 578
- Newsgroup 746
- Newsletter 701
- News-System 688

O

- ODBC 388
- OleDbCommand
 - Parameters 418
- OleDbDataAdapter
 - Fill 333, 414
 - RowUpdated 432
- OnPreRender 339
- Open Directory Project (ODP) 719
- Operatoren
 - as 111
 - is 111
- ORDER BY 421

P

- P/Invoke 573
 - Excel verwenden 664
- Page
 - FindControl 379
 - IsClientScriptBlockRegistered 243
 - LoadControl 367

- LoadTemplate 374
- ParseControl 378, 681, 791
- RegisterClientScriptBlock 243
- Paging
 - automatisches 327
 - manuelles 330
- ParamArray 124
- Parameter
 - unbekannte Anzahl übergeben 122
- Parameters-Collection 418
- params 119, 122
- ParseControl 378
- Passwort
 - erstellen 515
 - Hash erstellen 520, 732
- Path 485
 - GetDirectoryName 449
 - GetTempFileName 483, 667
 - GetTempPath 484
 - Umgebungsvariable 90
- PDF
 - downloaden 676
 - erzeugen 670
- Ping 578
- Placeholder 63, 428
- PostBack 250
- Postkarte 727
- Process
 - GetProcessById 579
 - GetProcesses 578
 - Start 576
- Produktübersicht 764
- Programm starten 576
- PropertyInfo
 - GetValue 135, 347
 - SetValue 134, 347
- Proxy 30

Q

- Query-String
 - abfragen 108

R

- RadioButtonList
 - sortieren 256
- Random 565
- RangeValidator 204
- RC2 535
- Redirect 52
 - für anderen Frame 53
 - serverseitig 55
- Referer 59
- Reflection
 - alle Eigenschaften abfragen 131
 - Eigenschaften abfragen oder setzen 133
 - indizierte Eigenschaft abfragen oder setzen 136
 - Methode aufrufen 138
 - Zugriff auf User Control 368
- Reflector 140
- Refresh 61
- Regex siehe Reguläre Ausdrücke
 - IsMatch 617
 - Replace 174, 678
- Reguläre Ausdrücke 173
 - Beispiele 212
 - Bildpfade ersetzen 363
 - Email validieren 205
 - Email validieren (Web Service) 616
 - validieren 210
 - Zeichen ersetzen 173
- RegularExpressionValidator 210
- Relationen 423
- Repeater
 - Vorlage dynamisch laden 374
 - Zahlen und Dati formatieren 271
- Request
 - aktuelle Seite ermitteln 57
 - AppendHeader 61
 - aus Klasse zugreifen 150
 - IsSecureConnection 513
 - PhysicalApplicationPath 449
 - PhysicalPath 449

- QueryString 108
- Referer 59
- UserHostAddress 609
- UserLanguages 72
- vorherige Seite ermitteln 59
- RequiredFieldValidator 199
- Response
 - aus Klasse zugreifen 150
 - ContentType 83, 458, 530, 547
 - End 676
 - IsClientConnected 76
 - OutputStream 547, 612
 - Redirect 52, 676, 684
 - verzögern 60
 - Write, Ersatz für 62
 - WriteFile 457, 534, 611, 667, 676
- Rijndael 535
- RSA 535

S

- Schaltjahr 187
- Scrapen 676
- Seite
 - automatisch ableiten 94
 - Eurozeichen ausgeben 69
 - globale Methoden 148
 - Linkzähler 683
 - mehrere Sprachen verwenden 87
 - MessageBox anzeigen 243
 - META-Tags 64
 - neu laden 61
 - neue Inhalte markieren 696
 - Variablen speichern 250
 - verzögern 60
 - von anderer Website übernehmen 676
 - Vorlage konfigurieren 94
 - Zufallslink anzeigen 725
- Seitenvorlage 35
- SELECT @@Identity 405
- Serializable-Attribut 343, 508, 643
- SerializationInfo 509

- Server
 - Computernamen abfragen 596
 - Ereignisprotokoll anzeigen 589
 - Ereignisprotokoll ergänzen 595
 - Execute 57
 - GetLastError 41
 - MapPath 392
 - Prozess neu starten 576
 - Prozesse anzeigen 578
 - Service anzeigen 582
 - Service starten oder beenden 584
 - Transfer 55, 684
 - Umgebungsvariablen 596, 599
 - Uptime ermitteln 598
- Server.MapPath 448
- ServerValidate 213
- Server-Variablen
 - auslesen 32
 - HTTP_REFERER 79
 - HTTPS_KEYSIZE 514
 - QUERY_STRING 108
 - SERVER_SOFTWARE 604
- ServiceController 582
 - GetServices 582
 - Start, Stop 584
 - WaitForStatus 587
- Session
 - aus Klasse zugreifen 150
 - DataSet ablegen 503
 - Email auf Echtheit überprüfen 514
 - Klasse ablegen 506
 - Objekt individuell serialisieren 509
 - ohne Cookies 498
 - SQL-Server 502
 - State-Service 499
 - strongly typed 492
 - typensicherer Zugriff 492
 - Web Service 628
- SHA1 521, 535
- Shop siehe E-Commerce
- Sicherheit

- alle Dateien vor Zugriff schützen 526
- Bestätigungstext als Grafik anzeigen 558
- Bilder schützen 534
- Datei oder Stream verschlüsseln 535
- Datei vor direktem Zugriff schützen 528
- Email auf Echtheit überprüfen 708
- Email verschlüsseln 540
- Hashcode erstellen 520
- Links verhindern 531
- Passwort erstellen 515
- sichere Verbindung überprüfen 513
- Web Service mit Passwort schützen 635
- Web Service mit Passwort verwenden 637
- Web Service per SSL schützen 639
- Sleep 60
- SmartNavigation 240
- sn.exe 105
- SoapHttpClientProtocol 624
 - AllowAutoRedirect 631
 - PreAuthenticate 638
- SortedListControls 256
- SortExpression 326
- Sprache
 - mehrere in einer Seite 87
 - Standard ändern 85
- SQL
 - Aggregatfunktion 437
 - ALTER TABLE 400
 - Anzahl Datensätze ermitteln 412
 - Autowert abfragen 405
 - COUNT 412
 - CREATE TABLE 397
 - Datensatz aktualisieren 407
 - Datensatz löschen 410
 - DELETE 410
 - ID eines neuen Datensatzes abfragen 405
 - Injection verhindern 415
 - INSERT INTO 402
 - neue Spalte anlegen 399
 - neue Tabelle anlegen 397
 - neuen Datensatz anlegen 402
 - NEWID (SQL-Server) 442
 - ORDER BY 421
 - SELECT @@Identity 405
 - Update 407
 - zufälligen Datensatz abfragen 442
- SqlCommand
 - Parameters 418
- SqlDataAdapter
 - RowUpdated 432
- SQL-Server 502
- SSL 513
- Standard-In/Out 578
- StateBag 250, 340
- State-Service 499
- StreamReader 227, 461
- StreamWriter 462
- string siehe Zeichenkette
 - Indexer 161
 - IndexOf 158
 - Join 169
 - Replace 440
 - Split 168
 - ToCharArray 162
- string.Format
 - Währung formatieren 70
- StringWriter 253
- strongly typed DataSet 384
- System
 - Web Service-Hilfeseite ändern 655
- System.Data 384
- System.Data.OleDb 384
- System.Data.SqlClient 384
- System.Diagnostics 488, 576, 589
- System.Drawing 455, 546
- System.Drawing.Drawing2D 546
- System.Drawing.Imaging 546
- System.Drawing.Text 546

- System.Globalization
 - CultureInfo 70, 73, 179
- System.IO 448
- System.Net 609, 676
- System.Reflection 131
- System.Runtime.InteropServices 456
- System.Security.Cryptography 535
- System.ServiceProcess 582
- System.Text.RegularExpressions 173, 211
- System.Web.Mail 230
- System.Web.SessionState 492
- System.DSN 388

T

- Tabelle parsen 679
- TCP/IP 30
- TemplateColumn 297, 304, 313, 428
- TextBox
 - Eingabelänge begrenzen 270
 - formatieren 254
 - in DataGrid 294
 - in DataGrid formatieren 299
 - MaxLength 270
 - mehrzeilig in DataGrid 300
 - mit Farbverlauf 268
 - schreibschützen 215
- TextureBrush 562
- Textverarbeitung 254
- Thread
 - Sleep 60
- Thumbnail
 - automatisch erzeugen 550
 - erzeugen 549
 - zwischenspeichern 550
- Ticks 598
- tlbimp.exe 571, 665
- Tracing 26
 - mit Visual Studio .NET 29
- TripeDES 535
- Type
 - GetProperties 131, 134

- GetProperty 347
- Type Library 571
- Typen
 - sichere Konvertierung 110
- typeof 129

U

- Umgebungsvariablen
 - abfragen 599
 - Path 90
 - Windows-Verzeichnis 596
- Umsatzsteuer-ID 775, 779
- Universal Description Discovery Integration (UDDI) 656
- unmanaged Code 575
- unsafe 575
- UPDATE 407
- Upload 225
 - abspeichern 229
 - ergibt immer null 266
 - Grafik uploaden und speichern 553
 - per Email versenden 230
- User Control
 - aus Zeichenkette parsen 377
 - automatisch ableiten 97
 - Code Behind 370
 - Code Behind verwenden 344
 - dynamisch laden 367
 - dynamisch platzieren 364
 - Eigenschaften 336
 - Eingabefelder verwenden 350
 - Ereignisse verwenden 351
 - im Browser anzeigen 381
 - InitializeAsUserControl 366
 - komplexes Control erstellen 358
 - ohne Seite 381
 - OnPreRender 339
 - Schnittstelle unterstützen 372
 - Status speichern 340
 - Werte übergeben 336
- UserAgent 46
- using 125

V

- ValidationExpression 210
- Validierung
 - CheckBox 206
 - Datum 202
 - Datum in Zukunft 203
 - Eingabe erzwingen 199
 - Eingaben in DataGrid 301
 - Email-Adresse 205
 - individuelle Eingabe 210
 - numerischer Wert 200
 - Schema 210
- ValueToCompare 203
- Variable
 - Typ ermitteln 111
- Vat-ID 775, 779
- vbc.exe 88
- Verschlüsselung 535
- Version
 - CLR 602
 - IIS 603
 - Windows 601
- Verzeichnis
 - aktuelles ermitteln 449
 - Dateien anzeigen 453
 - Existenz prüfen 458
 - Liste erstellen 450
 - löschen 486
 - Windows-Verzeichnis ermitteln 596
- Verzögern 60
- ViewState 250, 340
- Visual Basic .NET
 - IIf 121
 - IsDate 156
 - IsNumeric 154
 - ParamArray 124
 - With 124
- Vorlage
 - dynamisch laden 374
 - programmatisch anlegen 376

W

- Währungsinformationen 785
- Web Control
 - aus Zeichenkette parsen 377
- Web Controls
 - Reihenfolge festlegen 198
 - Tabelle parsen 679
 - zur Laufzeit rendern 253
- Web Matrix
 - Class Browser 141
- Web Service
 - Behaviour einsetzen 625
 - Bilder generieren 649
 - binäre Daten übertragen 647
 - Code Behind 627
 - Counter realisieren 632
 - DataBinder.Eval verwenden 645
 - DataSet übertragen 639
 - einfachen Dienst erstellen 616
 - Hilfeseite ändern 655
 - im UDDI-Verzeichnis finden 656
 - komplexe Daten übertragen 639
 - MapPoint .NET 655
 - mit Visual Studio .NET konsumieren 619
 - Newsletter An- und Abmeldung 715
 - ohne Visual Studio .NET konsumieren 622
 - passwortgeschützt erstellen 635
 - passwortgeschützt verwenden 637
 - Session ohne Cookie 630
 - Session verwenden 628
 - SSL/HTTPS verwenden 639
 - Umsatzsteuer-ID überprüfen 781
 - Währungskurse 788
 - Windows Authentication 636
 - Wrapper-Klasse 622
- web.config
 - Applikationseinstellungen 393
 - Assembly einbinden 102, 582
 - Code Behind-Klasse festlegen 94
 - Debugging einschalten 26

- Fehler 404 konfigurieren 77
- Fehlerbehandlung 40
- HttpModule konfigurieren 533
- individuelle Konfigurationsdaten 393
- Konfiguration der Datenbank-Connection 393
- pageBaseType 94
- Session konfigurieren 499
- Session State Service konfigurieren 500
- Sprache ändern 86
- SQL-Server für Session konfigurieren 503
- userControlBaseType 97
- WebClient 611
- WebMethod-Attribut
 - EnableSession 629
- WebRequest 613, 676
- Website
 - Community 688
 - Counter als Web Service 632
 - Forum 746
 - Gästebuch 739
 - Inhalte übernehmen 676
 - Link-Sammlung 719
 - Linkzähler 683
 - neue Inhalte markieren 696
 - Newsletter 701
 - News-System 688
 - Postkarten/E-Cards versenden 727
 - Produktübersicht 764
 - Shop siehe E-Commerce
 - strukturieren 34
 - Zufallslink anzeigen 725
- WebWasher 43
- Win32 573
- Windows
 - Version abfragen 601

- Windows-Verzeichnis 596
- With 124
- Wochentag 181
- wsdl.exe 622

X

- Xenu Link Sleuth 80

Z

Zahlen

- gerade ermitteln 195
- größere ermitteln 194
- in Vorlagen formatieren 271
- kleinere ermitteln 194
- nächste Ganzzahl 193
- runden 192
- ungerade ermitteln 195

Zeichenkette

- Anzahl Wörter ermitteln 166
- Art eines Zeichens ermitteln 163
- aufsplitten 168
- aus Zahl umwandeln 171
- ersetzen mit regulären Ausdrücken 173
- grafisch ausgeben 558
- grafisch formatieren 561
- ist Datum 156
- ist numerisch 154
- rotieren 563
- Swap 157
- Tabelle parsen 679
- Teile vertauschen 157
- Vorkommen zählen 164
- Wörter abfragen 158
- Zeichen iterieren 160
- zusammenführen 169
- zusammensetzen 167
- Zufälliges Passwort 516
- Zufallslink anzeigen 725

