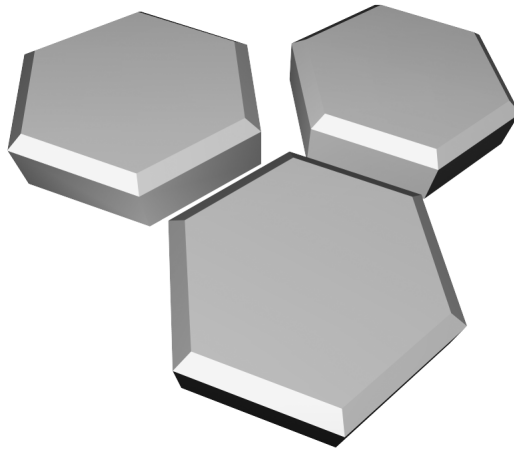


Das C# Codebook



Jürgen Bayer

Das C# Codebook

 ADDISON-WESLEY

An imprint of Pearson Education

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

Bibliografische Information Der Deutschen Bibliothek

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

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

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

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

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

Umwelthinweis:

Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.

10 9 8 7 6 5 4 3 2 1
05 04 03

ISBN 3-8273-2050-X

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

Korrektorat: Simone Meißner, Fürstenfeldbruck
Lektorat: Frank Eller, feller@pearson.de
Herstellung: Elisabeth Egger, eegger@pearson.de
Satz: reemers publishing services gmbh, Krefeld
Umschlaggestaltung: Marco Lindenbeck, mlindenbeck@webwo.de
Druck und Verarbeitung: Bercker, Kvelaer

Printed in Germany

Inhaltsverzeichnis

Vorwort	13
Teil I Einführung	15
Einführung	17
1 Zum Buch	17
2 Internet-Recherche	20
3 Grundlagen zu den im Buch verwendeten externen Technologien	29
Teil II Rezepte	81
Basics	83
1 (Ganz-)Zahlen hexadezimal darstellen	83
2 Hexadezimale Zahlen in dezimale Zahlen konvertieren	83
3 Ganzzahlen dual darstellen	84
4 Dualzahlen in Ganzzahlen konvertieren	88
5 Zahlen in verschiedenen Notationen darstellen	88
6 Zahlen kaufmännisch runden	89
7 Umstellen der aktuellen Kultur	91
8 Zahlen in verschiedenen Länderformaten darstellen	92
9 Einen Bytewert in ein besser lesbares Format umwandeln	93
10 Überprüfen, ob ein String eine gültige Zahl speichert	94
11 Überprüfen, ob ein String ein gültiges Datum speichert	97
12 Zufallszahlen erzeugen	98
13 High- und Low-Word lesen	101
14 High- und Low-Word setzen	102
15 Beep ausgeben	103
16 Strings ohne Berücksichtigung der Groß-/ Kleinschreibung vergleichen	104
17 Teilstrings flexibel ersetzen	105
18 Mehrere (Leer-)Zeichen in ein (Leer-)Zeichen konvertieren	110
19 Erstes Zeichen aller Wörter im String groß- und die restlichen Zeichen kleinschreiben	111
20 Anzahl der Wörter in einem String ermitteln	113
21 Alle Wörter eines Strings extrahieren	116
22 String auf die ersten Wörter bis zu einer Maximallänge kürzen	117
23 Teilstring nur dann ersetzen, wenn dieser am Anfang eines Strings gespeichert ist	119
24 Linken und rechten Teilstring extrahieren	120
25 (Ganz-)Zahlen aus einem String extrahieren	121
26 Überprüfen, ob ein String mit einer Ganzzahl beginnt	122

27	Zufalls-String berechnen	123
28	Konstantennamen einer Aufzählung als String auslesen	124
29	Strings in Aufzählungswerte konvertieren	125
30	String in Byte-Array umwandeln	126
31	Byte-Array in String lesen	128
32	Arrays eigener Objekte sortieren	129
33	StringCollection (und andere Auflistungen) sortieren	131
34	Arrays, ArrayList- und andere Auflistungen durchsuchen	133
35	Typisierte Auflistungen erzeugen	135
36	Das Programm für eine definierte Zeit anhalten	141
Datum und Zeit		143
37	Aktuelles Datum und aktuelle Uhrzeit ermitteln	143
38	Datumswerte formatieren	143
39	Datumswerte in verschiedenen Länderformaten ausgeben	145
40	ISO-Datum ermitteln	147
41	Datumswerte vergleichen	148
42	Die Differenz zwischen zwei Datumswerten in Tagen und deren Teilen berechnen	149
43	Die Differenz zwischen zwei Datumswerten in höheren Datumseinheiten berechnen	150
44	Ein Alter in Jahren berechnen	152
45	Datumswerte addieren und subtrahieren	153
46	Die Anzahl der Tage eines Monats berechnen	154
47	Die Anzahl der Tage eines Jahres berechnen	155
48	Ermitteln, ob ein Jahr ein Schaltjahr ist	156
49	Ermitteln der ISO-Kalenderwoche	156
50	Ermitteln, mit welchem Datum eine Kalenderwoche beginnt	160
51	Anzahl der Kalenderwochen eines Jahres bestimmen	163
52	Systemdatum und -zeit setzen	164
53	Atomuhr-Datum inklusive Uhrzeit von einem NIST-Server abfragen	167
54	Datum auf Gültigkeit überprüfen	172
55	Datum normalisieren	174
56	Deutsche Feiertage und andere besondere Tage berechnen	178
57	Ermitteln, ob ein bestimmter Tag ein Feiertag ist	188
58	Zeit genau messen	189
Anwendungs-Konfiguration, Anwendungen, Prozesse und Dienste		195
59	Den Dateinamen der Anwendung ermitteln	195
60	Das Verzeichnis der Anwendung ermitteln	196
61	Konfigurationsdaten aus der .config-Datei lesen	197
62	Konfigurationsdaten in eigenen Abschnitten speichern	198
63	Konfigurationsdaten in der .config-Datei speichern	202
64	Anwendungskonfiguration über eine eigene XML-Datei	204
65	Daten aus der Registry lesen und in die Registry schreiben	212
66	Verhindern, dass eine Anwendung mehrfach gestartet werden kann	218

67	Aktivieren einer laufenden Anwendung	223
68	Auflisten aller laufenden Prozesse eines Computers	227
69	Auflisten aller laufenden Anwendungen eines Computers	229
70	Informationen über die Dienste eines Computers auslesen	230
71	Dienste starten, anhalten, fortfahren und stoppen	233
72	Ermitteln, ob eine Anwendung ausgeführt wird	237
73	Anwendungen starten	239
74	Dateien mit der assoziierten Anwendung öffnen	239
75	Starten einer anderen Anwendung und warten, bis diese beendet ist	240
76	Andere Anwendungen beenden	242
77	Java-Anwendungen starten	243

Dateisystem	249

78	Dateiname einer Windows-Anwendung ermitteln	249
79	Dateiname einer Klassenbibliothek ermitteln	250
80	Ordner einer Windowsanwendung ermitteln	252
81	Ordner einer Klassenbibliothek ermitteln	252
82	Dateiname für eine temporäre Datei erzeugen	253
83	Ordner für temporäre Dateien ermitteln	253
84	Windows-(Spezial-)Ordner auslesen	253
85	Dateiname mit anderer Endung ermitteln	256
86	Die logischen Laufwerke des Systems ermitteln	256
87	Den Typ der Laufwerke eines Systems ermitteln	257
88	Alle Laufwerke eines bestimmten Typs ermitteln	263
89	Größe und freien Speicherplatz eines Laufwerks ermitteln	266
90	Überprüfen, ob ein Ordner existiert	268
91	Ordner erzeugen	269
92	Ordner mit .NET-Features kopieren	269
93	Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage über SHFileOperation kopieren	271
94	Ordner mit .NET-Features ohne Abbruch bei einer Ausnahme kopieren	277
95	Ordner umbenennen	285
96	Ordner verschieben	285
97	Ordner löschen	286
98	Ordnergröße ermitteln	287
99	Ordnergröße aller direkten Unterordner ermitteln	288
100	Überprüfen, ob eine Datei existiert	293
101	Programmdateien in den Systempfaden suchen	294
102	Dateien kopieren	295
103	Dateien umbenennen	297
104	Dateien verschieben	298
105	Dateien löschen	298
106	Dateien vergleichen	299
107	Ordner vergleichen	301
108	Die Größe einer Datei ermitteln	304

109	Dateiattribute auslesen	305
110	Die Version einer Datei auslesen	306
111	Den kurzen (8.3-)Dateinamen einer Datei auslesen	307
112	Alle Dateien eines Ordners auflisten	309
113	Dateien suchen	310
114	Dateien und Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage umbenennen oder verschieben	313
115	Dateien und Ordner in den Papierkorb verschieben	317
116	Größe des Papierkorbs und Anzahl der Dateien im Papierkorb ermitteln	319
117	Verknüpfungen anlegen	322
Text-, binäre und Zip-Dateien		325
118	Textdateien lesen	325
119	Textdateien schreiben	328
120	Texte an Textdateien anfügen	330
121	Textdateien ändern	331
122	Binäre Dateien lesen	332
123	Binäre Dateien schreiben	334
124	Binäre Dateien Base64-codieren	335
125	Base64-codierte Strings in Dateien umwandeln	336
126	(ZIP-)Archive erzeugen	337
127	(ZIP-)Archive aus einem Ordner erzeugen	343
128	(ZIP-)Archive aus beliebigen Daten erzeugen	345
129	Dateiinformationen aus (ZIP-)Archiven auslesen	346
130	(ZIP-)Archive entpacken	348
131	Den Inhalt von (ZIP-)Archiven lesen	352
XML		355
132	XML-Dateien über ein XmlTextReader-Objekt lesen	355
133	XML-Dateien beim Einlesen gegen ein Schema (oder DTD) prüfen	360
134	XML-Dokumente auf Gültigkeit überprüfen	365
135	XML-Dateien über das DOM lesen	371
136	XML-Dateien validierend über das DOM lesen	375
137	Gezielt über XPath auf Elemente in einem XML-Dokument zugreifen	377
138	XML-Dokumente über ein DataSet-Objekt lesen	385
139	XML-Dateien mit einem XmlTextWriter erzeugen	389
140	XML-Dateien über ein XmlDocument-Objekt erzeugen und ändern	392
141	XML-Dateien über ein DataSet-Objekt erzeugen	401
142	XML-Dateien über ein DataSet ändern	403
143	Binäre Daten in einer XML-Datei speichern	404
144	Base64-codierte Bilder (binäre Daten) aus XML-Dateien auslesen	406
145	XML-Dokumente über XSL-Dokumente transformieren	407

System	409
146 Den Namen des Computers ermitteln	409
147 Systempfade herausfinden	409
148 Windows-Version auslesen	411
149 Windows-Hauptversion ermitteln	412
150 Die Prozessorgeschwindigkeit ermitteln	413
151 Informationen über den Arbeitsspeicher auslesen	414
152 Die Version des aktuellen Service-Packs ermitteln	416
153 Installierte Programme auflisten	417
154 Informationen zu den parallelen Schnittstellen des Systems auslesen	419
155 Informationen zu den seriellen Schnittstellen des Systems auslesen	424
156 Ermitteln der Soundkarten des Systems	427
157 Windows herunterfahren und neu starten	429
158 Die verfügbaren Leistungsindikatoren auslesen	430
159 Den Arbeitsspeicher überwachen	437
160 Speicherauslastung und CPU-Belastung des aktuellen Prozesses überwachen	440
161 Speicherauslastung und CPU-Belastung des aktuellen Prozesses in ein Trace-Protokoll schreiben	442
162 Eigene Leistungsindikatoren implementieren	447
163 Monitor abschalten, in den Energiesparmodus schalten und wieder einschalten	452
Internet	455
164 Status der Internetverbindung abfragen	455
165 Pingen: Ermitteln, ob eine Internetverbindung besteht	458
166 TCP- und ICMP-Prüfsumme berechnen	461
167 Internetverbindung öffnen und schließen	463
168 Die IP-Adressen des Computers herausfinden	467
169 Mails über einen SMTP-Server versenden	469
170 E-Mails über MAPI/Outlook versenden	471
171 Browser starten	473
172 Webseiten (HTML-Dokumente) in der Anwendung darstellen	474
173 Dateien downloaden	481
Formulare und Steuerelemente	489
174 Formulare ohne Titelleiste	489
175 (Rahmenlose) Formulare über den Clientbereich verschiebbar machen	489
176 Unbewegbare Formulare mit Titelleiste	491
177 Andockende Formulare	492
178 Ändern des Systemmenüs: Entfernen des Schließen-Eintrags und -Schalters	493
179 Formulare verlaufend füllen	494
180 Formulare mit speziellen Formen	495
181 Splash-Formulare	500
182 Menüs in der Laufzeit erstellen oder erweitern	502
183 Steuerelemente mit transparentem Hintergrund	504
184 TextBox automatisch beim Eintritt selektieren	505
185 TextBox auf Zahleingaben beschränken	506

186	Auf Return in einer TextBox reagieren	510
187	ComboBox mit Autovervollständigung	510
188	Daten neben den Einträgen einer ListBox oder ComboBox verwalten	512
189	ListView sortieren	514
190	Knoten einer Ebene in einem TreeView-Steuerelement vertauschen	516
191	Einzelne Knoten eines TreeView-Steuerelements sortieren	518
192	Ein datengebundenes DataGrid anpassen	520
193	Ein DataGrid mit ComboBox-Spalten	525
194	Drag & Drop von Dateien und Ordnern	532
195	Ein Ordner-Dialog	533
196	Hooking: Eine MessageBox mit definierten Schalterbeschriftungen versehen	536
Benutzer, Gruppen und Sicherheit		547
197	Informationen zu den Benutzern eines Computers oder einer Domäne auflisten	547
198	Überprüfen, ob ein Benutzerkonto existiert	552
199	Benutzergruppen eines Computers oder einer Domäne auflisten	553
200	Benutzer ermitteln, die einer Gruppe angehören	556
201	Gruppen eines Benutzers abfragen	560
202	Eigenschaften eines Benutzers auslesen	562
203	Gruppe anlegen	568
204	Benutzer anlegen	570
205	Eigenschaften eines Benutzers ändern	572
206	Benutzer einer Gruppe zuweisen	573
207	Benutzer aus einer Gruppe entfernen	575
208	Benutzer löschen	576
209	Gruppe löschen	577
210	Den Namen des aktuellen Benutzers auslesen	577
211	Ein Programm unter einem spezifischen Benutzerkonto ausführen	578
212	Daten symmetrisch ver- und entschlüsseln	584
213	Daten mit Hashing-Verfahren verschlüsseln	597
Multimedia, Bilder und Grafiken		603
214	Wave-Dateien über das Windows-API abspielen	603
215	Multimedia-Dateien (Wave, MP3, Midi, AVI, MPEG etc.) über MCI abspielen	605
216	Multimedia-Dateien (Wave, MP3, Midi, AVI, MPEG etc.) über DirectX abspielen	617
217	Bild in Byte-Array umwandeln	620
218	Byte-Array in Bitmap umwandeln	621
219	Bitmap aus der Zwischenablage auslesen	621
220	Screenshot des Bildschirms und eines Formulars erstellen	622
221	Bilder skalieren	625
222	Thumbnails aus Bildern erzeugen	628
223	Bilder konvertieren	629
224	(JPEG-)Bilder mit definierter Qualität speichern	629
225	Bilder drehen und spiegeln	632
226	Bildausschnitte auslesen	636
227	Farbinformationen von Bildern gezielt verändern	637

228	Ein Negativ eines Bildes erzeugen	643
229	Die einzelnen Pixel eines Bildes bearbeiten	644
230	Farb-Bilder in Graustufen-Bilder umwandeln	645
231	Transparente Bilder und Grafiken erzeugen	646
232	Bilder mit Schatten zeichnen	648
233	Schräg zeichnen	649
234	Die Breite und Höhe eines auszugebenden Textes bestimmen	653
COM-Interop mit Office (Automation)		655
235	Laufende COM-Komponenten-Instanzen referenzieren	655
236	Word-Dokumente öffnen	657
237	Word-Dokumente basierend auf einer Dokumentenvorlage erzeugen, füllen, ausdrucken und speichern	660
238	Excel-Arbeitsmappen erzeugen	664
239	Daten in Excel-Arbeitsmappen erweitern	669
240	Kontakte aus Outlook auslesen	673
Sonstiges		677
241	Informationen über den Aufrufer in einer Methode oder einem Konstruktor ermitteln	677
242	Parameter an Threads übergeben und Ergebnisse auslesen	678
243	Objekte binär serialisieren	680
244	Objekte nach XML serialisieren und von XML deserialisieren	684
245	Font- und andere Objekte in einen String serialisieren	687
246	Die Anzahl der Datensätze ermitteln, die eine Abfrage in einer Datenbank ergibt	688
247	Datenbanken erzeugen	689
248	Abfragen der automatisch vergebenen Id eines neuen Datensatzes	692
249	Bilder und andere binäre Daten in einer Datenbank verwalten	693
Teil III Glossar		697
Referenz		699
1	Sprachreferenz	699
2	Referenz der wichtigsten Basisklassen	705
3	Referenz der wichtigsten der im Buch verwendeten Klassen	715
4	Referenz der im Buch verwendeten Windows-API-Funktionen	727
Stichwortverzeichnis		749

Vorwort

Das .NET Framework ist ein mächtiges Werkzeug mit sehr vielen Möglichkeiten. Und doch oder vielleicht gerade deswegen ist es manchmal nicht einfach herauszufinden, wie ein bestimmtes Problem gelöst werden kann. Hinzu kommt, dass bestimmte Bereiche wie z.B. Multimedia zurzeit noch nicht unterstützt werden oder über spezielle, externe Komponenten eingebunden werden müssen.

Eine Recherche in den zur Verfügung stehenden Quellen (Dokumentation, Quickstart Tutorials, Beispiele, Internet, Bücher) ist in vielen Fällen recht mühselig. Wie oft hatte ich mir während der Programmierung unter C# gewünscht, ein zentrales Nachschlagewerk für die wichtigsten Problemlösungen zu besitzen.

Jetzt habe ich eines ☺. Und Sie auch, falls Sie dieses Buch kaufen oder gekauft haben. In nahezu 250 Rezepten finden Sie hier Lösungen zu immer wiederkehrenden Problemen. Dabei habe ich darauf geachtet, diese möglichst praxisnah auszuwählen. Aufgrund der Vielfalt ist es natürlich nicht möglich, alle Fragen abzudecken. Sie werden aber einen Großteil der Probleme, die während der Programmierung auftreten, in diesem Buch gelöst finden.

Bei der Erstellung der einzelnen Rezepte habe ich wesentlich mehr Wert auf Qualität als auf Quantität gelegt. Besonders wichtig war mir, dass Sie das, was dort programmiert ist, auch verstehen. Deshalb enthält der entsprechende Quellcode jedes Rezepts Kommentare, die die Programmierung kurz erläutern. Dadurch und durch die Beschreibung der Rezepte sollten wohl keine Unklarheiten übrig bleiben. Außerdem finden Sie keine »Quick and Dirty«-Lösungen in diesem Buch. Ich habe jedes Rezept sehr intensiv getestet und potenzielle Fehlerquellen beseitigt (was manchmal für ein Rezept einige Stunden oder auch einen ganzen Tag in Anspruch genommen hat). Natürlich gibt es für einzelne Probleme mehrere Lösungswege. In sehr seltenen Fällen, für gleichwertige Lösungen, die unterschiedliche Features erbringen, habe ich die einzelnen Wege beschrieben. Meist habe ich aber den optimalen Weg gesucht und nur diesen erläutert.

Zudem sind die Rezepte größtenteils so gestaltet, dass Sie den Programmcode direkt in Ihr Projekt einfügen können. Das Repository, das Sie auf der Buch-CD finden, hilft enorm dabei, die Rezepte zu kopieren und weiterzuverwenden.

Bei der Auswahl der Themen habe ich mich vorwiegend an den Fragen orientiert, die in den einschlägigen C#-Newsgroups gestellt wurden. Sehr hilfreich waren dabei die Foren *microsoft.public.de.german.entwickler.dotnet.csharp*, *microsoft.public.de.german.entwickler.dotnet.framework*, *microsoft.public.dotnet.languages.csharp* und *microsoft.public.dotnet.framework*.

Andere Quellen für Ideen waren zahlreiche .NET-Internetseiten, die ich in der Einführung nenne.

Ich danke meinem Lektor Frank Eller (der selbst .NET-Programmierer und -Autor ist) für die Unterstützung bei der Erstellung dieses Buchs. Thomas Herold danke ich für das Gegen-

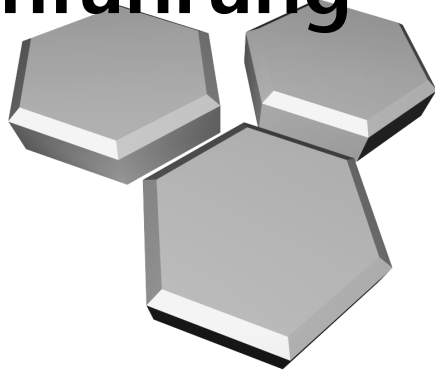
lesen einiger wichtiger Kapitel, für seine Anregungen und dafür, dass er mir seine Gitarre verkauft hat (die mir in den Schreib-Pausen enorm dabei half, Kräfte zu sammeln und meine Hände zu lockern). Holger Kohnen sage ich ebenfalls Dank, ohne seine mathematischen Künste wäre die Berechnung des Umfassungsrechtecks beim Drehen von Bildern wesentlich komplexer ausgefallen. Ein herzlicher Dank gilt auch meiner Lieblings-Korrektorin Simone Meißner. Die Zusammenarbeit war einmal wieder sehr anregend und produktiv.

Falls Sie Anregungen oder Kritik oder einfach nur Lob zum Buch haben, freue ich mich über eine E-Mail.

Jürgen Bayer
juergen.bayer@addison-wesley.de
Kamp-Lintfort im Juni 2003

TEIL I

Einführung



Einführung

1 Zum Buch

Der Inhalt

Das Buch beschreibt in nahezu 250 Rezepten (wenn die Einführung mitzählt sogar in mehr als 250) Lösungen zu Aufgabenstellungen, die in der Praxis häufig auftreten, aber meist nicht zu den trivialen Problemen gehören, die über eine einfache Verwendung einer .NET-Klasse gelöst werden können. Ausnahmen gibt es natürlich auch: Manchmal ist eine Lösung eigentlich trivial, aber bei vielen Programmierern unbekannt. Diese meist in Newsgroups gefragten Themen habe ich auch mit in den Rezeptteil aufgenommen.

Die meisten Rezepte dieses Buchs sind so allgemein, dass sie in jeder Anwendung einsetzbar sind. Der Fokus liegt aber auf der Programmierung von Windows-Anwendungen (über `Windows.Forms`). Techniken, die sich nur auf Webanwendungen beziehen, werden in diesem Codebook nicht beschrieben. Dazu gibt es ja schließlich das ASP.NET Codebook ☺. Damit Sie jetzt aber nichts Falsches denken: Nur das Kapitel »Formulare und Steuerelemente« bezieht sich direkt auf die Programmierung von `Windows.Forms`-Anwendungen.

Die Einführung

Die Einführung, die Sie gerade lesen, ist recht umfangreich. Das liegt daran, dass ich neben einer allgemeinen Einführung und wichtigen Hinweisen zur Recherche im Internet auch den (teilweise komplexen) Umgang mit den externen Technologien *Windows API*, *COM*, *WMI* und *ADSI* beschreibe, die alle im Buch eingesetzt werden. Ich habe diese Beschreibung mit in die Einführung aufgenommen, weil ich nicht davon ausgehen kann, dass Sie als C#-Programmierer damit sicher umgehen können.

Die Rezepte

Dieses Buch besteht neben der Einführung und dem Referenzteil ausschließlich aus Rezepten. Jedes Rezept beschreibt die komplette Lösung eines Problems. Nur selten, wenn es sinnvoll erschien, habe ich ein Rezept mit einem anderen verknüpft.

Die meisten Rezepte setzen Klassen des .NET Frameworks ein. Bei diesen gehe ich davon aus, dass Sie mit den Grundlagen des .NET Frameworks und natürlich mit denen von C# vertraut sind. Im jeweiligen Rezept werden aber die verwendeten, besonderen Techniken, die nicht zu den Grundlagen gehören, erläutert und die benötigten Assembly-Referenzen und Namensraum-Importe aufgelistet. In den Rezepten habe ich aus Platz- und Übersichtsgründen darauf verzichtet, die Klassen voll (inklusive Namensraum) zu referenzieren. Deshalb gebe ich vor dem eigentlichen Quellcode an, welche Namensräume importiert werden müssen.

Einige Rezepte arbeiten mit externen Technologien wie dem Windows API, WMI, COM oder ADSI. Grundlagen zu diesen Technologien, die ich nicht voraussetzen kann, finden Sie in dieser Einführung. In den Rezepten setze ich dann voraus, dass Sie diese Grundlagen kennen.

Nur wenige Rezepte nutzen fertige Klassen oder Bibliotheken, die Sie frei aus dem Internet beziehen können (und meist auch auf der Buch-CD finden). Die in diesen Klassen implementierten Programme waren einfach zu komplex für das Buch. Und außerdem lag es mir fern, die teilweise genialen Klassen einfach zu kopieren. Ich wollte Ihnen aber eine Lösung für das jeweilige Problem anbieten. Bei den Rezepten, die mit externen Klassen arbeiten, habe ich natürlich darauf verzichtet, die Programmierung der eingesetzten Klassen zu erläutern, und zeige nur deren Anwendung.

Ein Rezept beschreibt kurz das Problem und den Lösungsansatz bzw. den Lösungsweg, meist ohne zu erläutern, in welchen Fällen Sie das Programm überhaupt benötigen. Sie werden schon wissen, warum Sie ein Rezept lesen ☺. Die meisten Rezepte liefern die Problemlösung in Form einer Methode. Diese Methoden sind `static` deklariert, damit Sie diese einfach in eine separate Klasse einfügen und ohne eine Instanz zu bilden anwenden können. Wenn Sie die Methoden in eine Klasse einbinden, die instanziiert wird, entfernen Sie einfach das Schlüsselwort `static`.

Die Problemlösung einiger Rezepte erforderte aber auch die Programmierung kompletter Klassen. Diese sind teilweise recht umfangreich, das Ergebnis kann sich aber sehen lassen. Ein gutes Beispiel ist die Klasse `Mci` im Rezept 215, die Ihnen ermöglicht, Multimedia-Dateien abzuspielen (wobei im Hintergrund das etwas schwierig anzuwendende MCI verwendet wird).

Für wenige Rezepte war es prinzipiell nicht möglich, Methoden oder Klassen zu entwickeln. Dabei handelt es sich meist um Beschreibungen von Vorgehensweisen wie z.B. die Erstellung eines Formulars mit einer speziellen Formgebung (Rezept 180). Die Quellcodes dieser Rezepte müssen Sie natürlich an Ihre Bedürfnisse bzw. Anforderungen anpassen.

Neben der Beschreibung der Problemlösung im Fließtext habe ich in den Listings immer auch Kommentare untergebracht. Teilweise führt das zu einer leichten Redundanz, da Dinge zweimal erläutert werden. Ich denke aber, dass ein Quelltext mit Kommentaren wesentlich besser zu verstehen ist als einer ohne. So haben Sie auch die Möglichkeit, den Fließtext schlicht zu ignorieren (was eigentlich nicht in meinem Sinne ist, denn wozu habe ich das Ganze dann geschrieben J), und können sich einfach direkt den Quelltext anschauen. Verstehen Sie diesen nicht, können Sie im Fließtext nachlesen. Zu diesem Punkt würde mich übrigens ein Feedback über eine E-Mail sehr freuen.

Entwickelt habe ich die Rezepte unter Windows XP Professional mit den .NET Framework 1.0. Getestet habe ich alle Rezepte unter Windows XP mit dem .NET Framework 1.0 und 1.1, aber auch unter Windows 2000 und Windows 98 SE. Nur sehr wenige Rezepte liefen unter Windows 98 nicht oder mit Einschränkungen. Größtenteils lag das daran, dass bestimmte Dinge wie z.B. Dienste nicht unter Windows 98 verfügbar sind. Unter Windows 2000 waren bis auf eine kleine Ausnahme (im Rezept 66) keine Einschränkungen erkennbar.

Falls ein Rezept nur unter bestimmten Bedingungen funktioniert oder besondere Installationen erfordert, ist dies in der Regel im Rezept beschrieben. Bei den Rezepten, die WMI oder ADSI einsetzen, habe ich allerdings auch häufig darauf verzichtet, immer wieder zu schreiben, dass WMI bzw. ADSI installiert sein müssen. Grundlagen dazu finden Sie ja auch hier in der Einführung.

Das Repository, die Beispiele und Komponenten

Auf der Buch-CD finden Sie ein Repository in HTML-Form, das die Programmcodes aller Rezepte enthält. Über dieses können Sie eine Problemlösung sehr schnell in Ihr Projekt einfügen. Die Rezepte im Repository korrespondieren natürlich mit denen im Buch.

Zudem finden Sie auf der CD auch Beispiele zu allen Rezepten und die in einigen Rezepten verwendeten Komponenten. Die Beispiele liegen jeweils als Projekt für Visual Studio .NET 2002 und 2003 vor, sodass Sie beide Versionen dieser Entwicklungsumgebung verwenden können.

Das Erratum

Obwohl ich die einzelnen Rezepte sehr ausführlich unter Windows 98, 2000 und XP getestet habe, kann es in einem so umfangreichen Buch immer auch vorkommen, dass sich Fehler eingeschlichen haben. Einige Rezepte lassen sich eventuell auch verbessern oder funktionieren unter Umständen, unter denen ich diese nicht getestet habe, nicht oder nur eingeschränkt. An der Adresse www.juergen-bayer.net/buecher/csharpcodebook/index.html finden Sie deshalb neben allgemeinen Informationen zum Buch ein Erratum. In diesem werde ich eventuelle Fehler dokumentieren, aber auch Verbesserungen an einzelnen Rezepten veröffentlichen. Zum Füllen des Erratums bin ich auch auf Ihre Mitarbeit angewiesen. Es wäre also sehr nett, wenn Sie mir gefundene Fehler, Verbesserungen an Rezepten oder Anregungen zumailen. Meine Mailadresse finden Sie im Vorwort.

Die Referenz

Am Ende des Buchs finden Sie eine Referenz mit der Beschreibung der wichtigsten .NET-Klassen und von im Buch verwendeten Teilen externer Technologien. Um das Buch nicht zu sprengen, verzichtet die Referenz auf Vollständigkeit und liefert nur Informationen zu den wichtigen bzw. im Buch verwendeten Techniken.

Der Index

Jedes Fachbuch besitzt einen Index. Warum also darüber schreiben? Der Grund dafür ist, dass Sie im Index teilweise Problemlösungen finden, die nicht im Inhaltsverzeichnis auftauchen. Das liegt daran, dass ich innerhalb einiger Rezepte so ganz nebenbei auch andere Probleme lösen musste, für die ich keine eigenen Rezepte vorgesehen hatte. Damit Sie auch diese finden, habe ich die Lösungen immer mit in den Index aufgenommen.

Voraussetzungen

Das Buch setzt voraus, dass Sie die Grundlagen von C# beherrschen. Neben den Grundlagen der Sprache (Variablen, Schleifen, Verzweigungen etc.) sollten Sie die Idee der .NET-Klassenbibliothek kennen, mit Namensräumen umgehen können, wissen, was eine Klasse ist, und die wichtigen grundlegenden Klassen wie `String`, `Convert` oder `DateTime` kennen. Natürlich sollten Sie in der Lage sein, Ihre Programme zu kompilieren. Falls Sie keine Entwicklungsumgebung wie Visual Studio .NET oder SharpDevelop (www.icsharpcode.net/OpenSource/SD/Default.aspx) einsetzen, sollten Sie wissen, wie Sie beim Kompilieren Assemblies referenzieren. Ich empfehle aber dringend, wenigstens die hervorragende freie Entwicklungsumgebung SharpDevelop einzusetzen.

Typographische Konventionen

Das Buch verwendet einige typographische Konventionen, die hier kurz erläutert werden.

- ▶ Im Fließtext werden Wörter, die zu einem Programm gehören, in der Schriftart `Courier` ausgegeben,
- ▶ *kursiv formatierte* Wörter bezeichnen Dateinamen, Ordnernamen und allgemeine Namen,
- ▶ Internetadressen werden folgendermaßen formatiert: *www.addison-wesley.de*,
- ▶ WÖRTER IN KAPITÄLCHEN stehen für Begriffe, die Teil einer Programmoberfläche sind (wie z. B. die Beschriftung eines Schalters oder ein Link auf einer Internetseite),
- ▶ In den Listings werden die C#-Schlüsselwörter fett gedruckt.

2 Internet-Recherche

Die Programmierung mit derart komplexen Programmierungsumgebungen wie dem .NET Framework und den zusätzlich eingesetzten Technologien wie dem Windows-API, WMI und ADSI ist ohne die Möglichkeit einer Recherche im Internet kaum möglich. Im Internet finden Sie gerade für .NET eine Vielzahl an Informationsquellen, die Sie nutzen können, um Ihre Probleme effizienter zu lösen.

Das vorliegende Buch hilft Ihnen (hoffentlich ☺) bei der Lösung der meisten Aufgaben. Da Sie aber wahrscheinlich auch einmal spezielle Probleme oder solche lösen müssen, die im Buch nicht beschrieben werden, zeige ich Ihnen die Möglichkeiten der Internet-Recherche, die ich im Laufe meiner Tätigkeit als Softwareentwickler (im .NET-Bereich) gefunden habe.

Wichtige .NET-Websites

Viele Websites im Internet beschäftigen sich mit .NET. Ich kenne natürlich nicht alle, während meiner Arbeit mit .NET sind mir aber einige sehr gute Websites über den Weg gelaufen, die ich hier kurz vorstelle. Auf den meisten dieser Websites finden Sie neben Beispiel-Quellcodes auch kompakte Artikel, die spezifische und teilweise komplexe Themen (meist) verständlich erläutern.

Tipp

Die Links zu den Websites, die ich hier beschreibe, finden Sie auch in der Datei *Links.html* auf der Buch-CD.

CodeHound

Über die C#-Suchmaschine von *CodeHound* (www.codehound.com/csharp) können Sie auf allen wichtigen C#-Webseiten suchen und erhalten so eine sehr umfangreiche, aber eben auf C# spezialisierte Übersicht zu gesuchten Themen. In einigen Fällen findet CodeHound leider auch Artikel, die für andere Sprachen (wie z. B. C++) geschrieben wurden.

Windows Forms

Das zu Microsoft gehörende *Windows Forms Team* stellt auf seiner Website (www.windows-forms.net) eine Vielzahl an Artikeln, Beispielen, Foren etc. zur Verfügung. Viele Artikel stammen aus dem MSDN, sind aber hier übersichtlich zusammengefasst.

Got Dot Net

Die ebenfalls zu Microsoft gehörende Website des *Got Dot Net Teams* (www.gotdotnet.com) liefert eine Menge allgemeine und spezifische Informationen zu .NET.

dotnetjunkies

Bei den *dotnetjunkies* (www.dotnetjunkies.com) finden Sie viele, gute und hervorragend formatierte How-To-Artikel, Tutorial-Artikel und Beispiel-Codes, hauptsächlich zu ASP.NET und dessen Umfeld.

Planet Source Code / .NET

Unter der (etwas irreführenden) Adresse www.planetsourcecode.com/vb/default.asp finden Sie die .NET-Sektion von *Planet Source Code*. In Kategorien aufgeteilt können Sie Beispielprogramme für verschiedene Problemlösungen im Browser ansehen und/oder herunterladen.

The Code Project

Das *Code Project* (www.codeproject.com) veröffentlicht auf seiner Website im Vergleich zu anderen Websites zwar nicht allzu viele Artikel, die dafür aber qualitativ hochwertig sind und auch spezielle Themen behandeln. Auf der Seite www.codeproject.com/dotnet finden Sie allgemeine .NET-Artikel. Die Seite www.codeproject.com/csharp verweist auf Artikel zu C#. Andere Seiten, die Sie über das Menü der Einstiegsseite erreichen, beschäftigen sich (u.a.) mit speziellen .NET- und C#-Themen. Daneben finden Sie aber auch Seiten zu wichtigen Windows-Technologien wie z.B. DirectX und dem Windows-API.

.NET 247

.NET 247 (www.dotnet247.com) versteht sich als »erste, unabhängige .NET-Programmierer-Referenz« und verweist in gut geordneten Kategorien auf Artikel anderer Websites zu wichtigen .NET-Referenz- und -Programmier-Themen. Neben diesen finden Sie auf der Startseite Links zu den wichtigsten Namensräumen des .NET Frameworks und auf der jeweiligen Seite die im Namensraum enthaltenen Klassen. Die Seite einer .NET-Klasse enthält wiederum Links zu Ressourcen, Artikeln, Beispielen, Diskussionen etc., die mit der Klasse in Zusammenhang stehen. Das Ganze ist sehr hilfreich, wenn Sie nicht wissen, wie eine Klasse in der Praxis angewendet wird.

.net Magazine

Auf der Seite www.fawcette.com/dotnetmag finden Sie hochwertige Artikel des für US-Bürger frei verfügbaren *.net Magazine*. Im Archiv (www.fawcette.com/archives/magazines/dotnetmag) können Sie die älteren Ausgaben durchforsten.

Master C#

Auf der übersichtlich aufgebauten Seite von *Master C#* (www.mastercsharp.com) finden Sie einige recht gute Artikel zu verschiedenen .NET-Themen.

C# Corner

Die »C#-Ecke« (www.c-sharpcorner.com) enthält sehr viele Artikel, Quellcodes und Links zu C#-Themen. Die einfache Navigation und die Unterteilung der Inhalte in gut sortierte Kategorien machen diese Website für .NET-Programmierer sehr wertvoll. Die Artikel sind meist kurz gefasst, enthalten aber alle wesentlichen Informationen (was nicht bei allen .NET-Seiten der Fall ist).

C# Help

Auf der Website von *C# Help* (www.csharp-help.com) finden Sie viele Artikel zu C# (zurzeit über 490), die sich teilweise mit Grundlagen, aber auch mit fortgeschrittenen Themen beschäftigen. Leider existierte zur Zeit der Drucklegung dieses Buchs kein Verzeichnis aller Artikel (nur die aktuellen waren auf der Startseite verlinkt). Sie können aber natürlich nach archivierten Artikeln suchen.

.netWire

.netWire (www.dotnetwire.com) ist ein Website-übergreifender Newsletter für .NET. Auf der Startseite finden Sie neue Artikel verschiedener .NET-Websites, im Archiv können Sie nach älteren Artikeln suchen. Wenn Sie den .netWire-Newsletter abonnieren, erhalten Sie regelmäßige E-Mails mit Links zu den neuesten Artikeln.

www.aspheute.com

aspheute (www.aspheute.com) ist eine hervorragende deutschsprachige Website mit sehr vielen Artikeln zu ASP, ASP.NET und benachbarten Themen.

devX

Der .NET-Bereich der umfangreichen *devX*-Website (www.devx.com/dotnet) enthält viele Artikel zu .NET, die teilweise frei verfügbar sind, teilweise aber (leider) zum kostenpflichtigen »Premier Content« gehören. Die vielen und guten Artikel von devX sind leider nicht allzu einfach zu finden (klicken Sie auf den Link MORE C# ARTICLES, um eine Liste anzuzeigen). Sie können aber natürlich (leider mit einer eingeschränkten Google-Suche) auch nach Artikeln suchen. Der E-Mail-Newsletter, den Sie über die .NET-Startseite abonnieren können, ist hingegen hervorragend. Er enthält eine Kurzbeschreibung der neuesten Artikel und ermöglicht deren direkten Aufruf über den E-Mail-Client.

ONDotnet.com

Auf der .NET Website des Verlags O'Reilly (www.ondotnet.com) finden Sie eine Vielzahl an interessanten Artikeln, die teilweise Grundlagen-, aber auch spezifische Themen behandeln.

Lutz Roeder's PROGRAMMING.NET

Lutz Roeder veröffentlicht im .NET-Bereich seiner Website (www.aisto.com/roeder/dotnet) einige sehr interessante .NET-Komponenten wie z.B. einen Reflector, über den Sie .NET-Komponenten und Assemblierungen erforschen können, und eine CommandBar-Komponente, über die Sie verschiedene Befehls-Symbolleisten in Ihre Windows-Forms-Anwendung integrieren können.

George Shepherd's Windows Forms FAQ

Auf der Seite von *SyncFusion*, einem Hersteller von .NET-Komponenten, veröffentlicht George Shepherd ein umfangreiches FAQ, das sich nicht nur mit den Windows-Forms-Komponenten, sondern auch mit Themen wie GDI+, dem Windows-API, COM und allgemeinen Framework-Tipps beschäftigt. Die im FAQ enthaltenen Tipps und Tricks wurden dabei verschiedenen Newsgroups und Mailing-Listen entnommen bzw. stammen von Mitarbeitern von SyncFusion. Sie finden dieses FAQ an der Adresse www.syncfusion.com/faq/winforms.

Newsgroup-Recherche bei Google

Die bekannte Suchmaschine Google ermöglicht nicht nur die Suche nach Websites. Google archiviert daneben auch alle wichtigen Newsgroups und ermöglicht die Suche in diesem Archiv. Gerade .NET-Themen werden in einer Vielzahl spezialisierter Newsgroups umfangreich diskutiert. Hier finden Sie in den meisten Fällen Lösungen zu Ihren Problemen oder wenigstens unbeantwortete Fragen, an denen Sie erkennen, dass auch andere Programmierer ähnliche Probleme haben. Sie können natürlich einzelne Newsgroups in einem Newsgroup-Reader (wie Outlook oder dem Newsgroup-Reader des Mozilla-Browsers) abonnieren und die einzelnen Beiträge durchgehen. Da aber gerade die englischsprachigen Newsgroups mit Beiträgen geradezu »bombardiert« werden, ist diese Art der Suche sehr ineffizient. Die Google-Suche bringt eigentlich immer wesentlich schneller ein Ergebnis. Ehrlich gesagt hätte ich dieses Buch kaum ohne diese hervorragende Möglichkeit, in den aktuellen und älteren Newsgroup-Beiträgen zu suchen, schreiben können ☺.

Sie können die (erweiterte) Google-Newsgroup-Suche direkt über die Adresse www.google.de/advanced_group_search öffnen. Auf der Suchseite können Sie nach verschiedenen Kriterien suchen. Für die Praxis wichtig sind aber lediglich die Eingabe der Suchkriterien und die Einschränkung der Newsgroups. Abbildung 1 zeigt eine Suche nach einem »folder dialog« in allen Newsgroups, die »dotnet« im Namen tragen.

Wenn Sie nach englischen Begriffen suchen, finden Sie wesentlich mehr (englischsprachige) Beiträge, als wenn Sie deutsche Begriffe einsetzen. Die deutschen Newsgroups werden meist wesentlich weniger frequentiert als die englischen.

Die Einschränkung auf passenden Newsgroups im Feld NEWSGROUP ist sehr wichtig, da Sie ansonsten auch Beiträge aus Newsgroups finden, die nichts mit .NET oder C# zu tun haben. Das Einschränken ist auch sehr einfach, da (nach meiner Erfahrung) alle deutsch- oder englischsprachigen .NET-Newsgroups den Text »dotnet« im Namen tragen. Die Angabe von **dotnet** im Feld NEWSGROUP teilt Google dann auch mit, in allen Newsgroups zu suchen, deren Name »dotnet« enthält. Eine spezielle Einschränkung auf C# (über **csharp**) ist meiner Ansicht nach nicht notwendig, da Sie ohne Probleme auch Lösungen für Visual Basic .NET und mit ein wenig Gehirnschmalz auch Lösungen für Visual C++ .NET nach C# umsetzen können. Außerdem behandeln einige Newsgroups relativ sprachunabhängig allgemeine .NET-Themen und tragen folglich kein »csharp« im Namen. Es schadet allerdings auch nicht, einfach beide Newsgroup-Muster anzugeben: **dotnet**; **csharp**.

Google listet nach der Suche die gefundenen Beiträge sortiert nach Relevanz auf (Abbildung 2).

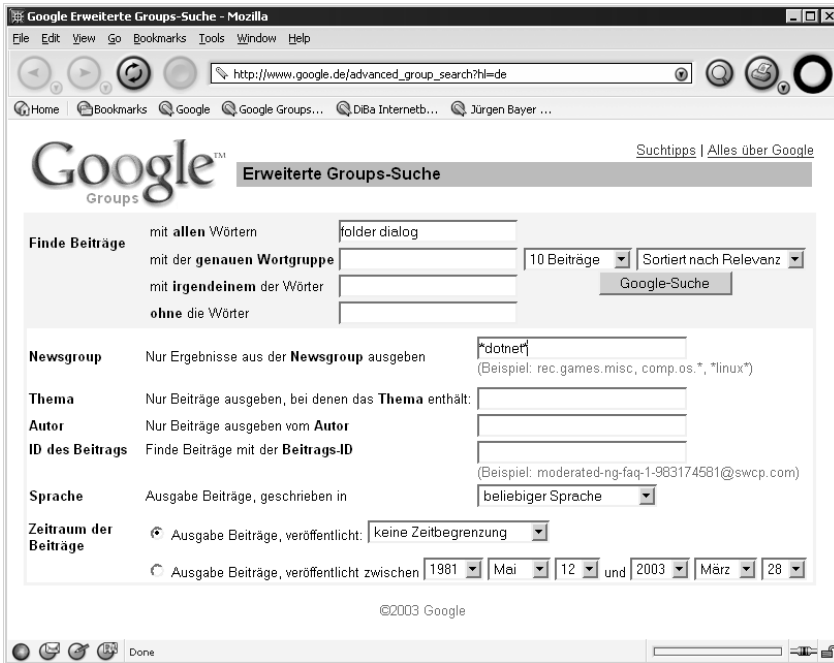


Abbildung 1: Google-Newsgroup-Recherche in allen .NET-Newsgroups

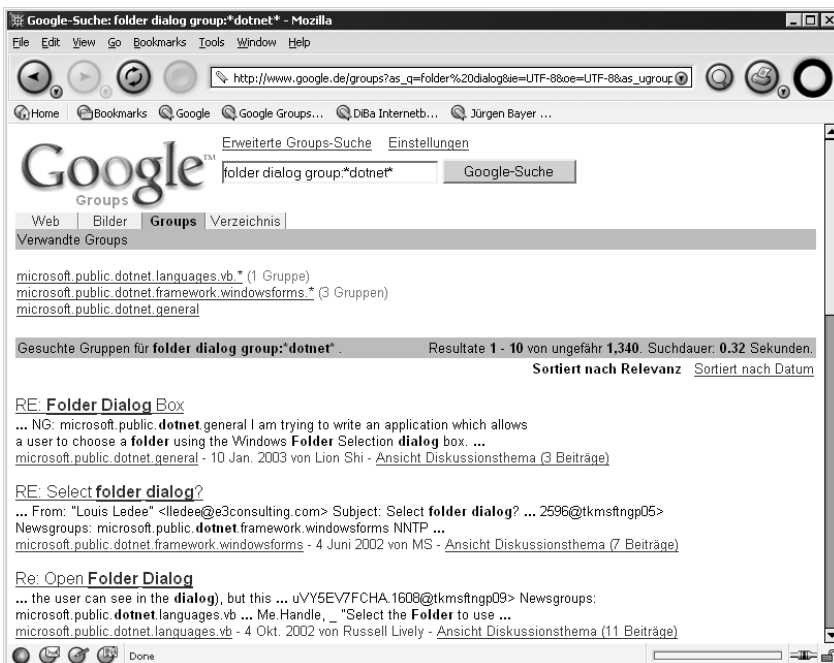


Abbildung 2: Das Ergebnis einer Google-Newsgroup-Recherche nach den Suchbegriffen »folder dialog« in den .NET-Newsgroups

Die Links führen zu den jeweiligen Beiträgen, die teilweise ursprüngliche Beiträge und oft Antworten auf vorhergehende Beiträge sind (was Sie am »RE:« erkennen).

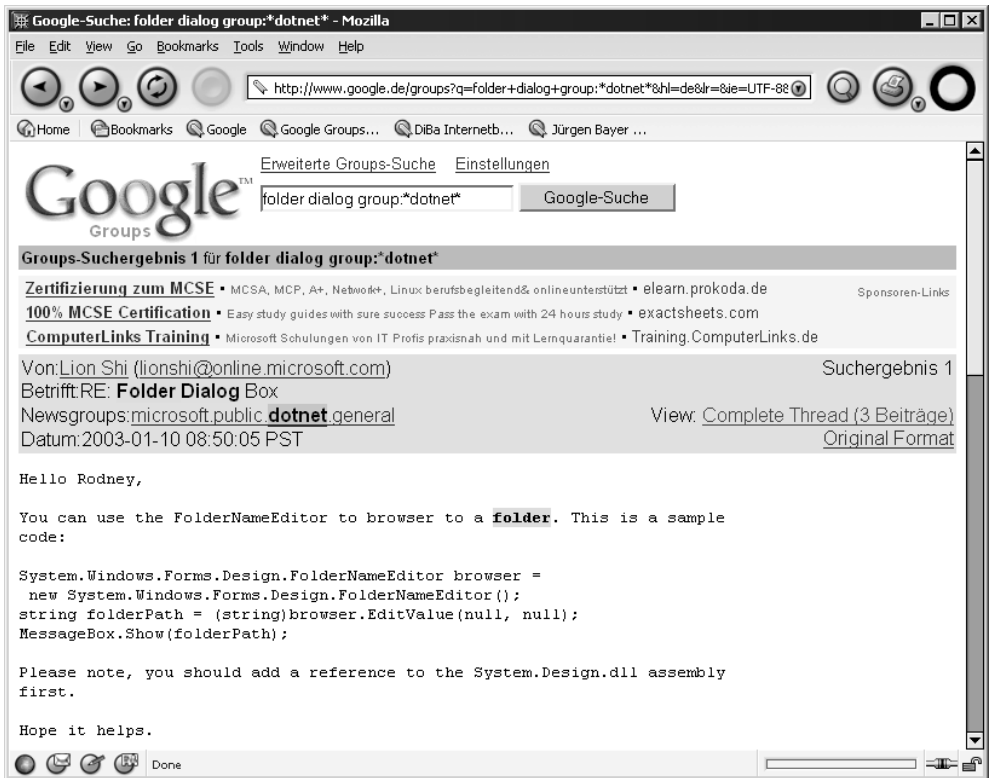


Abbildung 3: Ein gefundener Newsgroup-Artikel

In diesem Beispiel (das Sie aufgrund der permanent neuen Beiträge in den Newsgroups wahrscheinlich so nicht nachvollziehen können) führt die Suche direkt zum Erfolg. Das ist zwar nicht immer so, manchmal müssen Sie einige Beiträge durchforsten oder auch einmal die Suchbegriffe ändern um eine Lösung zu finden. Wenn Sie diese allerdings gefunden haben, erspart dies meist eine Menge Such- und Probierarbeit.

Falls der gefundene Beitrag Ihre Frage nicht direkt beantwortet oder Sie weitere Informationen benötigen, ist es oft sinnvoll, sich den kompletten Nachrichten-Thread anzusehen. Der Nachrichten-Thread geht vom ursprünglichen Beitrag aus und listet alle direkten und indirekten Antworten auf diesen Beitrag auf. Klicken Sie zur Ansicht des Nachrichten-Threads auf den Link COMPLETE THREAD (N BEITRÄGE) (der nur verfügbar ist, wenn es sich nicht um eine ursprüngliche Nachricht handelt, auf die niemand geantwortet hat). In der erscheinenden Navigations-Spalte können Sie alle mit dem gefundenen Beitrag assoziierten Beiträge öffnen (Abbildung 4).

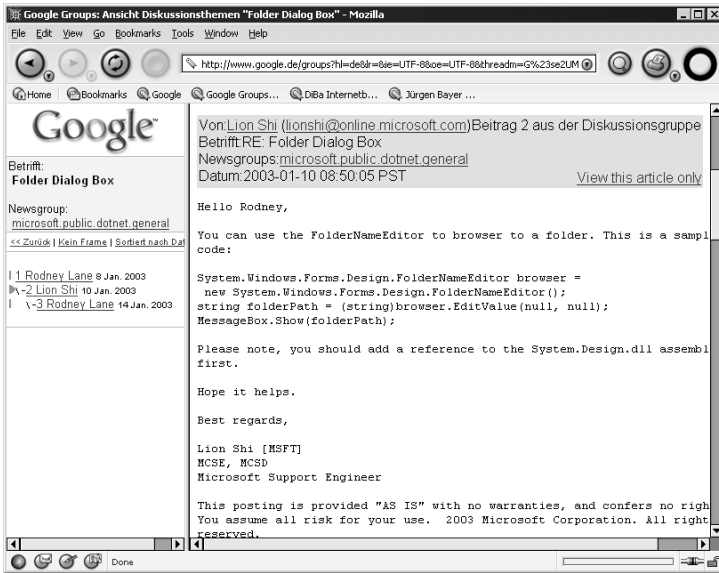


Abbildung 4: Der Nachrichten-Thread eines gefundenen Newsgroup-Artikels

Suche bei Microsoft

Auf der Seite *support.microsoft.com* können Sie sehr einfach in der recht umfangreichen Knowledge Base (der »Wissensdatenbank«) von Microsoft suchen (Abbildung 5).

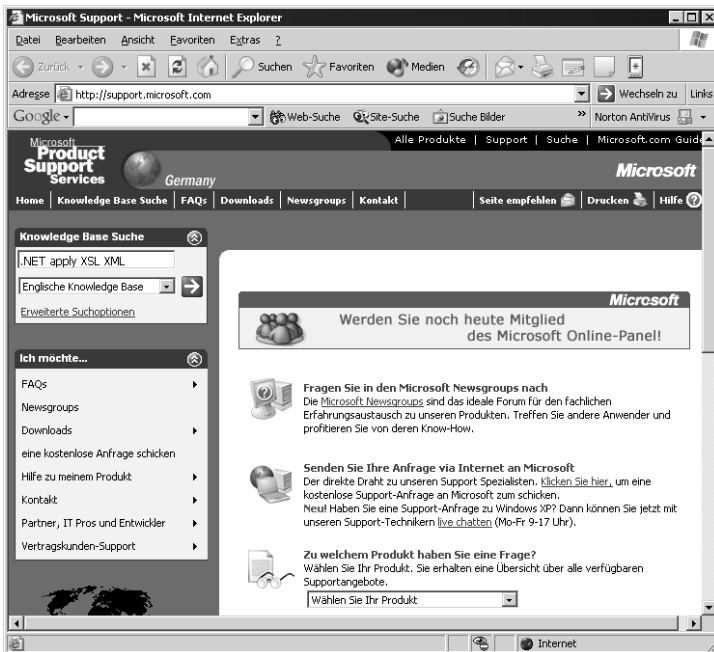


Abbildung 5: Microsoft-Support-Suche mit dem Internet Explorer

Interessant ist, dass die Support-Seite von Microsoft in einer anderen Variante erscheint, wenn Sie diese mit einem englischen Mozilla-Browser (oder einem beliebigen englischen Browser?) öffnen. Neben der geänderten Ansicht besitzen Sie dann nicht die Möglichkeit, die deutsche Knowledge Base auszuwählen. Scheinbar liest die Webanwendung die Spracheinstellung des Browsers aus und zeigt je nach Land eine spezifische Seite an.

Hinweis

Die Support-Seiten von Microsoft wurden in der Vergangenheit sehr häufig geändert, was Microsoft wahrscheinlich auch in der Zukunft so fortführen wird. Deshalb werden diese Seiten mit ziemlicher Wahrscheinlichkeit anders aussehen, wenn Sie die Microsoft-Support-Adresse im Browser öffnen.

Die Microsoft Knowledge Base enthält sehr viele How-To-Artikel, Problembereiche und Bug-Artikel zu allen Microsoft-Produkten. Schon allein deshalb empfiehlt es sich, bei der Suche einen Schlüsselbegriff für die Umgebung einzugeben (im Beispiel .NET). Etwas eigenartig ist, dass zurzeit (die Microsoft-Suchseite ändert sich sehr häufig ...) erst nach einer einmaligen Suche die Möglichkeit besteht, diese auf ein Produkt einzuzugrenzen (Abbildung 6).

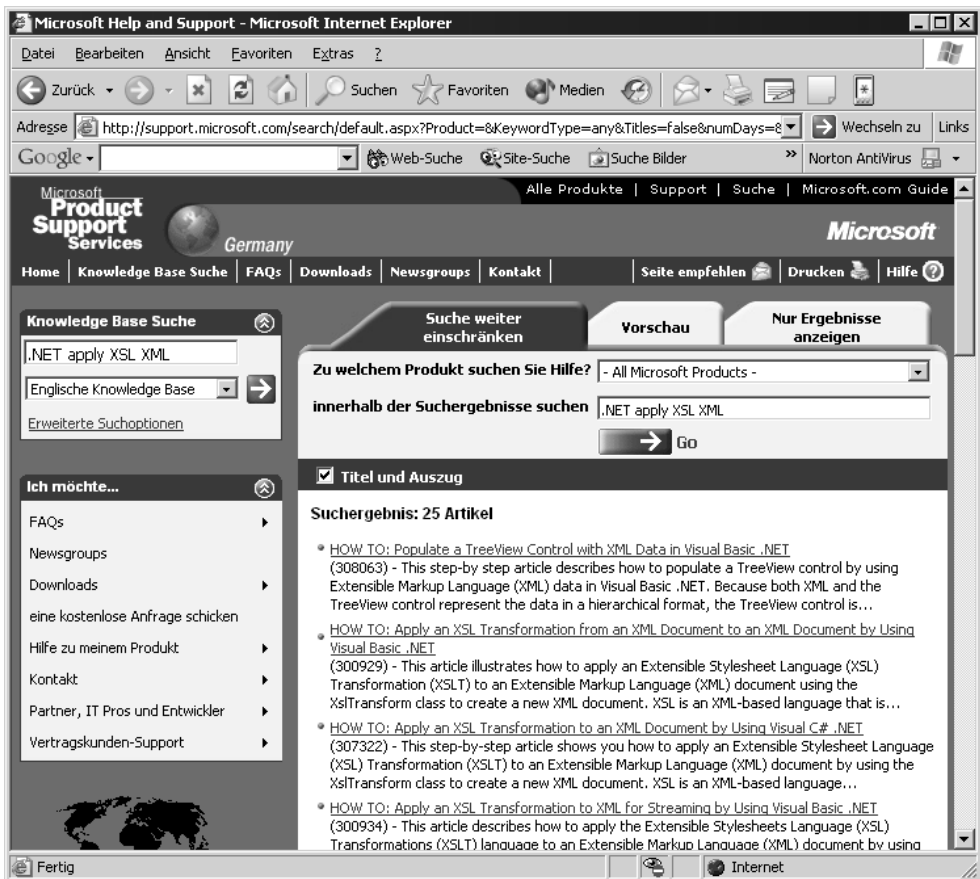


Abbildung 6: Suchergebnis mit der Möglichkeit, die Suche auf ein Produkt einzuzugrenzen

Knowledge Base-Artikel besitzen immer eine Nummer, die häufig in Newsgroup-Beiträgen angegeben wird. Sie können diese Nummer einfach im Suchfeld angeben um den entsprechenden Artikel zu suchen. Einfacher ist es aber, wenn Sie im Browser eine Adresse in der folgenden Form eingeben:

support.microsoft.com/?kbid=KB-Artikel-Nummer

Eine andere Suchmöglichkeit ist die MSDN-Suche. Im *Microsoft Developer Network* veröffentlichten Microsoft und externe Autoren sehr viele Informationen und Dokumentationen für Programmierer zu allen Microsoft-Produkten. Sie finden das spezielle deutsche MSDN an der Adresse *www.microsoft.com/germany/msdn*. Das englische MSDN finden Sie an der Adresse *msdn.microsoft.com*. In der MSDN-Bibliothek können Sie auch suchen, was in einer einfachen Form schon auf der Startseite möglich ist. Die erweiterte deutsche Suchseite finden Sie an der Adresse *www.microsoft.com/germany/ms/msdnbiblio/suche.asp*, die englische erweiterte Suche an der Adresse *search.microsoft.com/advanced_search.asp?siteid=us/dev*.

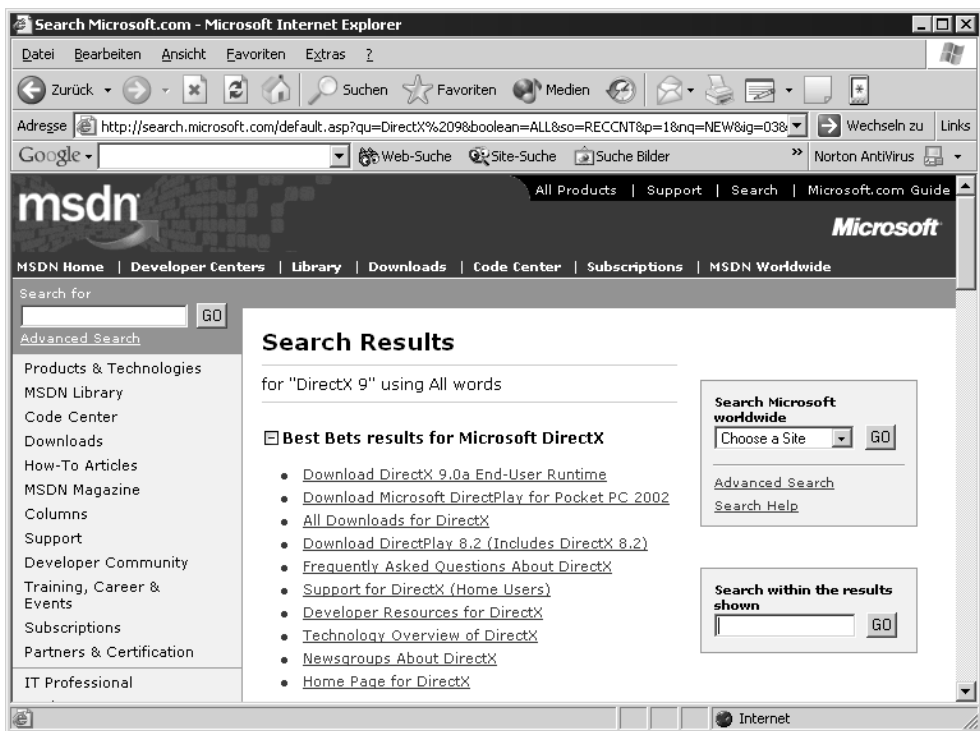


Abbildung 7: Ergebnis einer Suche nach »DirectX 9« bei *msdn.microsoft.com*

3 Grundlagen zu den im Buch verwendeten externen Technologien

Viele Probleme lassen sich nicht direkt über die Klassen des .NET Frameworks lösen. Das betrifft z.B. den im .NET Framework 1.0 und 1.1 komplett fehlenden Multimedia-Support, spezielle Windows-Aufgaben wie das Verschieben von Dateien und Ordner in den Papierkorb und andere Dinge, die in der Klassenbibliothek einfach fehlen (teilweise weil diese Features Windows-spezifisch sind und damit der Idee des betriebssystemunabhängigen Frameworks widersprechen würden). Zur Problemlösung müssen Sie häufig das Windows-API, COM-Komponenten, WMI-Klassen, ADSI oder andere externe Technologien einsetzen. Die folgenden Abschnitte erläutern wesentliche Dinge zum Einsatz dieser Technologien.

Das Windows-API

Das Windows-API¹ liefert mit (je nach Version) etwa 750 Funktionen und einigen wenigen COM-Komponenten die Basis für die Programmierung unter Windows. Bei der Programmierung unter Windows rufen die Funktionen und Methoden einer Programmiersprache auf der untersten Ebene meist Windows-API-Funktionen auf (was natürlich auch beim .NET Framework der Fall ist).

Obwohl das .NET Framework sehr umfangreich ist, müssen Sie in einigen Fällen Windows-API-Funktionen direkt aufrufen, nämlich dann, wenn die gesuchte Funktionalität nicht in den Klassen des .NET Frameworks, aber eben im API zu finden ist. Die folgenden Abschnitte klären einige dazu wichtige grundsätzliche Dinge.

Die Windows-API-Dokumentation

Die offizielle Dokumentation des Windows-API finden Sie auf der Microsoft-Website an der Adresse msdn.microsoft.com/library/en-us/winprog/winprog/windows_api_reference.asp. Suchen können Sie über das Suchfeld oben links auf der Seite.

Das Microsoft-Plattform-SDK

Da die MSDN-Suche auch Seiten findet, die nicht direkt zum Windows-API gehören, ist diese in vielen Fällen ineffizient. Häufig ist eine Recherche in einer Windows-Hilfe einfacher und schneller. Diese Hilfe erhalten Sie über das *Microsoft Platform SDK* (das »Software Development Kit« für die Entwicklung von Windows-Anwendungen mit dem Windows-API und C++). Leider ist dieses in der von Microsoft herunterladbaren Minimalvariante recht groß (ca. 500 MB) und beinhaltet viele Komponenten, die lediglich C++-Entwickler benötigen. Etwas weniger speicherhungrig ist die Plattform-SDK-Dokumentation, die mit Visual Studio .NET mitgeliefert wird, da diese lediglich die Dokumentation enthält und nicht die (für uns recht überflüssigen) Komponenten.

Wenn Sie Visual Studio .NET besitzen, können Sie die Dokumentation des Plattform-SDK über die Visual-Studio-Installations-CDs installieren, falls dies noch nicht installiert ist. Legen Sie dazu die erste Installations-CD ein, wählen Sie die Visual Studio .NET-Installation und dann den Link FEATURES HINZUFÜGEN / ÄNDERN. Unter dem Ordner DOKUMENTATION wählen Sie die Option PLATTFORM-SDK-DOKUMENTATION und installieren diese (Abbildung 8).

1. API = Application Programming Interface = Schnittstelle einer Anwendung für die Programmierung



Abbildung 8: Nachinstallation der Microsoft-Plattform-SDK-Dokumentation

Etwas problematisch bei der Visual Studio .NET-Variante ist, dass die Plattform-SDK-Dokumentation nicht in das Inhaltsverzeichnis der Hilfe integriert wird. Sie können lediglich im Index oder über die Suchfunktion der Hilfe nach bekannten SDK-Themen suchen (was aber in der Praxis meist ausreicht).

Alternativ (und wesentlich umfangreicher) können Sie das Plattform-SDK von Microsoft herunterladen und installieren. Dabei haben Sie auch die Möglichkeit, weitere SDKs zu speziellen Technologien wie z.B. DirectX zu installieren. Sie können die Installationsdateien des kompletten Plattform-SDK einzeln herunterladen (www.microsoft.com/msdownload/platformsdk/sdkupdate/psdk-full.htm). Da dieser Download sehr umfangreich ist und Sie in vielen Fällen nur das Basis-SDK benötigen, ist es vorteilhafter, das SDK über ein automatisches Update im Internet Explorer zu installieren (vorausgesetzt, Sie haben dabei keine Sicherheitsbedenken ...). Öffnen Sie dazu die Adresse www.microsoft.com/msdownload/platformsdk/sdkupdate. Wählen Sie in der linken Spalte den Link CORE SDK und auf der dann erscheinenden Seite den Link INSTALL THIS SDK um das Basis-SDK (mit der API-Dokumentation) zu installieren. Der Internet Explorer installiert daraufhin zunächst ein ActiveX-Steuerelement, das Ihre vorhandene Plattform-SDK-Installation erforscht (angeblich ohne das Übersenden von Informationen an Microsoft) und Ihnen die notwendigen und optionalen Updates zur Auswahl anbietet.

ACHTUNG

Beachten Sie, dass die Installation des Plattform-SDK mit Vorsicht zu genießen ist. In einigen Newsgroup-Beiträgen ist nachzulesen, dass Sie das Plattform-SDK *vor* dem .NET Framework-SDK installieren sollten. Das ist auch recht logisch, da es sich dabei um das ältere SDK handelt und dieses u.U. neuere Dateien überschreibt. Ich habe zudem die Erfahrung gemacht, dass mein Rechner nach der Installation nicht gerade stabiler wurde (um es vorsichtig auszudrücken). Aber das ist eine rein subjektive Erfahrung. Erzeugen Sie vor der Installation des Plattform-SDK also idealerweise ein Image Ihrer Systempartition (z. B. über *Drive Image*, www.driveimage.com), damit Sie diese im Fehlerfall wieder herstellen können. Unter Windows XP hilft u.U. auch ein System-Wiederherstellungspunkt.

Ein Hinweis zu Windows 98 und Me

Die API-Funktionen, die in diesem Buch beschrieben werden, gehören zu den 32-Bit-Funktionen. Windows 98 und Me kennen aber neben diesen auch die alten 16-Bit-Funktionen von Windows 3.x. Dummerweise wird die Konsole unter Windows 98 (und wahrscheinlich auch unter Windows Me) im 16-Bit-Modus ausgeführt. Wenn Sie also 32-Bit-API-Funktionen in einer Konsolenanwendung aufrufen und diese unter Windows 98 oder Me ausführen, erhalten Sie den Fehler »Diese Funktion ist nur im Win32-Modus gültig«. In einer Windows-Anwendung (mit Formularen) tritt dieser Fehler logischerweise nicht auf.

Aufruf von API-Funktionen über PInvoke

Windows-API-Funktionen werden über *PInvoke* (Platform Invocation) aufgerufen. PInvoke ermöglicht den Aufruf von Funktionen in unverwalteten DLLs, wie eben denen des Windows-API, und steht über die Klassen des Namensraums `System.Runtime.InteropServices` zur Verfügung.

API-Funktionen deklarieren Sie mit dem `DllImport`-Attribut (dessen Typ die Klasse `DllImportAttribute` ist), wobei Sie im Konstruktor den Dateinamen der DLL-Datei übergeben. Die DLLs des Windows-API können Sie ohne Pfad angeben, da diese im Windows-Systemordner gespeichert sind. Optional können Sie die Felder der `DllImportAttribute`-Klasse (siehe Tabelle 1) in der Attribut-Deklaration setzen um die Ausführung der Funktion zu steuern. Die Funktion deklarieren Sie dann mit dem `extern`-Modifizierer. Die Deklaration muss statisch erfolgen.

Eine einfache Deklaration sieht dann z.B. so aus:

```
[DllImport("winmm.dll", SetLastError=true)]  
public static extern int PlaySound(string pszSound, long hmod, int fdwSound);
```

In diesem Beispiel wird die API-Funktion `PlaySound` aus der DLL *winmm.dll* deklariert. Das Feld `SetLastError` des `DllImport`-Attributs wird auf `true` gesetzt, damit die Funktion bei einem Fehler den Fehlercode speichert (siehe »Umgehen mit API-Fehlern«).

Die API-Funktion können Sie innerhalb von C# dann aufrufen wie eine C#-Methode (allerdings ohne Klasse):

```
const int SND_FILENAME = 0x00020000;
PlaySound("c:\\windows\\media\\chimes.wav", 0, SND_FILENAME);
```

Das Problem des Herausfindens der verwendeten Konstanten (im Beispiel `SND_FILENAME`) kläre ich im Abschnitt »API-Konstanten-Werte«.

Im `DllImport`-Attribut können Sie die in Tabelle 1 angegebenen Eigenschaften definieren um den Aufruf der Funktion zu steuern.

Eigenschaft	Bedeutung
<code>CallingConvention</code>	gibt mit einem Wert der <code>CallingConvention</code> -Aufzählung die Aufrufkonvention der Funktionen in der DLL an. Möglich sind die Werte <code>CDecl</code> , <code>FastCall</code> , <code>StdCall</code> , <code>ThisCall</code> und <code>Winapi</code> . Die in DLLs fast immer eingesetzte Aufrufkonvention <code>StdCall</code> ist die Voreinstellung.
<code>CharSet</code>	bestimmt mit einem Wert der <code>CharSet</code> -Aufzählung, wie Zeichenketten an die Funktion übergeben werden: ► <code>Ansi</code>: Zeichenketten werden als 8-Bit-ASCII-Zeichen übergeben, ► <code>Auto</code>: Zeichenketten werden im Format des Zielsystems übergeben, ► <code>None</code>: veralteter Wert, der <code>Ansi</code> entspricht, ► <code>Unicode</code>: Zeichenketten werden als 16-Bit-Unicode-Zeichen übergeben. Die Voreinstellung ist <code>Ansi</code> . Diese Eigenschaft ist wichtig bei Funktionen, die mit Strings arbeiten (siehe »Strings übergeben«).
<code>EntryPoint</code>	An diesem Feld können Sie den Originalnamen der Funktion angeben, wenn Sie diese unter einem anderen Namen importieren wollen. Ein anderer Einsatz ist die Angabe der Ordinalzahl der Funktion (mit einem führenden #-Zeichen) für den Fall, dass Sie deren Namen nicht kennen.
<code>ExactSpelling</code>	Diese boolesche Eigenschaft gibt an, ob dem Namen der Funktion je nach Einstellung in <code>CharSet</code> automatisch ein »A« bzw. »W« angehängt wird. Damit können Sie festlegen, dass die zu <code>CharSet</code> passende Funktion aufgerufen wird. Die Standardeinstellung ist <code>false</code> .

Tabelle 1: Eigenschaften der `DllImportAttribute`-Klasse

Eigenschaft	Bedeutung
PreserveSig	Diese (spärlich und verwirrend dokumentierte ...) Eigenschaft ist für Funktionen interessant, die HRESULT-Werte zurückgeben. Ein HRESULT-Wert (der nur von COM-Methoden und COM-zugehörigen API-Funktionen zurückgegeben wird) ist ein 32-Bit-Integerwert, der einen Status meldet. Er enthält Informationen zum Schweregrad, zum Kontext, zum Verursacher und zum Status selbst. Einige HRESULT-Werte melden Fehler, S_OK (0) meldet eine fehlerfreie Ausführung. Daneben existieren weitere Konstanten wie S_TRUE oder S_FALSE, die einen Nicht-Fehler-Status melden. Bei COM-Methoden oder -Funktionen, die einen HRESULT-Wert zurückgeben, können Sie mit PreserveSig = false (true ist die Voreinstellung) festlegen, dass die Methode/Funktion bei einem HRESULT-Wert, der einen Fehler darstellt (was wohl am Schweregrad-Anteil des Werts erkannt wird) eine Ausnahme generiert. Die Signatur der Funktion bzw. Methode wird dann so verändert, dass kein HRESULT, sondern ein int-Wert zurückgegeben wird. Die Rückgabe enthält dann HRESULT-Statuswerte, die keinen Fehler darstellen. Bei einer fehlerfreien Ausführung können Sie den Status also aus dem zurückgegebenen int-Wert auslesen.
SetLastError	Viele API-Funktionen geben einen booleschen Wert zurück, der lediglich mit false darüber informiert, dass ein Fehler aufgetreten ist. Wenn Sie SetLastError auf true setzen (die Voreinstellung ist false), wird die API-Funktion aufgefordert, beim Eintreten eines Fehlers die Funktion SetLastError aufzurufen, die den Fehlercode für den aufrufenden Thread innerhalb des Windows-Systems zwischenspeichert. Über Marshal.GetLastWin32Error können Sie diesen Fehler dann auslesen (siehe »Umgehen mit API-Fehlern«).

Tabelle 1: Eigenschaften der DllImportAttribute-Klasse (Forts.)

Umsetzen von Datentypen (Marshalling)

Bei der Deklaration einer API-Funktion müssen Sie die Datentypen so angeben, dass diese mit den Datentypen der Funktion identisch sind. Geben Sie falsche Datentypen an, resultiert dies beim Aufruf der Funktion in einer Ausnahme oder im schlimmsten Fall in einem Programmabsturz (weil die Funktion Speicherbereiche verwendet, die nicht dem Programm gehören).

Beim Aufruf einer API-Funktion werden C#-Datentypen in API-Datentypen umgesetzt. Dieser Vorgang wird als *Marshalling* bezeichnet (die beste Übersetzung des Begriff »to marshal« ist wohl »geleiten«). Die meisten Datentypen der normalerweise in C oder C++ geschriebenen Funktionen besitzen eine direkte Entsprechung in C# und können deswegen ohne weitere Vorkehrungen eingesetzt werden. Einige API-Funktionen verwenden aber auch Datentypen, die in C# keine Entsprechung besitzen. Daneben kann es sein, dass C#-Datentypen bei der Weitergabe an die Funktion prinzipiell in mehrere C++-Datentypen umgesetzt werden können. Dies ist z.B. beim Datentyp `string` der Fall, der in die C++-Typen `LPStr`²,

2. LPStr = Long Pointer to String = Nullterminierter ASCII-String

LPWStr³, LPTStr⁴ oder BStr⁵ umgesetzt werden kann. Diese Typen besitzen ein Default-Marshalling-Verhalten. Der `string`-Typ wird z.B. standardmäßig in einen `BStr` umgesetzt.

Das Marshalling-Verhalten der einzelnen Typen beschreibt Microsoft in der .NET Framework-Dokumentation unter PROGRAMMIEREN MIT .NET FRAMEWORK / ZUSAMMENARBEIT MIT NICHT VERWALTETEM CODE / INTEROP-MARSHALLING / STANDARDMÄßIGES MARSHALLINGVERHALTEN / BLITZFÄHIGE UND NICHT BLITZFÄHIGE TYPEN.

Über das `MarshalAs`-Attribut können Sie das Marshalling-Verhalten von C#-Datentypen beeinflussen. Im Konstruktor dieses Attributs können Sie mit einem Wert der Aufzählung `UnmanagedType` den Zieldatentyp angeben. So können Sie z.B. bei der Deklaration einer API-Funktion, die mit Zeichenketten arbeitet, angeben, dass ein `string` je nach Zielsystem als `LPStr` oder `LPWStr` übergeben werden soll:

```
[DllImport("Kernel32.dll", SetLastError=true, CharSet=CharSet.Auto)
public static extern int GetDiskFreeSpaceEx(
    [MarshalAs(UnmanagedType.LPTStr)] string lpDirectoryName,
    ref ulong lpFreeBytesAvailable,
    ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);
```

Die Originaldeklaration der Funktion sieht übrigens so aus:

```
BOOL GetDiskFreeSpaceEx(
    LPCTSTR lpDirectoryName,           // directory name
    PULARGE_INTEGER lpFreeBytesAvailable, // bytes available to caller
    PULARGE_INTEGER lpTotalNumberOfBytes, // bytes on disk
    PULARGE_INTEGER lpTotalNumberOfFreeBytes // free bytes on disk
);
```

Da C# für die meisten C++-Datentypen direkte Entsprechungen besitzt, ist ein explizites Marshalling nur selten notwendig, weswegen ich an dieser Stelle auf die Auflistung der Konstanten der `UnmanagedType`-Aufzählung verzichte. Problematisch wird das Ganze bei der Übergabe von Strings, Arrays und Strukturen.

Die .NET-Entsprechungen der API-Typen

Wenn Sie eine API-Funktion in C# deklarieren wollen und nur die originale Dokumentation dieser Funktion besitzen, müssen Sie die API-Typen in passende C#-Typen umwandeln. Microsoft beschreibt die Entsprechungen der API-Typen in der .NET Framework-Dokumentation unter PROGRAMMIEREN MIT .NET FRAMEWORK / ZUSAMMENARBEIT MIT NICHT

-
3. LPWStr = Long Pointer to Wide String = Nullterminierter Unicode-String
 4. LPTStr = Nullterminierter String, der plattformabhängig als ASCII- oder Unicode-String verwaltet wird
 5. BStr = Basic String = Nullterminierter Unicode-String mit einem Präfix, der die Länge des Strings speichert

VERWALTETEM CODE / INTEROP-MARSHALLING / MARSHALLEN VON DATEN MIT PLATTFORM-AUFRUF / DATENTYPEN FÜR DEN PLATTFORMAUFRUF. Tabelle 2 fasst diese Informationen zusammen.

API-Typ	C#-Typ	Bemerkung
HANDLE	System.IntPtr	Bei der Programmierung mit dem Windows-API arbeiten Sie häufig mit Handles. Ein Handle repräsentiert ein Windows-Objekt wie z. B. eine geöffnete Datei, ein Fenster oder ein Steuerelement. In einer Windows.Forms-Anwendung können Sie den Handle eines Formulars oder Steuerelements aus dessen Eigenschaft <code>Handle</code> auslesen. API-Funktionen, die Windows-Objekte öffnen oder erzeugen, geben den Handle auf dieses Objekt zurück. API-Funktionen, die Objekte bearbeiten, erwarten den Handle in einem Argument. Der dazu verwendete Typ <code>HANDLE</code> ist in der Datei <code>wtypes.h</code> als Zeiger auf <code>void</code> (<code>void*</code>) definiert, also als 32- oder 64-Bit-Integer-Wert (je nach Plattform), der die Adresse eines Speicherbereichs mit einem beliebigen Datentyp verwaltet. Der Typ <code>IntPtr</code> wird automatisch in einen solchen Zeiger gemarshallt.
BYTE	byte	
SHORT	short	
WORD	ushort	
INT	int	
UINT	uint	
LONG	int	<code>LONG</code> ist als 32-Bit-Integerwert mit Vorzeichen definiert und wird deswegen nicht durch den 64 Bit großen <code>long</code> -Typ repräsentiert.
BOOL	int	Der im API vier Byte große <code>BOOL</code> -Typ kann nicht direkt in den nur ein Byte großen C#- <code>bool</code> -Typ umgesetzt werden. Deshalb sollten Sie für diesen Typ <code>int</code> verwenden. Sie können jedoch auch den <code>bool</code> -Typ marshallen (siehe »Boolesche Argumente, Felder und Rückgabewerte«).
DWORD	uint	
ULONG	uint	<code>ULONG</code> ist als 32-Bit-Integerwert ohne Vorzeichen definiert und wird deswegen nicht durch den 64 Bit großen <code>ulong</code> -Typ repräsentiert.
CHAR, LPSTR, LPWSTR, LPCWSTR	string oder System.Text.StringBuilder	Strings müssen korrekt gemarshallt werden, da diese in verschiedenen Varianten auftreten können. Funktionen, die einen String zurückgeben, müssen mit einem <code>StringBuilder</code> -Objekt aufgerufen werden. Informationen dazu finden Sie bei »Strings übergeben« und »Rückgabe von Strings«.
FLOAT	float	
DOUBLE	double	

Tabelle 2: C#-Entsprechungen der gängigen Windows-API-Typen

Viele Typen in API-Funktionen beginnen mit einem *LP* (z.B. *LPWORD*). *LP* steht für Long Pointer (32- oder 64-Bit-Zeiger). Ein solcher Typ ist also ein Zeiger auf den eigentlichen Typen. *LPWORD* ist z.B. deklariert als *WORD**. Handelt es sich bei diesem Typen um einen, dessen C#-Entsprechung ein Werttyp ist, müssen Sie entsprechende Argumente By Reference deklarieren (siehe »By-Reference-Argumente«). Strings, die ja Referenztypen sind, werden allerdings anders behandelt (siehe »Strings übergeben« und »Rückgabe von Strings«).

Hinweis

Die komplette Liste der Windows-API-Typen finden Sie in der Dokumentation des Plattform-SDK unter DEVELOPMENT GUIDES / WINDOWS API REFERENCE / WINDOWS DATA TYPES. Leider können Sie in der in Visual Studio .NET integrierten Plattform-SDK-Dokumentation nicht nach dieser Überschrift suchen. Suchen Sie im Index einfach nach einem der Windows-API-Typen (z.B. nach *BOOL*) um die Dokumentation der API-Typen zu finden.

Boolesche Argumente, Felder und Rückgabewerte

API-Funktionen, die boolesche Argumente oder Rückgabewerte besitzen, oder die boolesche Felder in Strukturen erwarten, arbeiten mit dem C-Typ *BOOL*. Dieser Typ ist (warum auch immer ...) definiert als 4-Byte-Wert, der den Wert 0 für *false* und einen Wert ungleich 0 für *true* speichert. Ein C#-*bool*-Typ verwaltet *true* und *false* zwar prinzipiell auf dieselbe Weise, ist aber nur ein Byte groß. Laut der Dokumentation wird *bool* in einen Wert der Größe 1, 2 oder 4 Byte (!) gemarshallt, der *true* in -1 oder 1 konvertiert. Wann welche Speichergröße verwendet wird ist leider nicht dokumentiert (bzw. habe ich die entsprechende Stelle nicht gefunden). Wenn Sie boolesche Argumente oder Strukturfelder deklarieren, sollten Sie diese also über das *MarshalAs*-Attribut explizit in den Typen *BOOL* marshallen. Ein Beispiel ist die Struktur *SHFILEOPSTRUCT*, die von der Funktion *SHFileOperation* verwendet wird:

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
    public IntPtr hwnd;
    public uint wFunc;
    public string pFrom;
    public string pTo;
    public short fFlags;
    [MarshalAs(UnmanagedType.Bool)]
    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
}
```

Den Rückgabewert von Funktionen, die einen *BOOL*-Wert zurückgeben, können Sie nicht marshallen. Diese Funktionen sollten Sie stattdessen mit einem *int* als Rückgabewert deklarieren und beim Aufruf überprüfen, ob dieser ungleich 0 ist. In einigen Beispielen ist hingegen zu sehen, dass dort der *bool*-Typ verwendet wird. Scheinbar ist es kein Problem, dass der 4-Byte-*BOOL*-Wert bei der Rückgabe auf den 1-Byte-*bool*-Typen zugewiesen wird. Bei meinen Versuchen sind bisher auf jeden Fall noch keine Probleme aufgetreten. Trotzdem sollten Sie

sich an die Microsoft-Empfehlung halten und `BOOL`-Rückgaben als `int` deklarieren. Damit stellen Sie sicher, dass auch eventuell mögliche Rückgabewerte, die den Byte-Bereich verlassen, korrekt ausgewertet werden.

Strings übergeben

Funktionen, die Strings übergeben bekommen oder zurückgeben, sind häufig in drei Varianten im Windows-API enthalten. Eine Variante, deren Name mit »A« endet, arbeitet mit ASCII-Strings, eine andere, deren Name mit »W« (Wide String) endet, arbeitet mit Unicode-Strings. Die dritte Variante, deren Namen kein Zeichen angehängt ist, verwendet automatisch die vom System benutzte Zeichencodierung. Das System selbst speichert Zeichenketten entweder als ASCII (Windows 95, 98) oder als Unicode (NT, 2000, XP). Sie können unter Windows 95 z.B. auch die W-Version einer Funktion aufrufen. Der übergebene Unicode-String wird dann aber intern nach ASCII konvertiert, was natürlich Zeit kostet. Umgekehrt können Sie unter Windows XP auch die A-Version aufrufen, was sogar zwei Konvertierungen verursacht (einmal vom Unicode-String im Programm nach ASCII, dann in der Funktion von ASCII nach Unicode).

Sicher gehen Sie, wenn Sie einfach die Version ohne angehängtes Zeichen verwenden und bei der Deklaration im `DllImport`-Attribut `Charset=CharSet.Auto` angeben. Dann stellen Sie sicher, dass die neutrale Variante der Funktion aufgerufen und die Zeichencodierung je nach Zielsystem automatisch bestimmt wird. Die explizite Angabe des `CharSet`-Felds ist notwendig, da die Voreinstellung dieses Feldes `CharSet.Ansi` ist. Über das Feld `ExactSpelling` des `DllImport`-Attributs können Sie übrigens mit `true` festlegen, dass je nach Einstellung des `CharSet`-Felds (auf `Ansi` oder `Unicode`) explizit die A- oder die W-Version aufgerufen wird, was aber wohl kaum notwendig sein sollte.

Ein weiteres Problem ist das Marshalling von Strings. Standardmäßig wird der `string`-Typ in einen `BStr` (Basic String) umgewandelt. Das hat übrigens den Grund, dass `BStr`-Strings in COM (für das `PInvoke` auch verwendet wird) ausschließlich verwendet werden. Ein solcher String verwaltet je nach der `CharSet`-Einstellung ASCII- oder Unicode-Zeichen, wird mit einem 0-Zeichen abgeschlossen und besitzt als Präfix einen 32-Bit-Integerwert, der die Länge des Strings verwaltet. Eine `BStr`-Variable ist ein Zeiger auf den Anfang des Strings (nicht auf den Anfang des Längen-Werts). Deshalb ist er mit einem `LPWStr` bzw. `LPStr` kompatibel.

Da ein `BStr` Unicode- oder ASCII-Zeichen verwalten kann, ist er auch beim Aufruf von API-Funktionen ideal geeignet:

```
[DllImport("Kernel32.dll", SetLastError=true, CharSet=CharSet.Auto)]
public static extern int GetDiskFreeSpaceEx(
    string lpDirectoryName,
    ref ulong lpFreeBytesAvailable,
    ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);
```

Da das Argument `lpDirectoryName` die Standard-Marshalling-Einstellung verwendet, wird es als `BStr` gemarshallt. Die Einstellung von `CharSet` auf `CharSet.Auto` bewirkt, dass je nach Zielsystem ASCII- oder Unicode-Zeichen übergeben werden. Der `BStr` wird entsprechend pas-

send als ASCII- oder Unicode-BStr übergeben. Diese vereinfachende Deklaration müsste eigentlich immer funktionieren. Die vier Byte Overhead des BStr gegenüber einem normalen String lassen sich dabei verschmerzen.

Alternativ können Sie auch über das `MarshalAs`-Attribut dafür sorgen, dass Strings korrekt übergeben werden. Dazu stehen Ihnen die Konstanten `LPStr`, `LPWStr` und `LPTStr` (Marshalling je nach Zielsystem als `LPStr` oder `LPWStr`) der `UnmanagedType`-Aufzählung zur Verfügung, die Sie im Konstruktor übergeben. Idealerweise setzen Sie eine Kombination von `CharSet=CharSet.Auto` im `DllImport`-Attribut mit `[MarshalAs(UnmanagedType.LPTStr)]` ein. Damit sorgen Sie dafür, dass unter Windows 98 ein `LPStr` und unter Windows NT, 2000 und XP ein `LPWStr` übergeben wird:

```
[DllImport("Kernel32.dll", SetLastError=true, CharSet=CharSet.Auto)
public static extern int GetDiskFreeSpaceEx(
    [MarshalAs(UnmanagedType.LPTStr)] string lpDirectoryName,
    ref ulong lpFreeBytesAvailable,
    ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);
```

Häufig werden API-Funktionen mit String-Argumenten in Beispielen auch so deklariert:

```
[DllImport("Kernel32.dll", SetLastError=true)]
public static extern int GetDiskFreeSpaceEx(
    [MarshalAs(UnmanagedType.LPStr)] string lpDirectoryName,
    ref ulong lpFreeBytesAvailable,
    ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);
```

Diese Variante funktioniert ebenfalls, da Strings als ASCII übergeben und in einen (ASCII-)LPStr gemarshallt werden. Eigenartigerweise wird der String auch unter Windows 2000 und XP korrekt übergeben, wo doch eigentlich die Unicode-Variante der Funktion aufgerufen werden sollte.

Rückgabe von Strings

Einige API-Funktionen geben Informationen als String zurück. Dazu gehört z.B. die Funktion `GetWindowsDirectory`, die den Namen des Windows-Ordners ermittelt. Diese Funktionen geben Strings nicht als Rückgabewert, sondern in Argumenten zurück. Der Grund dafür liegt darin, dass einige Programmiersprachen wie z.B. Visual Basic 6 mit dem in C++ üblichen Zeiger auf ein `char`-Array nichts anfangen können. Also werden Strings in den Speicherbereich übergebener String-Variablen geschrieben. `GetWindowsDirectory` ist im Original folgendermaßen deklariert:

```
UINT GetWindowsDirectory(
    LPTSTR lpBuffer, // buffer for Windows directory
    UINT uSize // size of directory buffer
);
```

Das Argument `lpBuffer` erwartet einen Zeiger (LP = Long Pointer) auf einen plattform-abhängigen String (TStr). Damit die Funktion weiß, wie viele Zeichen geschrieben werden können, wird diese Anzahl am Argument `uSize` übergeben. An diesem Argument können Sie aber nun keine C#-string-Instanz übergeben, da Strings in C# unveränderlich⁶ sind. Die Lösung ist die Verwendung einer `StringBuilder`-Instanz (aus dem Namensraum `System.Text`). Eine solche kann bei der Instanziierung ausreichend groß vorinitialisiert werden, behält den für den String reservierten Speicherbereich so lange, bis sie vom Garbage Collector zerstört wird, und wird so gemarshallt, dass ein Zeiger auf den gespeicherten String übergeben wird. Die C#-Deklaration der `GetWindowsDirectory`-Funktion sieht also so aus:

```
[DllImport("Kernel32.dll", SetLastError=true, CharSet=CharSet.Auto)]
public static extern uint GetWindowsDirectory(
    StringBuilder lpBuffer, uint uSize);
```

Beim Aufruf der Funktion müssen Sie die `StringBuilder`-Instanz ausreichend groß vorinitialisieren, sodass der Speicherbereich für den String reserviert wird. Sie müssen ein Zeichen mehr reservieren, als der String maximal groß werden kann, da API-Funktionen Strings immer mit einem 0-Zeichen abschließen. Am Argument `uSize` übergeben Sie dann die Länge des Strings.

ACHTUNG

Falls Sie die `StringBuilder`-Instanz nicht ausreichend groß initialisieren oder einen zu hohen Wert in `uSize` übergeben, resultiert dies darin, dass die API-Funktion versucht in einen Speicherbereich zu schreiben, der nicht zum `StringBuilder`-Objekt gehört. Im günstigsten Fall gehört der Speicher nicht zu Ihrem Programm, was in einem Windows-Ausnahmefehler (mit der Meldung »Der Vorgang written konnte nicht ausgeführt werden«) und einem Programmabsturz resultiert. Im ungünstigsten Fall gehört der Speicher zu Ihrem Programm und die Funktion überschreibt den Wert anderer Variablen oder Eigenschaften, was zu erheblichen und schwer findbaren Fehlern führt.

API-Funktionen mit String-Rückgabe geben die Länge des ermittelten Strings zurück. Daran können Sie erkennen, ob überhaupt ein String ermittelt wurde. Der Aufruf der `GetWindowsDirectory`-Funktion sieht also so aus:

```
StringBuilder buffer = new StringBuilder(261);
if (GetWindowsDirectory(buffer, 261) > 0)
    Console.WriteLine("Windows-Ordner: {0}", buffer.ToString());
else
    Console.WriteLine("Fehler bei der Ermittlung des Windows-Ordners");
```

6. Der Speicherbereich, den eine `string`-Instanz belegt, kann tatsächlich nicht verändert werden. Wenn Sie einem String Zeichen anfügen, legt die CLR einen neuen Speicherbereich an, kopiert den alten String dort hinein und hängt die neuen Zeichen hinten an. Ähnlich geht die CLR vor wenn Sie eine Stringvariable mit einem neuen Wert versehen.

By-Reference-Argumente

Einige Funktionen wie z.B. `GetUserName` geben auch andere Daten als Strings in By-Reference-Argumenten zurück. An der Deklaration erkennen Sie dies meist daran, dass das Argument mit einem Typen deklariert ist, der mit *LP* (Long Pointer) beginnt. `GetUserName` wird z.B. im Original so dokumentiert:

```
BOOL GetUserName(  
    LPTSTR lpBuffer,    // name buffer  
    LPDWORD nSize       // size of name buffer  
);
```

Handelt es sich bei diesen Argumenten um C#-Werttypen, müssen diese mit `ref` deklariert werden, damit die Übergabe By Reference erfolgt. Handelt es sich hingegen um Referenztypen (was aber eigentlich nicht vorkommt), werden die Argumente natürlich nicht mit `ref` deklariert.

Die C#-Deklaration der `GetUserName`-Funktion sieht also folgendermaßen aus:

```
[DllImport("Advapi32.dll", SetLastError=true)]  
public static extern int GetUserName(  
    StringBuilder lpBuffer, ref uint nSize);
```

Beim Aufruf müssen Sie bei diesem Beispiel zunächst wieder eine `StringBuilder`-Instanz für den zurückgegebenen String erzeugen. Da das Argument `nSize` mit `ref` deklariert ist, muss an diesem Argument eine passende Variable übergeben werden. Bei `GetUserName` muss diese Variable vor dem Aufruf auch mit der Größe des Strings im `StringBuilder`-Objekt initialisiert werden:

```
StringBuilder userName = new StringBuilder(1024);  
uint size = 1024;  
if (GetUserName(userName, ref size) != 0)  
    Console.WriteLine(userName.ToString());  
else  
    Console.WriteLine("Fehler beim Aufruf von GetUserName");
```

Nach dem Aufruf enthält die Variable in diesem Beispiel die Länge des ermittelten Strings, was aber für das Beispiel vollkommen unwichtig ist.

Die Übergabe von Strukturen und feste Strings

Viele API-Funktionen arbeiten mit Strukturen. Einige Strukturen besitzen Felder, die Zeichenketten verwalten. Da Strukturen in API-Funktionen immer eine definierte Größe besitzen müssen, sind String-Felder mit einer festen Länge definiert. Die Funktion `GetVersionEx`,

die Informationen zur Windows-Version zurückliefert, ist ein Beispiel dafür. Diese Funktion ist im Original folgendermaßen deklariert:

```
BOOL GetVersionEx(  
    LPOSVERSIONINFO lpVersionInfo // version information  
);
```

GetVersionEx erwartet eine Struktur vom Typ OSVERSIONINFO, die folgendermaßen deklariert ist:

```
typedef struct _OSVERSIONINFO{  
    DWORD dwOSVersionInfoSize;  
    DWORD dwMajorVersion;  
    DWORD dwMinorVersion;  
    DWORD dwBuildNumber;  
    DWORD dwPlatformId;  
    TCHAR szCSDVersion[ 128 ];  
} OSVERSIONINFO;
```

Diese Struktur müssen Sie in C# nachbilden. Dazu müssen Sie zunächst im `StructLayout`-Attribut festlegen, dass die Felder der Struktur (anders als normalerweise unter .NET) sequenziell im Speicher angelegt werden. Dazu übergeben Sie diesem Attribut im Konstruktor den Wert `LayoutKind.Sequential`. Optional können Sie zusätzlich im Feld `CharSet` angeben, in welcher Form Zeichenketten übergeben werden (siehe »Strings übergeben«).

Feste Zeichenketten werden dann als `string` deklariert. Über das `MarshalAs`-Attribut legen Sie aber fest, dass der String By Value übergeben wird, indem Sie im Konstruktor den Wert `UnmanagedType.ByValTStr` übergeben, und definieren über das Feld `Size` die feste Größe des Strings. Das Marshalling als `ByValTStr` bewirkt übrigens auch, dass die Zeichenkette je nach Einstellung des `CharSet`-Feldes des `StructLayout`-Attributs als ASCII- oder Unicode-String übergeben wird. Die Deklaration der `OSVERSIONINFO`-Struktur sieht in C# also so aus:

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Ansi)]  
public struct OSVERSIONINFO  
{  
    public uint dwOSVersionInfoSize;  
    public uint dwMajorVersion;  
    public uint dwMinorVersion;  
    public uint dwBuildNumber;  
    public uint dwPlatformId;  
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst=128)]  
    public string szCSDVersion;  
}
```

Die Größe fester Strings wird immer in Byte angegeben. Im Beispiel ist der String 128 Byte groß. Deshalb müssen Strings in Strukturen immer als ASCII-String gemarshallt werden. Würden Sie diese als Unicode-String marshallen, wäre das Feld für den String zu klein definiert und der Aufruf der Funktion würde zum API-Fehler 122 (»Der an einen Systemaufruf übergebene Datenbereich ist zu klein«) führen. Im Beispiel habe ich deswegen im `StructLayout-Attribut` explizit `CharSet=CharSet.Ansi` angegeben (obwohl das die Voreinstellung ist).

`GetVersionEx` wird dann folgendermaßen deklariert:

```
[DllImport("kernel32.dll")]
public static extern int GetVersionEx(ref OSVERSIONINFO lpVersionInfo);
```

Das Argument `lpVersionInfo` wird mit `ref` deklariert, weil die Struktur By Reference übergeben wird. Wenn Sie sich die originale Deklaration der Funktion anschauen, erkennen Sie dies daran, dass der Datentyp `LPOSVERSIONINFO` mit *LP* beginnt. Natürlich hilft auch die Dokumentation der Funktion bei der Ermittlung der Art der Übergabe.

Wenn Sie die Funktion dann aufrufen wollen, müssen Sie zunächst eine Instanz der Struktur erzeugen:

```
OSVERSIONINFO vi = new OSVERSIONINFO();
```

Den festen String müssen Sie nicht erzeugen, das erledigt die CLR automatisch. Sie müssen allerdings in einem dafür vorgesehenen Feld der Struktur deren tatsächliche Speichergröße angeben, damit die API-Funktion die Felder der Struktur korrekt auswerten kann. Die Speichergröße ermitteln Sie über `Marshal.SizeOf`:

```
vi.dwOSVersionInfoSize = Marshal.SizeOf(typeof(OSVERSIONINFO));
```

Nun können Sie die Funktion aufrufen:

```
if (GetVersionEx(ref vi) != 0)
{
    Console.WriteLine("Windows-Version: {0}.{1}.{2}",
        vi.dwMajorVersion, vi.dwMinorVersion, vi.dwBuildNumber);
    Console.WriteLine("Service-Pack: {0}", vi.szCSDVersion);
}
else
    Console.WriteLine("Fehler beim Aufruf von GetVersionEx");
```

API-Konstanten-Werte

An den Argumenten von API-Funktionen werden häufig Konstanten übergeben. Diese werden in der Dokumentation der jeweiligen Funktion beschrieben. Die Funktion `SHFileOperation`, die für verschiedene Datei- und Ordneroperationen verwendet werden kann, erwartet am einzigen Argument z.B. eine `SHFILEOPSTRUCT`-Struktur, in deren Feld `wFunc` die auszuführende Operation definiert wird. In der Dokumentation der Struktur finden Sie eine Erläuterung der Bedeutung der Konstanten. Die Konstante `FO_COPY` führt z.B. dazu, dass eine Datei oder ein Ordner kopiert wird, `FO_DELETE` führt zum Löschen einer Datei bzw. eines Ordners.

Das Problem ist nun, dass die Dokumentation nicht die Werte der Konstanten angibt. Diese müssen Sie aber kennen um die Konstanten deklarieren zu können.

Eine mir bekannte Lösung des Problems ist das Nachschauen in den Headerdateien, die mit Visual C++ oder Visual C++.NET mitgeliefert werden. In irgendeiner dieser Headerdateien sind die gesuchten Konstanten deklariert. Wenn Sie C++.NET installiert haben, finden Sie die Headerdateien des Windows-SDK normalerweise im Ordner `C:\Programme\Microsoft Visual Studio .NET\VC7\PlatformSDK\Include`. Suchen Sie über die Windows-Suchfunktion einfach in allen Dateien mit der Endung `.h` nach einer der Konstanten, deren Wert Sie benötigen. Die von `SHFileOperation` verwendeten Konstanten finden Sie z.B. in der Datei `Shell-API.h`:

<code>#define FO_MOVE</code>	<code>0x0001</code>
<code>#define FO_COPY</code>	<code>0x0002</code>
<code>#define FO_DELETE</code>	<code>0x0003</code>
<code>#define FO_RENAME</code>	<code>0x0004</code>

Die C++-Deklaration müssen Sie dann nur noch nach C# umsetzen:

```
private const int FO_MOVE = 0x0001;
private const int FO_COPY = 0x0002;
private const int FO_DELETE = 0x0003;
private const int FO_RENAME = 0x0004;
```

Umgehen mit API-Fehlern

Windows-API-Funktionen generieren grundsätzlich keine Ausnahmen, sondern liefern einen numerischen Fehlercode zurück. Einige Funktionen, wie z.B. `SHFileOperation`, liefern diesen Code direkt als Rückgabewert. Andere Funktionen, wie z.B. `GetDiskFreeSpaceEx`, geben einen `BOOL` Wert zurück (einen Wert ungleich Null bei Erfolg, Null beim Auftreten eines Fehlers). Diese Funktionen rufen intern meist die API-Funktion `SetLastError` auf, die den Fehlercode Windows-intern speichert. Ein C++-Programmierer würde dann `GetLastError` aufrufen, um diesen Fehlercode auszulesen. Ein C#-Programmierer sollte dazu allerdings besser die Methode `GetLastWin32Error` der `Marshal`-Klasse aus dem Namensraum `System.Runtime.InteropServices` verwenden. Der Grund liegt darin, dass das .NET Framework intern weitere API-Funktionen aufrufen kann, die den Fehler zurücksetzen könnten. `GetLastWin32Error` stellt sicher, dass Sie den bei *Ihrem* API-Aufruf gesetzten Fehler erhalten.

Voraussetzung dafür, dass die API-Funktion `SetLastError` aufruft, ist allerdings, dass Sie das Feld `SetLastError` im `DllImport`-Attribut auf `true` setzen.

Das folgende Beispiel deklariert die Funktion `GetDiskFreeSpaceEx`, über die Sie Größen-Informationen zu einem Laufwerk auslesen können:

```
[DllImport("Kernel32.dll", SetLastError=true, CharSet=CharSet.Auto)]
public static extern int GetDiskFreeSpaceEx(
    string lpDirectoryName,
    ref ulong lpFreeBytesAvailable,
    ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);
```

Wenn Sie beim Aufruf z.B. ein nicht vorhandenes Laufwerk übergeben:

```
ulong userSpace = 0, totalSize = 0, totalSpace = 0;
if (GetDiskFreeSpaceEx("x:", ref userSpace, ref totalSize,
    ref totalSpace) == 0)
{
```

gibt diese Funktion 0 zurück. Dann können Sie den API-Fehler auslesen:

```
int apiError =
    System.Runtime.InteropServices.Marshal.GetLastWin32Error();
```

In diesem Fall resultiert der Fehlercode 3, der für den Fehler »Das System kann den angegebenen Pfad nicht finden« steht.

Umsetzen von API-Fehlercodes in passende Fehlerbeschreibungen

Beim Einsatz von API-Funktionen nutzt der numerische Fehlercode, den viele Funktionen zurückliefern, weder dem Programmierer noch dem Anwender. Sie können zwar den Fehlercode in der API-Dokumentation nachlesen (wenn Sie diesen finden 😊), der Aufwand dazu ist aber für den Programmierer und erst recht für den Anwender eindeutig zu hoch. Professionelle Programme sollten statt des Fehlercodes eine aussagekräftige Fehlermeldung liefern.

Und das ist möglich. Den von einer API-Funktion direkt zurückgegebenen oder über `GetLastWin32Error` ermittelten Fehlercode können Sie (meist) über die API-Funktion `FormatMessage` in eine Fehlerbeschreibung umwandeln. Eine kleine Einschränkung ist, dass einige Funktionen (wie z.B. `SHFileOperation`) leider einen speziellen, oft undokumentierten Fehlercode zurückgeben, den Sie mit `FormatMessage` nicht sinnvoll umsetzen können. Für API-Funktionen, die beim Eintritt eines Fehlers intern `SetLastError` aufrufen, sollte `FormatMessage` allerdings problemlos funktionieren. Lesen Sie gegebenenfalls in der Dokumentation der aufgerufenen API-Funktion nach.

Die Deklaration der `FormatMessage`-Funktion sieht folgendermaßen aus:

```
[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
    int dwMessageId, int dwLanguageId, StringBuilder lpBuffer, int nSize,
    string []Arguments);
```

Am Argument `dwFlags` können Sie spezielle Flags übergeben, die die Funktionsweise von `FormatMessage` beeinflussen. Für die Auswertung der meisten API-Fehler reicht die Angabe der Konstante `FORMAT_MESSAGE_FROM_SYSTEM` (`0x1000`), die bewirkt, dass die Beschreibung aus der System-Meldungs-Tabelle ausgelesen wird. Ist die Fehlermeldung nicht in einer System-DLL, sondern in einer nicht zum System gehörenden DLL-Datei gespeichert, müssen Sie das Flag `FORMAT_MESSAGE_FROM_HMODULE` (`0x0800`) angeben, um die Meldungs-Tabelle eines »Moduls« (der DLL-Datei) auszulesen. In diesem Fall geben Sie am Argument `lpSource` den Windows-Handle zum Modul an. Diesen Handle können Sie über die API-Funktion `GetModuleHandle` ermitteln, die folgendermaßen deklariert wird:

```
[DllImport("kernel32.dll")]
static extern IntPtr GetModuleHandle(string lpFileName);
```

`GetModuleHandle` setzt voraus, dass die DLL-Datei in den Prozessraum der Anwendung geladen ist, was aber der Fall ist, wenn Ihr Programm eine Funktion der DLL-Datei aufruft (die ja den auszuwertenden Fehler verursacht).

Beim Auslesen der Systemtabelle übergeben Sie am Argument `lpSource` den Wert (`IntPtr`)0 (laut der Dokumentation eigentlich `null`, aber das ist bei der Deklaration als `IntPtr` nicht möglich). Das Argument `lpSource` ist in der Funktion als Zeiger auf `void` (spezieller C-Datentyp, der mit allen anderen Datentypen belegt werden kann) deklariert. Deshalb können Sie in C++ an diesem Argument entweder einen Zeiger auf eine `int`-Variable übergeben, die den Handle des Moduls verwaltet, oder einen Zeiger auf einen C-String, der die zu formatierende Meldung speichert. Da für uns nur die erste Variante interessant ist, habe ich dieses Argument als `IntPtr` (Zeiger auf `int`) deklariert.

Am Argument `dwMessageId` übergeben Sie dann den Fehlercode. In `dwLanguageId` können Sie eine Sprach-Id für die Sprache der Meldung angeben. Wenn Sie 0 angeben, erfolgt die Ausgabe in der Systemsprache. Der Wert `0x0409` steht z.B. für das amerikanische Englisch. Voraussetzung dafür, dass die Meldung in einer anderen Sprache ausgegeben wird, ist, dass diese Sprache auch installiert bzw. verfügbar ist. Die verwendbaren Konstanten finden Sie in der SDK-Dokumentation, wenn Sie nach »Table of Language Identifiers« suchen.

Am Argument `lpBuffer` übergeben Sie ein ausreichend groß dimensioniertes `StringBuilder`-Objekt (aus dem Namensraum `System.Text`). In `nSize` übergeben Sie die Initialgröße dieses Objekts. Achten Sie darauf, dass das `StringBuilder`-Objekt auch wirklich so groß initialisiert ist, wie Sie in `nSize` angeben, da ansonsten ein Windows-Ausnahmefehler droht, wenn die API-Funktion in einen undefinierten Speicherbereich schreibt.

Das letzte Argument ist für unsere Zwecke nicht interessant: `FormatMessage` kann ähnlich `String.Format` auch Zeichenketten mit festgelegten Platzhaltern formatieren, in die die Inhalte des Arrays eingetragen werden. Übergeben Sie hier einfach `null`.

Das Auslesen der Meldung eines API-Fehlers, die in der Systemtabelle gespeichert ist, sieht folgendermaßen aus:

```
int apiError = Marshal.GetLastWin32Error();

// StringBuilder erzeugen
StringBuilder errorMessage = new StringBuilder(1024);

// Beschreibung für den Fehlercode ermitteln und damit eine Ausnahme werfen
if (FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, IntPtr(0), apiError, 0,
    errorMessage, 1024, null) > 0)
    throw new Exception(message.ToString());
else
    throw new Exception("API-Fehler " + apiError);
```

Listing 1: Auswerten eines API-Fehlercodes, dessen Meldung in der System-Meldungstabelle gespeichert ist

Listing 2 zeigt das Auslesen einer Fehlermeldung für den Fall, dass diese nicht in der System-Meldungstabelle gespeichert ist (was bei DLL-Dateien der Fall ist, die nicht zum eigentlichen System gehören). Das Beispiel ruft die Funktion `InternetGetConnectedState` aus der DLL `wininet.dll` auf, die den aktuellen Status der Internetverbindung zurückgibt:

```
/* Deklaration der Funktion InternetGetConnectedState */
[DllImport("wininet.dll")]
public static extern int InternetGetConnectedState(out int flags,
    int reserved);

...

int flags;
if (InternetGetConnectedState(out flags, 0) == 0)
{
    int apiError = Marshal.GetLastWin32Error();

    // Den Text des Fehlers aus der Datei wininet.dll auslesen und damit eine
    // Ausnahme werfen
    IntPtr hModule = GetModuleHandle("wininet.dll");
    StringBuilder message = new StringBuilder(1024);
    if (FormatMessage(FORMAT_MESSAGE_FROM_HMODULE, hModule, apiError, 0,
        message, 1024, null) > 0)
        throw new Exception(message.ToString());
    else
        throw new Exception("API-Fehler " + apiError);
}
```

Listing 2: Auswerten eines API-Fehlercodes, dessen Meldung in einer speziellen DLL-Datei gespeichert ist

COM

COM (Component Object Model) ist die veraltete Microsoft-Komponenten-Technologie, die in .NET-Programmen durch Klassenbibliotheks-Assemblies ersetzt ist. COM-Komponenten spielen aber auch heute noch eine Rolle, besonders wenn es darum geht, Office-Programme fernzusteuern. Word, Excel, Outlook und Co. stellen (wenigstens bis zur Version XP) ihre Funktionalität ausschließlich über solche Komponenten zur Verfügung.

Eine COM-Komponente speichert ähnlich einer Assembly Klassen (als einfache Klassen oder als Steuerelemente), die in Anwendungen, die die Komponente referenzieren, verwendet werden können. Die wesentlichen Unterschiede zu .NET-Assemblies sind, dass COM-Komponenten in der Windows-Registry registriert sein müssen und nur einmal (in genau einer Version) auf dem Rechner vorkommen können. Daneben setzen COM-Komponenten spezielle COM-Datentypen ein, die u.U. in .NET-Typen konvertiert werden müssen. Eine weitere Besonderheit ist, dass die Methoden von COM-Klassen nicht überladen werden können und mit optionalen Argumenten arbeiten, für die ein .NET-Programm einen speziellen Wert eintragen muss, wenn diese nicht belegt werden sollen.

Tip

Sehr ausführliche Informationen zur Verwendung von COM-Komponenten in .NET finden Sie an der Adresse www.codeproject.com/dotnet/cominterop.asp.

COM-Komponenten können mit früher oder später Bindung verwendet werden. Frühe Bindung bedeutet, dass dem Compiler zum Kompilierungszeitpunkt alle verwendeten Typen bekannt sind. Damit kann er den Aufruf von Methoden bzw. den Zugriff auf Eigenschaften (Eigenschaften werden im COM-Modell allerdings immer über Methoden zum Setzen und Lesen implementiert) fest in die Assembly einbetten und beim Kompilieren Syntaxprüfungen vornehmen. Bei der späten Bindung werden die Methodenaufrufe erst zur Programmausführzeit ermittelt.

Methodenaufrufe sind mit früher Bindung ein wenig schneller als mit später Bindung (was aber in der Praxis meist unerheblich ist). Der größte Vorteil ist, dass in Visual Studio Intellisense mit früher Bindung funktioniert und dass alle verwendeten Typen und Aufzählungen bekannt sind. Ein Nachteil der frühen Bindung ist, dass Programme, die eine bestimmte Version einer COM-Komponente referenzieren, mit älteren und manchmal auch mit neueren Versionen nicht unbedingt laufen. Wenn Sie z.B. eine .NET-Anwendung entwickeln, die Outlook einsetzt um Mails zu versenden (siehe Rezept 170), diese unter der Verwendung von Outlook XP programmieren und das Programm später auf einem Rechner ausführen, auf dem Outlook 97 installiert ist, führt dies zu einer Ausnahme bei der Erzeugung der Outlook-Instanz (Abbildung 9).

Microsoft empfiehlt, Programme immer mit der niedrigst möglichen Version einer COM-Komponente zu entwickeln, damit bei der Ausführung des Programms keine Versions-Probleme auftreten. Manchmal ist das aber einfach nicht möglich, z.B. wenn Sie auf Ihrem Computer Office 97 einfach nicht installiert haben, das Programm beim Kunden aber unter Office 97 ausgeführt werden soll. Um diese Probleme zu lösen, können Sie die späte Bindung verwenden (siehe »COM-Komponenten mit später Bindung verwenden«).



Abbildung 9: Fehler bei der Ausführung eines Programms, das mit Outlook XP entwickelt wurde, auf einem Rechner, auf dem lediglich Outlook 97 installiert ist

COM-Komponenten mit früher Bindung verwenden

Zur Verwendung der frühen Bindung benötigen Sie eine »Wrapper⁷«-Assembly, die die COM-Typen in .NET-Typen umsetzt. Diese Assembly können Sie dann in einem C#-Programm referenzieren, um die COM-Klassen zu verwenden. Sie müssen die Wrapper-Assembly aber nicht selbst programmieren (was sehr aufwändig wäre). Um diese Assembly zu erzeugen können Sie in Visual Studio .NET einfach eine Referenz auf die COM-Komponente anlegen, wie es Abbildung 10 für die Komponente *Microsoft Scripting Runtime* zeigt.

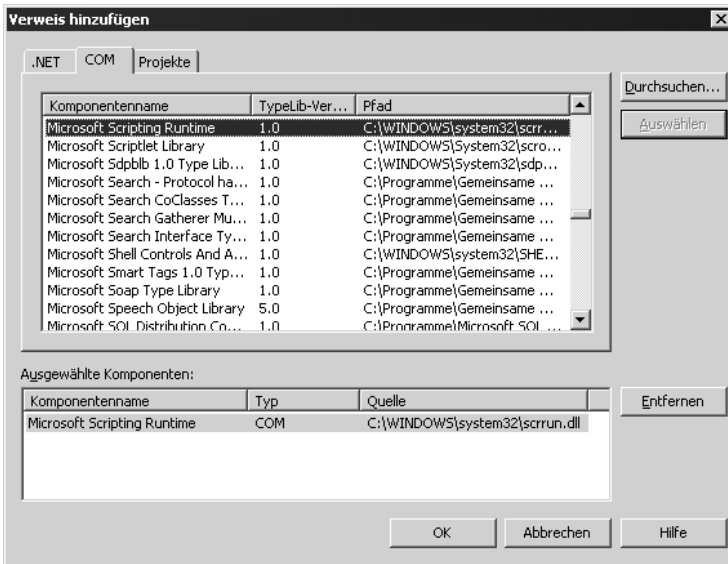


Abbildung 10: Anlegen eines Verweises auf die COM-Komponente *Microsoft Scripting Runtime*

Visual Studio .NET erzeugt die für die COM-Komponente (die u.U. von anderen COM-Komponenten abhängig ist) notwendigen Wrapper-Assemblies automatisch, legt diese im Ordner für die binären Dateien des Projekts ab (normalerweise *bin/debug* bzw. *bin/release*) und referenziert diese für das Projekt.

Wenn Sie ohne Visual Studio .NET arbeiten, verwenden Sie das Tool *tlbimp.exe* (Type Library Import) zur Erzeugung der Wrapper-Assemblies. Dieses Tool finden Sie im *bin*-Ordner des

7. Englisch für »Umschlag«

.NET Framework-SDK. Zum Aufruf müssen Sie lediglich wissen, wie die Datei heißt, die die Typen der COM-Komponente verwaltet (was in der Praxis oft nicht leicht herauszufinden ist). Das kann im einfachsten Fall die *dll*- oder *exe*-Datei der Komponente selbst oder auch eine externe Datei mit der Endung *tlb* (Type Library) oder *olb* (Object Library) sein. Wenn Sie Office oder Visual Basic 6 besitzen, können Sie im dortigen Visual Basic-Editor einen Verweis anlegen und sehen im Verweisfenster dann den Namen der DLL-Datei (Abbildung 11).

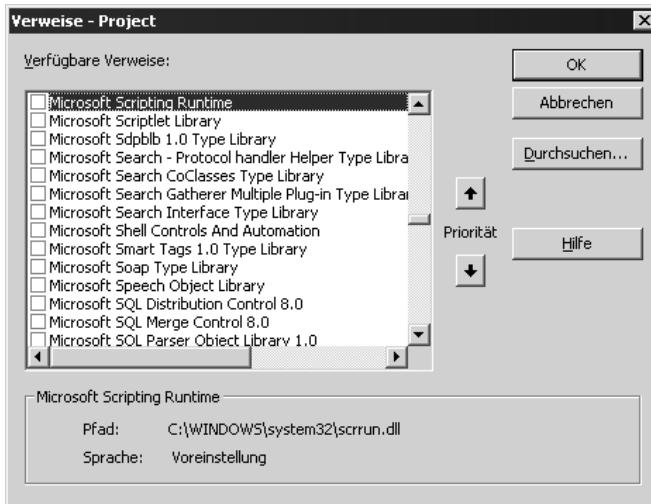


Abbildung 11: Der Verweisdialog von Office XP zeigt den Namen der Typbibliotheks-Datei einer COM-Komponente an

Nun können Sie über *tlbimp.exe* die Wrapper-Assemblies erzeugen:

```
tlbimp scrrun.dll out:/Interop.Scripting.dll /namespace:Scripting
```

Dieses Beispiel vergibt über die *out*-Option einen definierten Namen für die Assembly und weist die enthaltenen Klassen dem Namensraum *Scripting* zu (wie es auch Visual Studio .NET macht). Zusätzliche Wrapper-Assemblies für COM-Komponenten, von denen die zu verwendende Komponente abhängig ist, erzeugt *tlbimp* übrigens automatisch.

Die so erzeugten Assemblies referenzieren Sie dann beim Kompilieren der Anwendung:

```
csc Demo.cs /r:Interop.Scripting.dll
```

Im Programm können Sie COM-Klassen (fast) verwenden wie normale .NET-Klassen. Das folgende Beispiel setzt die *FileSystemObjectClass*-Klasse der Scripting-Bibliothek zur Ausgabe von Informationen zu einem Laufwerk ein:

```
// FileSystemObjectClass-Instanz erzeugen
Scripting.FileSystemObjectClass fso = new Scripting.FileSystemObjectClass();

// Laufwerk referenzieren
Scripting.Drive drive = fso.GetDrive("c:");

// Informationen zum Laufwerk ausgeben
Console.WriteLine("Freier Platz: {0}", drive.FreeSpace);
Console.WriteLine("Größe: {0}", drive.TotalSize);
```

Listing 3: Anwendung einer COM-Klasse

Einige Methoden von COM-Klassen arbeiten mit optionalen Argumenten, die in echten COM-Anwendungen (wie Visual-Basic-6- oder Office XP-Programmen) oder auch in Visual Basic .NET einfach weggelassen werden können. Für weggelassene Argumente verwendet die COM-Methode dann in der Regel Voreinstellungen. Ein Beispiel dafür ist die Methode `Open` der Klasse `Word.ApplicationClass` aus der COM-Komponente *Microsoft Word x.x Object Library* (normalerweise `C:\Programme\Microsoft Office\Office10\msword.olb`). Abbildung 12 zeigt die Syntax dieser Methode im Visual-Basic-Editor von Word XP. Optionale Argumente werden bei dieser Syntax mit eckigen Klammern gekennzeichnet. Der Datentyp der Argumente ist `Variant`, ein dem `object`-Typ ähnlicher Typ, der alle anderen Typen aufnehmen kann.

```
Documents.Open(
    Open(FileName, [ConfirmConversions], [ReadOnly], [AddToRecentFiles],
    [PasswordDocument], [PasswordTemplate], [Revert], [WritePasswordDocument],
    [WritePasswordTemplate], [Format], [Encoding], [Visible], [OpenAndRepair],
    [DocumentDirection], [NoEncodingDialog]) As Document
```

Abbildung 12: Eine COM-Methode mit optionalen Argumenten im Word-Visual-Basic-Editor

Eine weitere Besonderheit vieler COM-Methoden wie `Open` ist, dass die Argumente `By Reference` deklariert sind (was am Fehlen des Schlüsselworts `ByVal` erkennbar ist). Die `By-Reference`-Übergabe wurde von Microsoft in vielen COM-Methoden deswegen bevorzugt, weil damit die Übergabe von Variablen performanter ausgeführt wird als bei einer `By-Value`-Übergabe. Für den eigentlichen Zweck – die Rückgabe von Werten – wird `By-Reference` im COM-Modell kaum eingesetzt.

Abbildung 13 zeigt die Syntax der `Open`-Methode der `Word-AppicationClass` Methode in Visual Studio .NET. Der `Variant`-Datentyp wurde in den passenden `object`-Typ umgewandelt und die Argumente sind aufgrund der `By-Reference`-Übergabe mit `ref` deklariert. Optionale Argumente sind, da diese in .NET nicht möglich sind, nicht vorhanden.

Wenn Sie eine Methode mit optionalen Argumenten wie die `Open`-Methode aufrufen, können Sie an Stelle der optionalen Argumente natürlich einen passenden Wert eintragen, was in der Praxis aber sehr mühselig und meist auch unnötig ist. Stattdessen können Sie einfach den speziellen Wert `System.Reflection.Missing.Value` einsetzen, der beim Aufruf von COM-Methoden dafür steht, dass an einem optionalen Argument nichts übergeben wird.

```
word.Documents.Open(
```

```
Word.Document Documents.Open (ref object FileName, ref object ConfirmConversions, ref object ReadOnly,
    ref object AddToRecentFiles, ref object PasswordDocument, ref object PasswordTemplate,
    ref object Revert, ref object WritePasswordDocument, ref object WritePasswordTemplate,
    ref object Format, ref object Encoding, ref object Visible, ref object OpenAndRepair,
    ref object DocumentDirection, ref object NoEncodingDialog)
```

Abbildung 13: Eine COM-Methode mit optionalen Argumenten in Visual Studio .NET

Da die Argumente der `Open`-Methode als `object` und By-Reference deklariert sind, können Sie diesen Wert jedoch nicht direkt einsetzen, sondern müssen dazu eine `object`-Variable verwenden, wie es Listing 4 zeigt. In diesem Beispiel wird das Dokument *hitchhiker.doc* im Ordner der Anwendung geöffnet.

```
// Word-Instanz erzeugen und sichtbar schalten
Word.Application word = new Word.ApplicationClass();
word.Visible = true;

// Variable für den Wert Missing.Value für nicht belegte optionale Argumente
object missing = Missing.Value;

// Dokument öffnen
object fileName = Path.Combine(Application.StartupPath, "Hitchhiker.doc");
word.Documents.Open(ref fileName, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing);
```

Listing 4: Öffnen eines Word-Dokuments mit der Übergabe optionaler Argumente

COM-Komponenten mit später Bindung verwenden

Die frühe Bindung von COM-Komponenten (siehe »COM-Komponenten mit früher Bindung verwenden«) führt manchmal zu Problemen. Das ist z. B. dann der Fall, wenn Sie eine Anwendung entwickeln, die Office-XP-Programme fernsteuert, die aber beim Kunden unter Office 97 ausgeführt werden soll. Da Microsoft zwischen Office 97 und Office 2000/XP die CLSIDs⁸ vieler Klassen geändert hat, findet die Anwendung die CLSID der XP-Klassen nicht auf dem Office-97-Rechner und meldet eine Ausnahme (siehe Abbildung 9).

Wenn Sie solche Probleme erwarten, können Sie statt der frühen Bindung auch die späte Bindung verwenden. Bei dieser Art der Bindung erzeugen Sie keine Referenz auf eine Wrapper-Assembly, sondern ermitteln die zu verwendenden Typen zur Laufzeit des Programms. Wenn Sie sicherstellen, dass Sie nur Typen und deren Methoden und Eigenschaften verwenden, die in allen relevanten Versionen der COM-Komponente verfügbar sind (und dieselbe Signatur

8. Eine CLSID (Class Id) ist ein GUID-Wert (Global Unique Identifier), der eine COM-Klasse weltweit eindeutig identifiziert. COM-Klassen sind über ihre CLSID in der Registry registriert. Die CLSID wird bei der frühen Bindung in der Wrapper-Assembly mit der Wrapper-Klasse verknüpft, sodass das Programm beim Erzeugen einer Instanz der Wrapper-Klasse an Hand der CLSID das COM-Server-Programm über die Registry ausfindig machen kann.

besitzen), kann Ihre Anwendung problemlos unter den verschiedenen Versionen der COM-Komponente ausgeführt werden. Das ist besonders dann interessant, wenn Ihre Anwendung auf mehreren Rechnern ausgeführt werden soll, die unterschiedlich ausgestattet sind.

Für die späte Bindung erzeugen Sie zunächst einen .NET-Typ für die COM-Klasse. Dazu können Sie die Methode `GetTypeFromProgID` der `Type`-Klasse verwenden. Dieser Methode übergeben Sie die Prog-Id⁹ der COM-Klasse, die üblicherweise die Form *Komponentenname.Klassenname* besitzt. Zur Erzeugung einer Word-Instanz setzen Sie z.B. die `ProgId Word.Application` ein. Alternativ können Sie auch die Methode `GetTypeFromCLSID` verwenden, wenn Sie die CLSID der COM-Klasse kennen. Dann gehen Sie allerdings wieder die Gefahr ein, dass diese auf einem anderen Rechner mit einer älteren oder neueren Version der COM-Komponente geändert ist und folglich nicht gefunden wird. Verwenden Sie also besser die `ProgId`, falls die COM-Klasse mit einer solchen registriert ist.

Wenn Sie den Typ erzeugt haben, können Sie eine Instanz des Typen über die Methode `CreateInstance` der `Activator`-Klasse erzeugen, der Sie den Typen übergeben. `CreateInstance` gibt eine Referenz auf ein `object` zurück, da dem Programm ja nicht bekannt ist, um welchen Typ es sich wirklich handelt. Listing 5 zeigt, wie das für eine Instanz von Word programmiert wird.

Das Beispiel benötigt neben der Referenz auf die Word-Wrapper-Assembly den Import der Namensräume `System`, `System.Reflection`, `System.Runtime.InteropServices` und `System.Windows.Forms`.

```
Type wordType = null;
object wordObject = null;
try
{
    // .NET-Typ für die COM-Klasse Word.Application erzeugen
    wordType = Type.GetTypeFromProgID("Word.Application");

    // Instanz dieses Typs erzeugen
    wordObject = Activator.CreateInstance(wordType);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Erzeugen der Word-Instanz: " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}
```

Listing 5: Erzeugen einer Word-Instanz über die späte Bindung

9. Die meisten COM-Klassen sind neben der CLSID auch mit einer ProgId (Program Id) in der Registry eingetragen, die ein für Menschen besser lesbares Format in der Form *Komponentenname.Klassenname* besitzt. Bei der Erzeugung eines .NET-Typen über eine ProgId ermittelt das Programm in der Registry zunächst über die ProgId die CLSID der Klasse, über die CLSID dann die Server-Anwendung und fordert diese auf, eine Instanz der Klasse zu erzeugen.

Nun kommt der etwas komplexere Teil, nämlich das Setzen von Eigenschaften und der Aufruf von Methoden. Da der COM-Typ im .NET-Programm nicht bekannt ist (und über einen `object`-Typen repräsentiert wird), können Sie Eigenschaften und Methoden nicht direkt verwenden. Dazu benutzen Sie einen Teil von Reflektion, den dynamischen Aufruf von Eigenschaften und Methoden über die Methode `InvokeMember` des Typen. Dieser Methode übergeben Sie am ersten Argument den Namen der Eigenschaft bzw. der Methode. Am zweiten Argument definieren Sie die Art der Bindung an das entsprechende Mitglied der COM-Klasse. Dazu verwenden Sie die Konstanten der `BindingFlags`-Aufzählung aus dem Namensraum `System.Reflection`. Diese Konstanten, die Sie mit `|` kombinieren können, geben an, nach welchen Mitgliedern der COM-Klasse der Reflektion-Mechanismus sucht. Die für COM wichtigsten Konstanten finden Sie in Tabelle 3.

Konstante	Bedeutung
<code>GetField</code>	gibt an, dass der Wert eines einfachen Feldes (einer Eigenschaft ohne <code>set-/get-</code> Zugriffsmethoden) gelesen werden soll
<code>GetProperty</code>	gibt an, dass der Wert einer Eigenschaft (mit <code>set-/get-</code> Zugriffsmethoden) gelesen werden soll
<code>IgnoreCase</code>	gibt an, dass die Groß- und Kleinschreibung des Namens nicht berücksichtigt werden soll
<code>Instance</code>	gibt an, dass nur Instanz-Eigenschaften und Methoden in die Suche einbezogen werden
<code>InvokeMethod</code>	gibt an, dass eine Methode aufgerufen werden soll
<code>Public</code>	gibt an, dass nur nach öffentlichen Mitgliedern mit dem angegebenen Namen gesucht werden soll
<code>SetField</code>	gibt an, dass der Wert eines einfachen Feldes (einer Eigenschaft ohne <code>set-/get-</code> Zugriffsmethoden) gesetzt werden soll
<code>SetProperty</code>	gibt an, dass der Wert einer Eigenschaft (mit <code>set-/get-</code> Zugriffsmethoden) gesetzt werden soll
<code>Static</code>	gibt an, dass nur statische Eigenschaften und Methoden in die Suche einbezogen werden

Tabelle 3: Die wichtigsten Konstanten der `BindingFlags`-Aufzählung für die späte Bindung an COM-Klassen

Am dritten Argument können Sie eine Instanz einer von der abstrakten Klasse `Binder` abgeleiteten Klasse übergeben, über das Sie spezielle Angaben zu Bindungen machen können, die nicht ohne weiteres aufgelöst werden können. Das kann z.B. dann der Fall sein, wenn die Klasse überladene Varianten einer Methode besitzt und Sie entscheiden müssen, welche dieser Varianten aufgerufen werden soll. Da diese und andere Spezialitäten bei COM-Klassen (meines Wissens nach) nicht vorkommen (bzw. nicht möglich sind), können Sie an diesem Argument in der Regel einfach `null` übergeben.

Am vierten Argument übergeben Sie die `object`-Variable, die die Referenz auf das COM-Objekt verwaltet. Das fünfte Argument übernimmt beim Setzen von Eigenschaften den Eigenschaftswert und beim Aufruf von Methoden die Argumente der Methode in Form eines `object`-Arrays.

Beim Lesen von Eigenschaften und beim Aufruf von Methoden gibt `InvokeMember` den Wert der Eigenschaft bzw. den Rückgabewert der Methode als `object`-Wert zurück.

Listing 6 setzt dieses Wissen am Beispiel der Fernsteuerung von Word ein. Das Programm setzt zunächst die `Visible`-Eigenschaft der neu erzeugten Word-Instanz auf `true` um Word sichtbar zu schalten. Danach wird ein neues Dokument erzeugt. Mit früher Bindung könnte dazu die `Add`-Methode der `Documents`-Auflistung direkt aufgerufen werden. Mit der späten Bindung ist das jedoch nicht möglich, da der Name `Documents.Add` nicht direkt aufgelöst werden kann. Deshalb ermittelt das Programm zunächst eine Referenz auf die `Documents`-Auflistung über die Abfrage der entsprechenden Eigenschaft in die Variable `documents`. Über diese Referenz ruft das Programm dann die `Add`-Methode auf. Diese Methode besitzt vier optionale Argumente, über die u.a. die zu verwendende Dokumentenvorlage spezifiziert werden kann. Da ein normales Dokument erzeugt werden soll, übergibt Listing 6 keine Argumente. Wie in COM üblich verwendet die `Add`-Methode damit die Voreinstellungen für die Argumente.

Über die `TypeText`-Methode der `Selection`-Eigenschaft schreibt das Programm dann einen Text in das Dokument. Um das Dokument auszudrucken, referenziert das Programm schließlich das aktuelle Dokument über die `ActiveDocument`-Eigenschaft der Word-Instanz und ruft deren `PrintOut`-Methode auf. Über die `Close`-Methode wird das Dokument dann geschlossen, wobei am ersten Argument `false` übergeben wird um das Dokument zu schließen ohne die Änderungen zu speichern.

```
// Visible-Eigenschaft setzen
wordType.InvokeMember("Visible", BindingFlags.Public | BindingFlags.SetProperty,
    null, wordObject, new object[] {true});

// Documents-Auflistung referenzieren ...
object documents = wordType.InvokeMember("Documents",
    System.Reflection.BindingFlags.Public | BindingFlags.Instance |
        BindingFlags.GetProperty, null, wordObject, null);

// ... und deren Add-Methode ohne Argumente aufrufen
documents.GetType().InvokeMember("Add", BindingFlags.Public | BindingFlags.Instance
    | BindingFlags.InvokeMethod, null, documents, null);

// Die Methode TypeText der Selection-Eigenschaft aufrufen
object selection = wordType.InvokeMember("Selection", BindingFlags.Public |
    BindingFlags.GetProperty, null, wordObject, null);
selection.GetType().InvokeMember("TypeText", BindingFlags.Public |
    BindingFlags.Instance | BindingFlags.InvokeMethod, null, selection,
    new object[] {"Hallo, das ist ein Text, der von außen kommt"});
```

Listing 6: Späte Bindung am Beispiel der Fernsteuerung von Word

```
// Das aktive Dokument referenzieren und ausdrucken
```

```
object document = wordType.InvokeMember("ActiveDocument", BindingFlags.Public |
    BindingFlags.GetProperty, null, wordObject, null);
document.GetType().InvokeMember("PrintOut", BindingFlags.Public |
    BindingFlags.InvokeMethod, null, document, null);
```

Listing 6: Späte Bindung am Beispiel der Fernsteuerung von Word (Forts.)

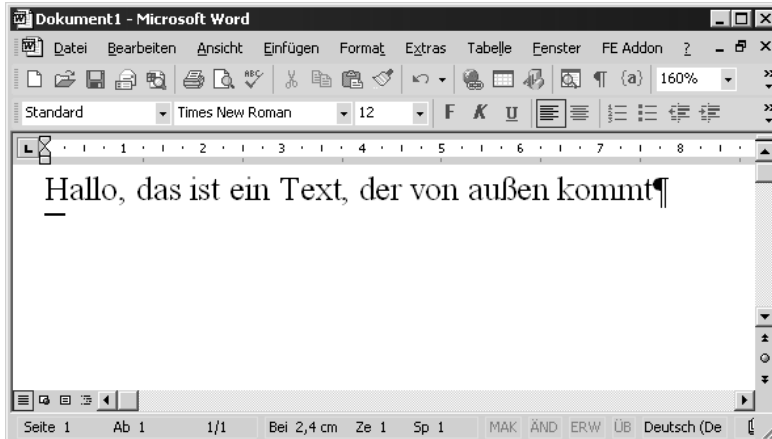


Abbildung 14: Das erzeugte Word-Dokument

Hinweis

Die Fernsteuerung von Word behandeln die Rezepte 236 und 237 noch ein wenig ausführlicher.

WMI

WMI (Windows Management Instrumentation) ist eine mächtige und interessante Microsoft-Komponente, über die Sie Informationen zu nahezu allen Objekten eines (lokalen oder entfernten) Computers oder Netzwerks erhalten und in Grenzen verändern können. Über WMI können Sie prinzipiell alle Systemobjekte, wie z.B. logische Laufwerke, Prozessoren, Benutzerkonten, Netzwerk-Domänen, Netzwerk-Clients etc. abfragen, einige Eigenschaften dieser Objekte verändern und spezielle Objekte wie z.B. Netzwerkfreigaben erzeugen. Spezielle WMI-Provider externer Hersteller erlauben darüber hinaus auch die Abfrage von Informationen von Objekten, die nicht zum Windows-System gehören.

Hinweis

Die Informationen in diesem Abschnitt sind bewusst sehr lückenhaft und beziehen sich lediglich auf die WMI-Rezepte in diesem Buch. WMI ist wesentlich komplexer, als ich es hier darstellen kann.

WMI-Installation

WMI muss auf dem Rechner, auf dem ein Programm ausgeführt wird, das WMI-Klassen einsetzt, installiert sein. Auf Windows 2000 und XP ist WMI in der Version 1.5 bereits vorinstalliert. Unter Windows 98 oder NT 4 muss WMI separat installiert werden.

ACHTUNG

Ist WMI nicht installiert, resultiert die Instanziierung einer WMI-Klasse in einer .NET-Anwendung in einer nicht abfangbaren (!) Ausnahme vom Typ `System.TypeInitializationException` mit der Meldung »Der Typinitialisierer für *Klassenname* hat eine Ausnahme verursacht«. Da fast jeder Anwender damit überlastet ist, sollten Sie auf jeden Fall überprüfen, ob WMI installiert ist. Dazu können Sie die folgende Methode verwenden (womit eigentlich schon das erste echte Rezept dieses Buches beginnt):

```
public static bool WmiInstalled()
{
    Microsoft.Win32.RegistryKey regKey =
        Microsoft.Win32.Registry.LocalMachine.OpenSubKey(
            @"Software\Microsoft\Wbem");
    return regKey != null;
}
```

Listing 7: Methode zur Überprüfung, ob WMI installiert ist

`WmiInstalled` überprüft einfach, ob der für WMI verwendete Registry-Schlüssel vorhanden ist.

Den Download der deutschen Version finden Sie unter der Adresse download.microsoft.com/download/platformsdk/wmicore/1.5/W9XNT4/DE/wmicore.EXE. Die URL der eigentlichen Download-Seite ist nach moderner Microsoft-Art sehr komplex und kann deswegen hier nicht dargestellt werden (vor allen Dingen, weil diese sich mit Sicherheit ändert). Suchen Sie idealerweise bei www.microsoft.com/downloads/search.aspx?displaylang=de nach »WMI« – über den Link WINDOWS MANAGEMENT INSTRUMENTATION (WMI) SDK 1.5 (WINDOWS 95/98/NT 4.0) erreichen Sie die Download-Seite.

WMI-Dokumentation, -Tutorial und -Tools

Die WMI-Dokumentation finden Sie in der Hilfe von Visual Studio .NET unter MSDN LIBRARY / SETUP UND SYSTEMVERWALTUNG / WINDOWS MANAGEMENT INSTRUMENTATION. Dazu müssen Sie allerdings die Plattform SDK Dokumentation installiert haben (was Sie aber natürlich auch nachholen können). Im Internet finden Sie diese Dokumentation über die WMI-Startseite msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_start_page.asp, an der Sie auch grundsätzliche Informationen zu WMI erhalten. Microsoft bietet in seinen Downloads zudem eine recht gute WMI-Einführung. Da die Download-URL sehr kompliziert ist und mit einiger Wahrscheinlichkeit zwischenzeitlich geändert wird, kann ich diese hier nicht darstellen. Suchen Sie an der Adresse www.microsoft.com/downloads/search.aspx nach »WMI Tutorial« oder »Windows Management Instrumentation Tutorial« um den Download zu finden. Wenn Sie auf dieser Suchseite dann noch nach »WMITools« oder

»WMI Administrative Tools« suchen, finden Sie den Download der *WMI Administrative Tools*, die mit dem WMI OBJECT BROWSER ein sehr hilfreiches Werkzeug zum Erforschen der WMI-Objekte eines Systems enthalten.

Einfache Abfragen über die ManagementObject-Klasse

Basis für die Verwendung von WMI ist die Klasse `System.Management.ManagementObject`, die ein Systemobjekt repräsentiert, das Sie verwalten wollen. Zur Verwendung dieser Klasse müssen Sie die Assembly *System.Management.dll* referenzieren. Im Konstruktor können Sie u.a. einen speziellen WMI-Klassenpfad übergeben. Dieser Pfad sieht in einer einfachen Version zum Zugriff auf das logische Laufwerk C: z. B. so aus:

```
"Win32_LogicalDisk.DeviceId='C:'"
```

Der erste Teil dieses Pfades bezeichnet die zu verwendende WMI-Klasse (hier `Win32_LogicalDisk`). Sofern Sie die Plattform SDK Dokumentation installiert haben, finden Sie die verfügbaren Windows-Klassen in der Hilfe von Visual Studio .NET unter MSDN LIBRARY / SETUP UND SYSTEMVERWALTUNG / WINDOWS MANAGEMENT INSTRUMENTATION / SDK-DOKUMENTATION / WINDOWS MANAGEMENT INSTRUMENTATION / WMI-REFERENCE / WIN32 CLASSES (vielleicht suchen Sie lieber einfach im Index nach dem Klassennamen oder nach »Win32_« um eine Klasse zu finden). Im Internet finden Sie diese Klassen an der Adresse msdn.microsoft.com/library/en-us/wmisdk/wmi/win32_classes.asp.

Der zweite Teil des WMI-Pfades bezeichnet eine oder mehrere Eigenschaften der anzusprechenden Klasse (hier `DeviceId`) und definiert deren Wert (hier `'C:'`). Diese Zuweisung wird verwendet, um das entsprechende WMI-Objekt zu lokalisieren. Die dazu verwendeten Eigenschaften sind innerhalb der WMI-Klasse als Schlüssel (über den Zusatz »Qualifiers: Key«) gekennzeichnet. Besitzt eine Klasse mehrere Schlüssel, geben Sie alle weiteren durch Kommata getrennt an. Der folgende Pfad bezieht sich z.B. auf das Benutzerkonto *Administrator* in der Domäne *Galaxy*. Die Klasse `Win32_Account` definiert dazu die Eigenschaften `Domain` und `Name` als Schlüssel:

```
"Win32_Account.Domain='Galaxy',Name='Administrator'"
```

Die Groß- und Kleinschreibung spielt beim WMI-Pfad übrigens keine Rolle.

WMI-Objekte sind immer in speziellen Namensräumen organisiert, deren Grundbedeutung dieselbe ist wie die der .NET Framework-Namensräume (namentliche Trennung eventuell gleich benannter Klassen mit unterschiedlicher Bedeutung). WMI-Namensräume beginnen immer mit *root* und werden dann über durch Schrägstriche getrennte Unternamen weiter spezifiziert. Der Namensraum der Standard-WMI-Objekte ist immer *root/cimv2*. Wenn Sie einen WMI-Klassenpfad ohne Namensraum angeben, bezieht sich dieser immer auf den lokalen Rechner und dessen Standard-WMI-Namensraum. Sie können den zu verwendenden Namensraum (und den anzusprechenden Rechner) aber auch angeben:

```
"\\localhost\root\cimv2:Win32_LogicalDisk.DeviceId='C:'"
```

Dieses Beispiel entspricht dem vorhergehenden, nur dass der Zielrechner (*localhost*) und der Namensraum (*root\cimv2*) hier explizit angegeben wurden. Auf diese Weise können Sie nun auch andere WMI-Namensräume und WMI-Objekte auf entfernten Systemen ansprechen.

Nachdem Sie das Objekt erzeugt haben, können Sie über den Indizierer auf die einzelnen Eigenschaften zugreifen, indem Sie deren Namen übergeben. Die Eigenschaften einer WMI-Klasse werden natürlich in der Dokumentation dieser Klasse beschrieben. Die Eigenschaften der Klasse `Win32_LogicalDisk` finden Sie z.B. an der Adresse msdn.microsoft.com/en-us/wmisdk/wmi/win32_logicaldisk.asp. Da der Indizierer lediglich einen `object`-Typ zurückgibt, müssen Sie den erhaltenen Wert normalerweise noch in den korrekten Typ konvertieren.

Etwas hinderlich (und teilweise unverständlich ...) ist dabei, dass die Eigenschaften einer WMI-Klasse zwar in der Dokumentation mit einem Datentyp angegeben sind, aber neben einem zum Datentyp passenden Wert auch einfach nichts speichern können. Der Indizierer gibt in diesem Fall `null` zurück. Sie sollten die Konvertierung also in einer Ausnahmebehandlung vornehmen.

Das folgende Listing zeigt die grundsätzliche Anwendung von WMI am Beispiel des Auslesens der wichtigsten Eigenschaften des logischen Laufwerks C:. Zum Kompilieren dieses Quellcodes müssen Sie die Namensräume `System` und `System.Management` einbinden.

```
// WMI-Objekt für das Laufwerk C: erzeugen
ManagementObject mo = new ManagementObject("Win32_LogicalDisk.DeviceId='C:'");

// Einige Eigenschaften auslesen
Console.WriteLine("Eigenschaften von C:\\");

uint availability = 0;
try
{
    availability = (uint)mo["Availability"];
}
catch {}

string description = null;
try
{
    description = (string)mo["Description"];
}
catch {}

uint mediaType = 0;
try
{
    mediaType = (uint)mo["MediaType"];
}
catch {}
```

Listing 8: Grundsätzliche Anwendung von WMI am Beispiel des Auslesens von Informationen zum logischen Laufwerk C:

```
ulong size = 0;
try
{
    size = (ulong)mo["Size"];
}
catch {}

Console.WriteLine("Verfügbarkeit: {0}", availability);
Console.WriteLine("Beschreibung: {0}", description);
Console.WriteLine("Medientyp: {0}", mediaType);
Console.WriteLine("Größe: {0}", size);
```

Listing 8: Grundsätzliche Anwendung von WMI am Beispiel des Auslesens von Informationen zum logischen Laufwerk C: (Forts.)

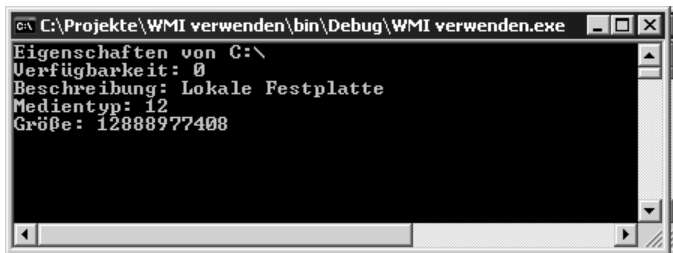


Abbildung 15: Das WMI-Beispielprogramm in Aktion

Sicherheit

WMI verwaltet Sicherheitsinformationen auf zwei Ebenen. Zum einen werden grundsätzliche WMI-Zugriffsrechte auf der Basis von Namensräumen verwaltet. Auf dieser Ebene können Benutzern und Gruppen grundsätzliche Rechte wie z.B. das Schreiben von Daten in WMI-Objekte oder der Aufruf von Methoden erteilt oder verboten werden. Benutzer, die nicht in der WMI-Benutzerliste auftauchen, können Daten erst gar nicht lesen. Per Voreinstellung besitzen Administratoren (natürlich) alle Rechte und alle anderen Benutzer Leserechte. Über die Computerverwaltung der Systemsteuerung können Administratoren die WMI-Rechte bearbeiten (SYSTEMSTEUERUNG / VERWALTUNG / COMPUTERVERWALTUNG; im ORDNER DIENSTE UND ANWENDUNGEN mit der rechten Maustaste auf WMI-STEUERUNG klicken und dort EIGENSCHAFTEN wählen; unter Windows XP müssen Sie u.U. dazu erst den Befehl NEUES FENSTER wählen, wenn EIGENSCHAFTEN nicht verfügbar ist).

Die untere Ebene der WMI-Sicherheit bestimmt der Provider. Bei den Objekten des Standard-Namensraums ist das das Windows-System. Ein Benutzer, der unter WMI das Recht besitzt, die Daten von WMI-Objekten zu lesen, muss auch auf der Systemebene über ausreichende Rechte verfügen. Problematisch dabei ist, dass Microsoft leider nicht beschreibt, welche Rechte für die Verwendung der einzelnen WMI-Klassen notwendig sind.

Wenn Sie WMI-Objekte so erzeugen, wie ich es oben gezeigt habe, wird zur Authentifizierung gegenüber dem System das Konto des Benutzers verwendet, unter dem der aktuelle Thread ausgeführt wird. In einem normalen Programm ist das das Konto des aktuell eingeloggten

Benutzers, in einem Dienst normalerweise ein speziell für den Dienst angegebenes Konto. Das kann in einem Programm problematisch werden, das z. B. Administrationsaufgaben automatisch vornehmen soll, für die der aktuelle Benutzer keine Rechte besitzt. Deshalb können Sie beim Zugriff auf *entfernte* Systeme den Namen und das Passwort eines speziellen Benutzers angeben (siehe »Entfernte Objekte«). Für *lokale* Abfragen ist das aber in WMI grundsätzlich nicht möglich. Lokale WMI-Abfragen werden immer im Kontext des Benutzerkontos ausgeführt, das für den aktuellen Thread verwendet wird. Den Grund für diese Einschränkung kann ich zwar nicht nachvollziehen, ich kann aber mit dem Rezept 211, das ein Programm mit dem Konto eines beliebigen Benutzers personifiziert, eine Lösung anbieten.

Abfragen von Auflistungen

WMI-Pfade können nicht nur ein einzelnes WMI-Objekt, sondern auch eine Auflistung von Objekten ergeben. Das ist immer dann der Fall, wenn Sie im Pfad keine Eigenschaft angeben, die ein Schlüssel ist. Wenn Sie z. B. die Klasse `Win32_LogicalDisk` abfragen und dabei keine `DeviceId` übergeben, erhalten Sie eine Auflistung von Instanzen dieser Klasse zurück, die alle logischen Laufwerke des Systems repräsentieren.

Einen solchen Pfad können Sie nicht im Konstruktor der `ManagementObject`-Klasse übergeben, da diese ja nur ein einzelnes Objekt repräsentiert. Zur Abfrage von WMI-Objekt-Auflistungen können Sie eine Instanz der Klasse `ManagementClass` einsetzen. Ein solches Objekt repräsentiert eine WMI-Klasse, deren Name Sie (u. a.) im Konstruktor übergeben können.

Die Methode `GetInstances` liefert dann eine Referenz auf ein `ManagementObjectCollection`-Objekt zurück, das eine Auflistung von `ManagementObject`-Instanzen ist.

Zur Sicherheit sollten Sie also eine `ManagementBaseObject`-Referenz zum Durchgehen verwenden. Nur wenn Sie Methoden aufrufen wollen, sollten Sie überprüfen, ob tatsächlich eine `ManagementObject`-Instanz referenziert wird, und entsprechend casten.

Hinweis

Laut der Dokumentation werden eigentlich `ManagementBaseObject`-Instanzen referenziert. Tatsächlich handelte es sich aber in meinen Programmen immer um Instanzen der von `ManagementBaseObject` abgeleiteten Klasse `ManagementObject`. Der Unterschied ist in der Praxis nur selten wichtig. Lediglich wenn Sie WMI-Methoden aufrufen wollen, können Sie dazu keine `ManagementBaseObject`-Referenz verwenden, da diese die `InvokeMethod`-Methode nicht kennt.

Wenn Sie die `ManagementObjectCollection`-Auflistung durchgehen, können Sie über den Indizierer der `ManagementBaseObject`-Objekte schließlich auf die Eigenschaften der WMI-Objekte zugreifen.

Das folgende Listing zeigt dies am Beispiel der Klasse `Win32_LogicalDisk`. Es geht alle Laufwerke des Systems durch und gibt deren Device-Id und Größe aus. Zum Kompilieren dieses Quellcodes müssen Sie die Namensräume `System` und `System.Management` importieren.

```
// ManagementClass-Instanz für die WMI-Klasse erzeugen
ManagementClass mc = new ManagementClass("Win32_LogicalDisk");

// Alle dieser Klasse angehörenden Objekte abfragen
ManagementObjectCollection moc = mc.GetInstances();

// Die einzelnen Objekte durchgehen
foreach (ManagementBaseObject mbo in moc)
{
    Console.WriteLine("{0} {1} Byte",
        mbo["DeviceId"], mbo["Size"]);
}

```

Listing 9: Abfragen mit WMI-Pfaden, die eine Auflistung von Objekten ergeben

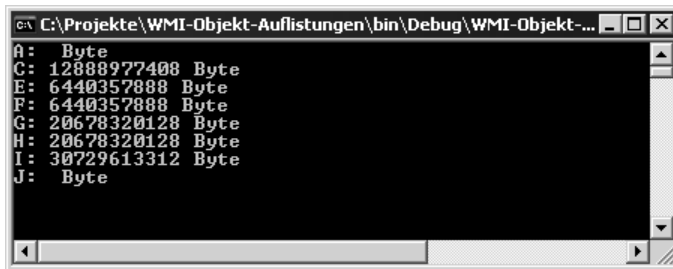


Abbildung 16: Das Programm zur Abfrage der Systemlaufwerke über WMI

Gezielt auf Informationen mit WQL zugreifen

WMI erlaubt auch Abfragen mit WQL (WMI Query Language), einer Abfragesprache, die auf SQL basiert. Dazu verwenden Sie eine Instanz der Klasse `ManagementObjectSearcher`, der Sie u. a. die Abfrage als String übergeben können. Die Abfrage verwendet die von SQL her bekannten Schlüsselwörter (`SELECT`, `FROM`, `WHERE`) und bezieht sich auf Namen von WMI-Klassen. In der Feldliste und der `WHERE`-Klausel können Sie die Namen von Eigenschaften der WMI-Klasse angeben. Die Abfrage der wichtigsten Daten der logischen Laufwerke mit dem Laufwerkstyp 3 (Festplatten-Laufwerke) sieht z. B. so aus:

```
SELECT DeviceId, Description, Size
FROM Win32_LogicalDisk WHERE DriveType = 3

```

Tip

Falls Sie SQL nicht kennen, finden Sie auf meiner Website an der Adresse www.juergen-bayer.net/artikel/SQL/SQL.html einen recht guten Artikel, den Sie auch als PDF-Datei downloaden können.

Über die `Get`-Methode des erzeugten `ManagementObjectSearcher`-Objekts erhalten Sie eine Auflistung vom Typ `ManagementObjectCollection` mit dem Ergebnis der Abfrage. Über die `Properties`-Auflistung der in dieser Auflistung referenzierten `ManagementBaseObject`-Instanzen können Sie die Eigenschaften der einzelnen WMI-Objekte abfragen.

Das folgende Listing fragt alle logischen Laufwerke des Systems ab, die in der Eigenschaft `DriveType` den Wert 3 speichern (die also Festplattenlaufwerke sind). Das Beispiel erfordert die Referenzierung der Assembly `System.Management.dll` und den Import der Namensräume `System` und `System.Management`.

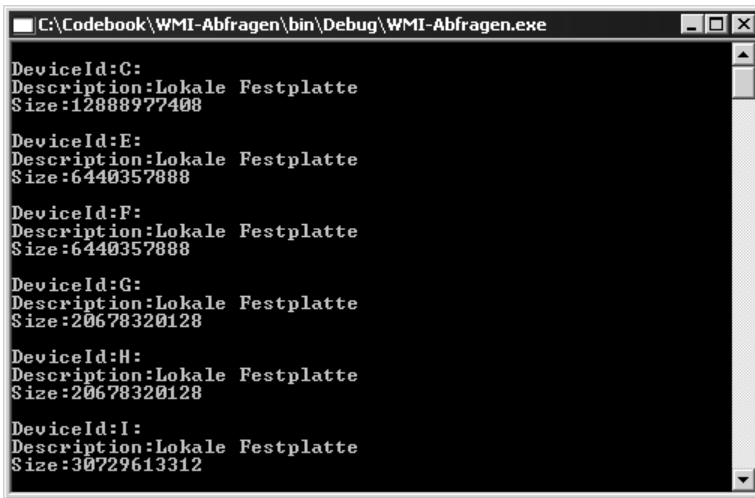


Abbildung 17: Abfragen der Festplatten-Partitionen über WMI

```

// ManagementObjectSearcher-Instanz für die Abfrage aller
// Festplatten-Laufwerke erzeugen
ManagementObjectSearcher searcher = new ManagementObjectSearcher(
    "SELECT DeviceId, Description, Size " +
    "FROM Win32_LogicalDisk WHERE DriveType = 3");

// Das Ergebnis abfragen ...
ManagementObjectCollection moc = searcher.Get();

// ... und durchgehen
foreach (ManagementBaseObject mbo in moc)
{
    Console.WriteLine();
    Console.WriteLine("DeviceId:{0}", mbo["DeviceId"]);
    Console.WriteLine("Description:{0}", mbo["Description"]);
    Console.WriteLine("Size:{0}", mbo["Size"]);
}
  
```

Listing 10: SQL-Abfragen in WMI

Entfernte Objekte abfragen

WMI erlaubt nicht nur die Abfrage lokaler Objekte. Sie können auch Objekte von entfernten (über das Netzwerk erreichbaren) Computern abfragen und manipulieren. Dazu erweitern Sie im einfachsten Fall den WMI-Klassenpfad um den Namen des Rechners und den des WMI-Namensraums:

```
"\\Trillian\\root\\cimv2:Win32_LogicalDisk.DeviceId='C:'"
```

Problematisch dabei ist allerdings, dass der aktuell in Windows eingeloggte Benutzer u.U. nicht über ausreichende Rechte zum Zugriff auf das entfernte System verfügt. Um dieses Problem zu lösen, können Sie den für die Authentifizierung zu verwendenden Benutzer auch explizit angeben. Dazu verwenden Sie eine Instanz der Klasse `ManagementScope`, der Sie im Konstruktor den Namen oder die IP-Adresse des entfernten Rechners und den anzusprechenden WMI-Namensraum (für die Windows-Systemobjekte ist das `root\\cimv2`) übergeben. Zur Authentifizierung geben Sie den Benutzernamen und das Passwort in den Eigenschaften `Username` und `Password` der `Options`-Eigenschaft an (die ein `ConnectionOptions`-Objekt referenziert).

Das `ManagementScope`-Objekt können Sie dann bei der Erzeugung eines `ManagementObject`-, `ManagementClass`- oder `ManagementObjectSearcher`-Objekts am ersten Argument angeben. Da der entsprechende Konstruktor am dritten Argument noch eine Instanz der Klasse `ObjectGetOptions` erwartet (in der Sie Optionen für das Lesen der Daten wie z.B. einen Timeout definieren können), müssen Sie zusätzlich wenigstens eine neue Instanz dieser Klasse an diesem Argument übergeben.

Das folgende Listing zeigt diese prinzipielle Vorgehensweise für das Abfragen einiger Daten des Betriebssystems des entfernten Rechners *Trillian*. Um dieses Beispiel kompilieren zu können, müssen Sie die Namensräume `System` und `System.Management` einbinden.

```
string hostName = "Trillian";
string authUser = "Ford";
string authPassword = "Handtuch";

// Verbindungsinformationen für das entfernte System und die Authentifizierung
// der Abfrage erzeugen
ManagementScope scope = new ManagementScope(@"\\" + hostName +
    @"\root\\cimv2");
scope.Options.Username = authUser;
scope.Options.Password = authPassword;

// Die Betriebssysteme des Remote-Systems abfragen
ManagementClass mc = new ManagementClass(scope,
    new ManagementPath("Win32_OperatingSystem"), new ObjectGetOptions());
ManagementObjectCollection moc = mc.GetInstances();
```

Listing 11: Abfragen entfernter Objekte in WMI

```
foreach (ManagementBaseObject mbo in moc)
{
    Console.WriteLine("Computername: {0}", mbo.Properties["CSName"].Value);
    Console.WriteLine("Betriebssystem: {0}", mbo.Properties["Caption"].Value);
    Console.WriteLine("Version: {0}", mbo.Properties["Version"].Value);
    Console.WriteLine("Freier physischer Speicher: {0}",
        mbo.Properties["FreePhysicalMemory"].Value);
}
```

Listing 11: Abfragen entfernter Objekte in WMI (Forts.)



Abbildung 18: Das Beispielprogramm zum Abfragen entfernter Objekte über WMI

Ein Tipp

Das .NET Framework-SDK enthält das Tool *MgmtClassGen*, über das Sie für alle WMI-Klassen recht umfangreiche C#-Klassen erzeugen können, die alle grundsätzlichen WMI-Möglichkeiten implementieren. Die Verwendung dieses Tools ist recht einfach. Zum Erzeugen einer C#-Klasse für die WMI-Klasse `Win32_OperatingSystem` verwenden Sie z.B. den folgenden Aufruf:

```
mgmtclassgen Win32_Product
```

WMI-Datumswerte konvertieren

WMI-Datumswerte sind leider nicht mit `DateTime`-Typen kompatibel. Das verwendete String-Format ist speziell für CIM (Common Information Model, das sprachunabhängige Modell zur Beschreibung von Unternehmen, auf dem WMI basiert) und recht kompliziert, da neben festen Platzhalterzeichen auch der Wildcard `*` erlaubt ist. Ein gültiges Datum ist z.B. »20051231*****.000000+***«.

Die Lösung des Datumsproblems liefert das Tool *MgmtClassGen*, das die Konvertierungsfunktion `ToDateTime` erzeugt. Ich habe die etwas komplizierte Implementierung dieser Funktion in Form der Methode `WMIDateToDateTime` vereinfacht. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Management` importieren.

```
public static DateTime WMIDateToDateTime(string wmiDate)
{
    int year = DateTime.Now.Year;
    int month = 1, day = 1, hour = 0, minute = 0;
    int second = 0, millisec = 0;
    string tempString = "";

    if ((wmiDate == "") || (wmiDate == null))
        return DateTime.MinValue;
    if (wmiDate.Length != 25)
        return DateTime.MinValue;
    tempString = wmiDate.Substring(0, 4);
    if (tempString != "****")
        year = Int32.Parse(tempString);
    tempString = wmiDate.Substring(4, 2);
    if (tempString != "***")
        month = Int32.Parse(tempString);
    tempString = wmiDate.Substring(6, 2);
    if (tempString != "**")
        day = Int32.Parse(tempString);
    tempString = wmiDate.Substring(8, 2);
    if (tempString != "**")
        hour = Int32.Parse(tempString);
    tempString = wmiDate.Substring(10, 2);
    if (tempString != "**")
        minute = Int32.Parse(tempString);
    tempString = wmiDate.Substring(12, 2);
    if (tempString != "**")
        second = Int32.Parse(tempString);
    tempString = wmiDate.Substring(15, 3);
    if (tempString != "***")
        millisec = Int32.Parse(tempString);

    return new DateTime(year, month, day, hour, minute,
        second, millisec);
}
```

Listing 12: Methode zum Konvertieren eines WMI-Datums in einen DateTime-Wert

Methoden aufrufen

Einige WMI-Klassen besitzen neben Eigenschaften auch Methoden. Ähnlich dem Zugriff auf Eigenschaften können Sie Methoden über die generische `ManagementObject`-Klasse nicht direkt aufrufen. Dazu verwenden Sie die Methode `InvokeMethod`. Der wichtigsten Variante dieser Methode übergeben Sie am ersten Argument den Namen der aufzurufenden WMI-Methode und am zweiten Argument ein `object`-Array mit den Argumenten. Falls die WMI-Methode eine Funktion ist, können Sie deren Rückgabe über die von `InvokeMethod` zurückgegebene `object`-Instanz auswerten.

WMI-Klassen können genau wie normale Klassen Instanz- und Klassenmethoden (statische Methoden) besitzen. Instanzmethoden rufen Sie über eine `ManagementObject`-Instanz auf, Klassenmethoden hingegen über eine Instanz der Klasse `ManagementClass`. Da der Aufruf der `InvokeMethod`-Methode in beiden Fällen gleich ist, zeige ich hier nur den Aufruf einer statischen WMI-Methode. Das Beispiel, das die Referenzierung der Assembly `System.Management.dll` und den Import der Namensräume `System` und `System.Management` erfordert, erzeugt über die WMI-Klasse `Win32_Share` eine Netzwerkfreigabe:

```
// Instanz der Klasse Win32_Share erzeugen
ManagementClass mc = new ManagementClass("Win32_Share");

// Über die Create-Methode eine Freigabe erzeugen
string share = "c:\\windows";
string name = "Windows-Ordner";
int type = 0; // Festplattenlaufwerk
object result = mc.InvokeMethod("Create", new object[] {share, name, type});
if (Convert.ToInt32(result) == 0)
    Console.WriteLine("Freigabe erfolgreich erzeugt");
else
    Console.WriteLine("Freigabe konnte nicht erzeugt werden." +
        "Möglicher Grund: Es existiert bereits eine identische Freigabe");
```

Listing 13: Aufruf einer WMI-Methode

Einige WMI-Methoden geben in By-Reference-Argumenten Werte zurück. Wenn Sie diese Methoden aufrufen wollen, übergeben Sie ebenfalls ein `object`-Array für die Argumente, deklarieren und erzeugen dieses aber vor dem Methodenaufruf. In den entsprechenden Elementen des Arrays erhalten Sie dann die Daten zurück. Das folgende Beispiel ermittelt alle Prozesse des lokalen Rechners und ruft die Methode `GetOwner` auf um den Besitzer des Prozesses zu identifizieren. `GetOwner` erwartet im Original zwei `string`-Argumente, die By-Reference übergeben werden. Am ersten wird der Name des dem Prozess zugeordneten Benutzers und am zweiten dessen Domäne zurückgegeben. Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.Management` einbinden.

```
// Die aktuellen Prozesse ermitteln
ManagementClass mc = new ManagementClass("Win32_Process");
ManagementObjectCollection moc = mc.GetInstances();
foreach (ManagementObject mo in moc)
{
    // Prozessname ausgeben
    Console.WriteLine(mo.Properties["Name"].Value.ToString());

    // Den Besitzer des Prozesses ermitteln
    object[] resultArray = new string[2];
    object result = mo.InvokeMethod("GetOwner", resultArray);
```

Listing 14: Aufruf einer WMI-Methode mit By-Reference-Argumenten

```
if (Convert.ToInt32(result) == 0)
{
    // Auswerten der zurückgegebenen Daten
    string user = resultArray[0].ToString();
    string domain = resultArray[1].ToString();
    Console.WriteLine("User: {0}", user);
    Console.WriteLine("Domäne: {0}", domain);
}
Console.WriteLine();
}
```

Listing 14: Aufruf einer WMI-Methode mit By-Reference-Argumenten (Forts.)

ADSI

ADSI (Active Directory Service Interface) ist eine (eingeschränkt¹⁰) objektorientierte Microsoft-Komponente zum Zugriff auf verschiedene Verzeichnisdienste wie Microsofts Active Directory, LDAP (Lightweight Directory Access Protocol) und NDS (Novell NetWare Directory Service). Ein Verzeichnisdienst ermöglicht den (relativ) einheitlichen Zugriff auf Ressourcen im Netzwerk, wie z.B. auf Drucker, Benutzer oder Benutzergruppen. Der Begriff *Verzeichnis* wird hier übrigens nicht im Sinne eines Dateisystem-Verzeichnisses (eines Ordners), sondern im Sinne eines Verzeichnisses von Ressourcen oder Personen, wie dies z.B. bei einem Telefon-Verzeichnis der Fall ist, verwendet. Ein Verzeichnisdienst verwaltet Ressourcen in einer hierarchischen Form. Container-Objekte enthalten dabei Objekte, die die Ressourcen bzw. Personen repräsentieren. Ein *Computer*-Objekt des WinNT-ADSI-Providers verwaltet z.B. mehrere *Group*- und *User*-Objekte (die die Benutzergruppen bzw. Benutzer repräsentieren, die auf einem Computer verwaltet werden). Über ein *Computer*-Objekt sprechen Sie einen bestimmten Computer an und können (u.a.) dessen Gruppen und Benutzer auslesen und manipulieren. Da ein *Container*-Objekt auch wieder *Container*-Objekte besitzen kann, ist eine hierarchische Aufteilung der Objekte eines Verzeichnisdienstes möglich. In einigen Verzeichnisdiensten wie dem WinNT-Dienst sind die *Container*-Objekte fest definiert. Einige Verzeichnisdienste wie z.B. Microsofts Active Directory erlauben aber auch die Erstellung eigener Container zur Verwaltung spezifischer Ressourcen oder Personen (innerhalb eines Computers oder einer Domäne).

Über ADSI können Sie die im Verzeichnisdienst enthaltenen Ressourcen relativ einfach (und vor allen Dingen einheitlich) abfragen, manipulieren, entfernen und in vielen Fällen auch neue hinzufügen.

ADSI ist älter als WMI und bietet deswegen in vielen Bereichen eine ähnliche Funktionalität. So können Sie z.B. die Daten von Benutzerkonten über ADSI und über WMI abfragen. ADSI besitzt jedoch teilweise mehr Möglichkeiten. So ist es über WMI (nach meinen Erfahrungen

10. Ein ADSI-Provider stellt für die Objekte eines Verzeichnisses zwar Klassen zur Verfügung, diese sind innerhalb von ADSI aber normalerweise nicht bekannt, da üblicherweise keine Referenz zum Provider erstellt werden kann. Zum Zugriff auf die Eigenschaften werden dann üblicherweise die Methoden *Get* und *Put* verwendet, die jedes ADSI-Objekt zur Verfügung stellt, und denen der Name der zu lesenden bzw. zu schreibenden Eigenschaft übergeben wird.

und Recherchen in Newsgroups) nicht möglich, Benutzerkonten zu erzeugen oder ein Benutzerkonto einer Gruppe hinzuzufügen, was mit ADSI kein Problem darstellt

Hinweis

ADSI ist sehr komplex, weswegen ich hier nur die für das Buch wesentlichen Grundlagen erläutern kann. Zudem ist ADSI teilweise nur schwer durchschaubar, relativ schlecht dokumentiert und liefert in einigen Fällen schwer verständliche Fehlermeldungen in Form einer Fehlernummer, deren Bedeutung Sie in der ADSI-Dokumentation nachschlagen müssen. Trotzdem ist ADSI für viele Aufgaben eine gute oder in einigen Fällen auch die einzige Lösung.

Die Installation

ADSI muss auf dem Computer, auf dem ein Programm läuft, das diese Komponente verwendet, installiert sein. Die zurzeit aktuelle Version 2.5 ist Bestandteil von Windows 2000, XP und nachfolgenden Versionen. Für Windows 98, Me und NT 4 müssen Sie ADSI separat installieren. Die Verwendung von ADSI in einem .NET-Programm führt auf einem System, auf dem ADSI nicht installiert ist, zu einer Ausnahme vom Typ `DllNotFoundException` mit der Meldung »Die DLL (activeds.dll) kann nicht geladen werden«.

Den Download finden Sie an der Adresse www.microsoft.com/ntserver/nts/downloads/other/ADSI25/default.asp.

Hinweis

Die Installation von ADSI 2.5 führte bei meinen Versuchen unter einem frisch installierten Windows 98 SE mit .NET Framework 1.0 leider nicht zum Erfolg. Ein unter Windows XP kompiliertes C#-Programm meldete leider auch nach der Installation den oben genannten Fehler. Und das, obwohl die Datei *activeds.dll* im Windows-Systemordner zu finden war. Möglicherweise handelt es sich dabei um ein Versionsproblem.

Der Provider

Die Verbindung eines Verzeichnisdienstes mit ADSI erfolgt über einen ADSI-Provider. ADSI 2.5 liefert die folgenden Provider mit:

- ▶ *WinNT*: Provider für den WinNT-Verzeichnisdienst zum Zugriff auf Windows-Ressourcen
- ▶ *LDAP*: Provider für einen LDAP-Verzeichnisdienst
- ▶ *NDS*: Provider für den Novell-Netware-Verzeichnisdienst
- ▶ *NWCOMPAT*: Provider für den Novell-Netware-3-Verzeichnisdienst

Der WinNT-Provider stellt eine Schnittstelle zu einem Windows-System dar. Über diesen Provider können Sie prinzipiell alle Verwaltungsaufgaben ausführen, die unter Windows möglich sind.

Der LDAP-Provider bietet hingegen Zugriff auf spezielle Verzeichnisdienste, die das LDAP-Protokoll implementieren. Dazu gehören z.B. die Verzeichnisdienste des Exchange Servers, des Site Servers und der Internet-Informationdienste. Alternativ zum WinNT-Provider können Sie für einige System-Aufgaben auch den LDAP-Provider für den Zugriff auf das Microsoft Active Directory verwenden. Der wesentliche Unterschied hierbei ist, dass der WinNT-Provider den Zugriff auf lokale und globale Systemressourcen, der LDAP-Provider aber nur den Zugriff auf globale Ressourcen ermöglicht. So können Sie über den LDAP-Provider z.B. nur globale (Domänen-)Benutzerkonten anlegen, über den WinNT-Provider aber auch lokale.

Die ADSI-Dokumentation

Die ADSI-Dokumentation finden Sie in der Plattform-SDK-Dokumentation unter NETWORKING AND DIRECTORY SERVICES / DIRECTORY SERVICES /ACTIVE DIRECTORY SERVICE INTERFACES. In Visual Studio .NET ist die ADSI-Dokumentation zwar vorhanden, wenn Sie die Plattform-SDK-Dokumentation installiert haben. Einen korrespondierenden Eintrag im Inhaltsverzeichnis der Hilfe von Visual Studio .NET konnte ich leider nicht finden. Suchen Sie einfach im Index der Hilfe nach »ADSI [ADSI]«. Im Internet können Sie die Dokumentation über die Adresse msdn.microsoft.com/library/en-us/netdir/adsi/active_directory_service_interfaces_adsi.asp abrufen.

ADSI-Objekte

ADSI arbeitet mit Objekten. So können Sie über den WinNT-Provider z.B. ein User-Objekt erzeugen, das ein bestimmtes Benutzerkonto repräsentiert. ADSI-Objekte sind aber im Vergleich zu normalen Objekten etwas »eigenartig«. Diese Objekte basieren auf so genannten Schemata (nicht auf Klassen). Ein Schema legt fest, welche Eigenschaften ein darauf basierendes ADSI-Objekt besitzen *kann* und welche es besitzen *muss*. Das Schema von Service-Objekten (zum Zugriff auf Windows-Dienste) definiert z.B. (u.a.) die obligatorischen Eigenschaften `StartType`, `ServiceType` und `DisplayName` und die optionalen Eigenschaften `Computer` und `ConnectTime`. Die optionalen und obligatorischen Eigenschaften eines Schemas werden in der ADSI-Dokumentation für die Windows-Provider beschrieben.

ADSI-Schemata können in Programmiersprachen, die mit ADSI-Objekten arbeiten, nicht als Typ für die Referenzierung der Objekte verwendet werden. ADSI-Objekte implementieren deshalb bestimmte Schnittstellen, über die externe Programme typischer auf Eigenschaften und Methoden der Objekte zugreifen können. Ein User-Objekt implementiert z.B. die Schnittstellen `IAds`, `IAdsUser` und `IAdsPropertyList`. `IAds`, die von jedem ADSI-Objekt implementiert wird, bietet grundsätzliche Methoden zur Arbeit mit den Objekten wie z.B. die Methoden `get` und `put` zum Lesen und Schreiben von Eigenschaften, die nicht über eine andere Schnittstelle zur Verfügung stehen. `IAdsUser` implementiert die Eigenschaften und Methoden eines Benutzers, wie z.B. `FullName`, `Description`, `SetPassword` und `ChangePassword`. `IAdsPropertyList` implementiert eine Art Auflistung, über die Sie auf die Eigenschaften des Objekts zugreifen können.

Diese Schnittstellen sind allgemein gehalten und werden für verschiedene Provider eingesetzt. Die aus diesem Grunde teilweise umfangreichen Elemente der Schnittstellen werden (was in ADSI möglich ist) teilweise nicht komplett von den Objekten der einzelnen Provider implementiert. Hier spielt auch das ADSI-Schema hinein, das ja optionale Eigenschaften erlaubt.

Das User-Objekt des WinNT-Providers implementiert z.B. eine große Anzahl der `IAdsUser`-Eigenschaften nicht. Microsoft hat dies in der Beschreibung des User-Objekts (die Sie im Internet (!) an der Adresse msdn.microsoft.com/library/en-us/netdir/adsi/winnt_user_object.asp finden) dokumentiert.

Wenn Sie auf optionale Eigenschaften eines ADSI-Objekts zugreifen, müssen Sie immer eine Ausnahmebehandlung für den Fall vorsehen, dass die Eigenschaft nicht vorhanden ist. Für Eigenschaften, die im Schema als obligatorisch definiert sind, können Sie natürlich auf die Ausnahmebehandlung verzichten.

Das Ganze und die teilweise komplexe und oft nur schwer zu findende Dokumentation macht ADSI nicht eben leicht anzuwenden. Hinzu kommen noch die teilweise komplexen Pfadangaben, mit denen Sie auf ADSI-Objekte zugreifen.

ADSI-Container

Einige ADSI-Objekte sind Container für andere Objekte. Ein `Computer`-Objekt des WinNT-Providers ist z.B. (u.a.) Container für `User`- und `Group`-Objekte. Container-Objekte implementieren die ADSI-Schnittstelle `IAdsContainer`, die über verschiedene Eigenschaften und Methoden die Arbeit mit den enthaltenen Objekten ermöglicht. Über die `Create`-Methode können Sie z.B. neue Objekte anlegen, `Delete` löscht vorhandene Child-Objekte, eine spezielle unsichtbare Eigenschaft `NewEnum` erlaubt das Durchgehen der Child-Objekte mit `foreach`, die `Filter`-Eigenschaft ermöglicht es, die Auflistung der Objekte nach einem bestimmten Schema zu filtern, sodass beim Durchgehen nur noch diese Objekte zurückgegeben werden.

ADSI-Objekte in C# verwenden

Alle ADSI-Schemata und Schnittstellen sind in C# zunächst nicht bekannt, da es sich um COM-Typen handelt. Sie können also nicht direkt auf ADSI-Objekte zugreifen. Wenn Sie in C# auf ADSI-Objekte zugreifen wollen, können Sie dies auf verschiedene Weise realisieren:

- Verwenden einer `DirectoryEntry`-Instanz aus dem Namensraum `System.DirectoryServices`, der im Konstruktor ein providerspezifischer Pfad zu einem spezifischen ADSI-Objekt übergeben wird. Bei dieser Variante benötigen Sie den genauen Pfad bzw. den genauen Namen des anzusprechenden Objekts. Ein `DirectoryEntry`-Objekt ist ein generisches Objekt, das den Zugriff auf alle ADSI-Objekte ermöglicht. Über die `Properties`-Eigenschaft können Sie auf die Eigenschaften des ADSI-Objekts zugreifen, die `Invoke`-Methode erlaubt den Aufruf von Methoden. Handelt es sich um ein Container-Objekt, können Sie über die `Children`-Auflistung auf alle Child-Objekte zugreifen,
- Verwenden einer `DirectoryEntry`-Instanz in Zusammenarbeit mit den ADSI-Schnittstellen, die Sie über die COM-Typbibliothek `activeds.tlb` referenzieren können. Über die Eigenschaft `NativeObject` einer solchen Instanz erhalten Sie eine `object`-Referenz auf das ADSI-Objekt, die Sie dann in die passende Schnittstelle casten können. Häufig ist diese Vorgehensweise sinnvoll, da Sie dann mit den Eigenschaften und Methoden der Schnittstelle direkt arbeiten können. In einigen Fällen, wenn Sie über das generische `DirectoryEntry`-Objekt einfach nicht an die gewünschte Funktionalität herankommen, ist die Verwendung der ADSI-Schnittstellen auch unumgänglich.

- ▶ Verwenden eines `DirectorySearcher`-Objekts, über das Sie ADSI-Objekte suchen können, die bestimmte Bedingungen erfüllen. Damit können Sie z. B. alle Benutzer einer Domäne ermitteln, die einen bestimmten vollen Namen besitzen,
- ▶ Verwenden des ADSI-ADO-Providers, der es ermöglicht, ADSI-Abfragen über ADO.NET auszuführen.

Ich beschreibe hier aus Platzgründen nur die erste Variante, da diese für das Buch prinzipiell ausreicht. Die zweite Variante, die die originalen ADSI-Schnittstellen benutzt, wird lediglich in einem Rezept verwendet (200, »Benutzer ermitteln, die einer Gruppe angehören«), wo sie natürlich auch erläutert wird. Ich denke, dass Sie mit dem hier erworbenen Wissen auch problemlos die weiteren ADSI-Zugriffsvarianten einsetzen können.

Für die Verwendung der Klassen des `System.DirectoryServices`-Namensraums benötigt Ihr Programm eine Referenz auf die Assembly `System.DirectoryServices.dll`.

Der ADSI-Pfad

Bei der ersten Variante hängt der im Konstruktor der `DirectoryEntry`-Instanz zu übergebende ADSI-Pfad vom Provider ab. Für den WinNT-Provider beginnt der Pfad mit »WinNT:«, für einen allgemeinen LDAP-Provider mit »LDAP:«. Spezielle LDAP-Provider wie z. B. der der Internet-Informationdienste arbeiten mit eigenen Präfixen (hier: »IIS:«).

Danach folgen providerspezifische Angaben, die zum gesuchten Objekt führen. Der WinNT-Provider erwartet den Pfad in einer der folgenden Formen:

- ▶ WinNT:
- ▶ WinNT://Domäne[,Objektschema]
- ▶ WinNT://Domäne/Server[,Objektschema]
- ▶ WinNT://Domäne/Pfad[,Objektschema]
- ▶ WinNT://Domäne/Objektname[,Objektschema]

Der Domänenname kann der Name einer Rechner-Domäne oder einfach nur der Name eines Rechners sein. Handelt es sich um eine Rechner-Domäne, gibt *Server* einen Server in dieser Domäne an.

Die bis hier genannten Pfade sprechen übergeordnete Container-Objekte (den Provider selbst, Domänen, Computer, Server) an.

Pfad ist der Pfad zu einem spezifischen Objekt innerhalb der Domäne (z. B. *PrintServer1/Printer1*). *Objektname* ist der Name eines Objekts, das direkt in einer Domäne oder auf einem Computer verwaltet wird. Optional können Sie dem Pfad noch das Schema des anzusprechenden ADSI-Objekts angeben, um sicherzustellen, dass bei eventuell gleich benannten Objekten mit unterschiedlichen Schemata das richtige referenziert wird.

Der Pfad zur Domäne *Galaxy* sieht beispielsweise so aus:

```
WinNT://Galaxy, domain
```

Der Pfad zum Computer *Zaphod* sieht folgendermaßen aus:

```
WinNT://Zaphod,computer
```

Sind Computer und Domäne angegeben, enthält der Pfad beide Angaben:

```
WinNT://Galaxy/Zaphod,computer
```

Das folgende Beispiel spricht den Benutzer *Administrator* auf dem lokalen Rechner an (der Punkt steht für den lokalen Rechner):

```
WinNT://./Administrator,user
```

Leider ist nicht dokumentiert, wie Sie auf den lokalen Computer zugreifen können, ohne dessen Name anzugeben. Wenn Sie den Computer direkt ansprechen, können Sie dazu einen Punkt verwenden:

```
WinNT://.
```

Das ist aber (bei meinen Versuchen unter Windows XP) nicht mehr möglich, wenn Sie einen Objektnamen angeben:

```
WinNT://./Administrator,user
```

In diesem Beispiel führt der Zugriff zum Fehler »Der angegebene Netzwerkpfad wurde von keinem Netzwerkdienstanbieter angenommen«. Sie können aber einfach den Namen des lokalen Rechners auslesen und den Pfad damit zusammensetzen:

```
string adsiPath = "WinNT://" + Environment.MachineName + "/Administrator,user";
```

Der Pfad eines normalen LDAP-Providers ist ungleich komplexer. In den Rezepten des Buchs setze ich diesen Provider nicht ein, weil die Problemlösungen, die sich lediglich auf lokale oder entfernte Rechner oder die Domäne beziehen, den WinNT-Provider erfordern. Deshalb beschreibe ich den Aufbau des LDAP-Pfades nur grundlegend und vereinfachend.

Den LDAP-Provider verwenden Sie z.B. zum Zugriff auf die Verzeichnisdienste des Exchange-Servers, des Site-Servers, der Internet-Informationsdienste und des Microsoft Active Directory. Active Directory kann auf einem Domänen-Controller (DC) konfiguriert werden und Ressourcen verzeichnen, die in der Domäne verwaltet werden. Über eine Abfrage des Active Directory können diese Ressourcen zentral lokalisiert werden. Eine recht

gute und einfache Einführung zu Active Directory finden Sie in der ADSI-Dokumentation unter NETWORKING AND DIRECTORY SERVICES / DIRECTORY SERVICES / ACTIVE DIRECTORY SERVICE INTERFACES / ACTIVE DIRECTORY SERVICE INTERFACES QUICK START TUTORIALS / ACCESSING ACTIVE DIRECTORY USING VISUAL BASIC.

Ein LDAP-Pfad besitzt normalerweise das folgende Format:

```
LDAP://[Hostname[:Portnummer]][Distinguished Name]
```

Der Hostname (der vorwiegend beim Microsoft Active Directory-Verzeichnisdienst verwendet wird) bezeichnet einen speziellen Server, dessen LDAP-Verzeichnisdienst angesprochen werden soll. Lassen Sie diesen weg, sprechen Sie einen der erreichbaren Server an. ADSI verwendet dabei den Server, der die Anfrage am besten verarbeiten kann.

Der »Distinguished Name« (DN; Ausgezeichneter Name), der auf dem RFC¹¹ 1779 basiert (www.ietf.org/rfc/rfc1779.txt), ist das eigentliche Problem. Der Pfad zu einem Objekt wird hierbei aus verschiedenen, hierarchisch angeordneten Angaben zusammengesetzt, die mit einem Token beginnen, das eine Aussage über die Hierarchieebene macht. Die folgenden Token werden in einem normalen LDAP-DN verwendet:

- ▶ CN: Der Name des Objekts (Common Name = Allgemeiner Name),
- ▶ O: Name der Organisation,
- ▶ OU: Name der Einheit innerhalb der Organisation (Organizational Unit),
- ▶ C: Name des Landes (Country).

Der Microsoft LDAP-Provider erweitert die Token noch um das Token *DC* (Domain Component), das für Teile von DNS-Domännennamen wie *MyDomain.MyCompany.com* verwendet wird.

Nun müssen Sie nur noch wissen, welche Bedeutung die einzelnen Token besitzen. Dazu ist es wichtig zu wissen, wie ein LDAP-Verzeichnisdienst Objekte verwaltet.

In einem LDAP-Verzeichnisdienst werden Objekte immer hierarchisch organisiert. In einem normalen LDAP-Verzeichnisdienst verwaltet häufig eine übergeordnete *Organization* (O) eine oder mehrere *Organizational Units* (Organisationseinheiten, OU). Organizational Units können direkt Objekte verwalten oder auch wieder Organizational Units. Die in einer Organizational Unit verwalteten Objekte besitzen einen *Common Name* (CN), über den sie angesprochen werden können. Da ein LDAP-Objekt auch Container für andere Objekte sein kann, kann es sein, dass einem Objekt noch weitere Objekte untergeordnet sind.

11. Ein RFC (Request For Comment = Aufforderung zum Kommentar) ist eine Empfehlung für einen Standard im Internet

Die Unterscheidung zwischen einer OU und einem Container-Objekt ist an dieser Stelle vielleicht unklar. OUs dienen lediglich der hierarchischen Grundeinteilung. Objekte können jedoch auch Eigenschaften und Methoden besitzen. Ein Group-Objekt kann z.B. User-Objekte verwalten und Methoden zum Hinzufügen oder Löschen von Benutzern anbieten und in Eigenschaften den vollen Namen der Gruppe und Zugriffsrechte für Systemressourcen verwalten.

Ein Beispiel für einen solchen hierarchischen Aufbau ist die Verwaltung der Ressourcen einer Firma mit mehreren Filialen. Die Organization steht für die Firma, die erste Ebene der Organizational Units sind die Regionen (Nord, Süd, Ost, West). Denen sind wieder Organizational Units zugeordnet, die die Filialen einer Region verwalten. Innerhalb einer Filiale sind u.a. Group-Objekte gespeichert, die bestimmte Eigenschaften (wie z.B. Zugriffsrechte) besitzen. Ein Group-Objekt ist ein Container für Employee-Objekte, die die einer Gruppe zugewiesenen Mitarbeiter verwalten. Der LDAP-Pfad zum Mitarbeiter »Ford« in der Gruppe »Reiseleiter« der OU »NRW« in der OU »West« der Organization »Galaxy-Travel« sieht dann z. B. so aus:

```
LDAP://CN=Ford,CN=Reiseleiter,OU=NRW,OU=West,O=Galaxy-Travel
```

Active Directory verwendet für die Verwaltung von Domänenressourcen keine Organization-Einteilung, sondern den Domännennamen. Diesen geben Sie dann über das DC-Token in seinen Teilen an. Angenommen, das oben angegebene Szenario wird statt in der Organization Galaxy-Travel in der Domäne Galaxy-Travel.de verwaltet, sieht der ADSI-Pfad so aus:

```
LDAP://CN=Ford,CN=Reiseleiter,OU=NRW,OU=West,DC=Galaxy-Travel,DC=de
```

Da mehrere Domänencontroller Active Directory-Verzeichnisdienste anbieten können (und dabei sogar in der Lage sind sich gegenseitig auszutauschen), können Sie den Server angeben, den Sie gezielt abfragen wollen:

```
LDAP://DomainController1/CN=Ford,CN=Reiseleiter,OU=NRW,OU=West,DC=Galaxy-Travel,DC=de
```

Geben Sie den Domänencontroller nicht an, sucht ADSI automatisch den Verzeichnisdienst, der am besten zur Anfrage passt.

Einige spezielle LDAP-Verzeichnisdienste werden über einen bestimmten Port angesprochen und verzichten auf Organization-Zuordnungen. Das ist z.B. beim Membership-Server des Microsoft Site Server der Fall (der üblicherweise Benutzerkonten für eine Website verwaltet, deren Zugang nicht für alle Benutzer möglich sein soll). Der Pfad zum Member mit dem Common Name Ford auf dem Membership-Server *Zaphod*, der auf dem Port 1500 lauscht, sieht dann z.B. so aus:

```
LDAP://Zaphod:1500/ou=Members/cn=Ford
```

Falls Sie einen LDAP-Verzeichnisdienst ansprechen wollen, kommen Sie aber wohl kaum um das intensive Studium der Dokumentation herum, um den korrekten LDAP-Pfad für den Zugriff auf Ressourcen zu ermitteln.

Erzeugen der DirectoryEntry-Instanz

Wenn Sie den Pfad zum anzusprechenden Objekt zusammengestellt haben, können Sie damit eine Instanz der Klasse `DirectoryEntry` aus dem Namensraum `System.DirectoryServices` erzeugen. Den Pfad übergeben Sie am ersten Argument des Konstruktors.

Der Zugriff auf die Ressourcen eines Verzeichnisdienstes erfordert natürlich bestimmte Rechte. Die dazu verwendeten Benutzerkonten können im Verzeichnisdienst selbst definiert sein. Greift der Verzeichnisdienst auf Ressourcen eines Computers zu, werden zudem (bzw. ausschließlich) die Sicherheitseinschränkungen des Betriebssystems verwendet. Wenn Sie eine `DirectoryEntry`-Instanz nur mit einem Pfad erzeugen, verwendet ADSI zur Authentifizierung gegenüber dem Verzeichnisdienst bzw. dem System das Benutzerkonto, das dem aktuellen Thread zugeordnet ist (also normalerweise das Konto des in Windows eingeloggten Benutzers). Da dieses häufig nicht über ausreichende Rechte verfügt (besonders, wenn Sie über ADSI auf ein entferntes System zugreifen), können Sie am zweiten und dritten Argument des Konstruktors den Namen und das Passwort eines Benutzerkontos übergeben, das dann explizit für die Authentifizierung verwendet wird. In einer weiteren Variante des Konstruktors können Sie die Art der Authentifizierung einstellen, aber das führt hier zu weit.

Die Instanz sollten Sie in einer Ausnahmebehandlung erzeugen, um auf ADSI-Fehler reagieren zu können. Diese Fehler werden aber leider erst gemeldet, wenn Sie versuchen auf das ADSI-Objekt zuzugreifen. Wenn Sie z.B. einen ungültigen Pfad übergeben, führt das erst beim Lesen oder Schreiben einer Eigenschaft oder beim Aufruf einer Methode zu einer Ausnahme. Deshalb sollten Sie im `try`-Block eine Eigenschaft des Objekts lesen, damit Fehler direkt bei der Erzeugung erkannt werden. Ich verwende einfach die Eigenschaft `Name` des `DirectoryEntry`-Objekts, die den Common Name des ADSI-Objekts verwaltet. Das Beispiel setzt die Variable `de` im Fehlerfall auf `null`, damit dieser Zustand bei der späteren Verwendung des Objekts abgefragt werden kann.

Zum Kompilieren dieses Quellcodes müssen Sie die Namensräume `System` und `System.DirectoryServices` einbinden. Außerdem müssen Sie in den Variablen `authUser` und `authPassword` die Daten eines Benutzers eintragen, der auf Ihrem System existiert.

```
// DirectoryEntry-Objekt für den Zugriff auf den Benutzer "Administrator" auf
// dem lokalen Computer erzeugen
DirectoryEntry de = null;
string adsiPath = "WinNT://" + Environment.MachineName +
    "/Administrator,user";
string authUser = "Ford";
string authPassword = "Handtuch";

try
{
```

Listing 15: Erzeugen einer `DirectoryEntry`-Instanz für ein ADSI-Objekt

```
// DirectoryEntry-Objekt für das ADSI-Objekt erzeugen
de = new DirectoryEntry(adsiPath, authUser, authPassword);

// Einmal auf das Objekt zugreifen, um dessen Existenz zu überprüfen
string name = de.Name;
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    de = null;
}
```

Listing 15: Erzeugen einer DirectoryEntry-Instanz für ein ADSI-Objekt (Forts.)

Fehlerauswertung

Die Fehlerauswertung ist unter ADSI leider nicht allzu einfach (was auch meine Erfahrungen mit LDAP unter dem Site Server bestätigen haben). Da ADSI über COM angesprochen wird, führen ADSI-Fehler zu einer Ausnahme vom Typ `System.Runtime.InteropServices.COMException`. Die Fehlermeldung ist, je nach Provider, in einigen Fällen aussagekräftig, in anderen aber leider nicht. Wenn Sie z.B. beim WinNT-Provider beim Zugriff auf ein User-Objekt den Namen eines nicht vorhandenen Benutzers angeben, erzeugt dieser die Fehlermeldung »Der Benutzername konnte nicht gefunden werden«. Geben Sie aber einen falschen ADSI-Pfad an (z.B. »WinNT://« + `Environment.MachineName` + »/Administrator,uXer«), resultiert dies in einer Ausnahme mit der Meldung »Unbekannter Fehler (0x80005000)«. Der in dieser Meldung angegebene `HRESULT`-Fehlercode (vgl. »Aufruf von API-Funktionen über `PInvoke`« im Abschnitt »Das Windows-API«) wird in der ADSI-Dokumentation unter `NETWORKING AND DIRECTORY SERVICES / DIRECTORY SERVICES / ACTIVE DIRECTORY SERVICE INTERFACES / ACTIVE DIRECTORY SERVICE REFERENCE / ADSI ERROR CODES` beschrieben. Sie können aber im Index auch einfach nach »ADSI [ADSI]« suchen, dann auf `ADSI REFERENCE` und schließlich auf `ERROR CODES` klicken um die Beschreibung der Fehler zu erhalten. Im Internet finden Sie diese an der Adresse msdn.microsoft.com/library/en-us/netdir/adsi/adsi_error_codes.asp.

Der Fehler `0x80005000` sagt demnach aus, dass ein ungültiger Pfadname übergeben wurde. Ich habe mich schon immer gefragt, warum ADSI die Fehlercodes nicht automatisch in die passende Fehlermeldung umsetzt. Bisher habe ich auch noch keine Möglichkeit gefunden, die `HRESULT`-Werte über das Windows-API oder ADSI in die passende Fehlermeldung umzuwandeln. Schade.

Lesen von Eigenschaften

Wenn Sie das ADSI-Objekt erfolgreich instanziiert haben, können Sie dessen Eigenschaften lesen. Lediglich eine Eigenschaft, nämlich `Name` (die für den Common Name steht), können Sie direkt auslesen. Auf alle anderen können Sie über die `Properties`-Auflistung zugreifen. Dem Indizierer dieser Auflistung übergeben Sie den Namen der Eigenschaft.

`Properties` gibt eine Referenz auf ein `PropertyValueCollection`-Objekt zurück. Dass es sich dabei wieder um eine Aufzählung handelt, ist darin begründet, dass ADSI auch mehrwertige Eigenschaften erlaubt (was allerdings selten genutzt wird). Wenn Sie eine solche abfragen, können Sie über die Eigenschaft `Count` die Anzahl der Teile der Eigenschaft abfragen und über

den Indizierer des `PropertyValueCollection`-Objekts auf diese Teile zugreifen, indem Sie den Index übergeben. Den Wert einwertiger Eigenschaften können Sie aber auch einfach über die Eigenschaft `Value` auslesen. `Value` besitzt (natürlich) den Typ `object`, den Sie in der Regel vor der Verarbeitung konvertieren müssen.

Beim Lesen müssen Sie außerdem beachten, dass einige Eigenschaften in ADSI-Objekten optional sind und deshalb nicht zwangsläufig implementiert werden müssen. Welche das sind, können Sie in der Dokumentation des ADSI-Schemas nachlesen. Diese Eigenschaften sollten Sie immer in einer Ausnahmebehandlung lesen, die die eventuell auftretende Ausnahme einfach verwirft.

Das folgende Beispiel, das das Beispiel in Listing 15 weiterführt, liest auf diese Weise die ADSI-Eigenschaften `Name`, `FullName` und `Description` des erzeugten User-Objekts aus. Da die beiden letztgenannten Eigenschaften im User-Schema (wie alle Eigenschaften außer `Name`) optional sind, erfolgt das Lesen in einer Ausnahmebehandlung:

```
if (de != null)
{
    Console.WriteLine(de.Name);

    string fullName = null;
    try {fullName = de.Properties["FullName"].Value.ToString();}
    catch {}
    Console.WriteLine("Voller Name: {0}", fullName);

    string description = null;
    try {description = de.Properties["Description"].Value.ToString();}
    catch {}
    Console.WriteLine("Beschreibung: {0}", description);
}
```

Listing 16: Lesen von Eigenschaften eines ADSI-Objekts



Abbildung 19: Das Programm hat Eigenschaften des Benutzers ausgelesen

Schreiben von Eigenschaften

ADSI ermöglicht nicht nur das Lesen, sondern auch das Schreiben von Eigenschaften (sofern das Benutzerkonto über ausreichende Rechte verfügt). Um Eigenschaften mit neuen Werten zu beschreiben, greifen Sie über die `Properties`-Auflistung auf die Eigenschaft zu, dürfen dann aber nicht einfach in die `Value`-Eigenschaft schreiben. Wenn Sie direkt in `Value` schreiben, erhalten Sie den »sehr aussagekräftigen« Fehler »Unbekannter Fehler«. Der Grund dafür (dass dieser Fehler auftritt, nicht dass die Meldung so nutzlos ist) liegt darin, dass ADSI-Objekte wie bereits gesagt auch mehrwertige Eigenschaften ermöglichen und ein `Directory-`

Entry-Objekt scheinbar davon ausgeht, dass eine Eigenschaft immer mehrwertig ist. Sie müssen auch bei einwertigen Eigenschaften die `Add`-Methode aufrufen. Bei einwertigen Eigenschaften führt `Add` aber nicht dazu, dass der übergebene Wert hinzugefügt wird, sondern dass der vorhandene durch den neuen ersetzt wird. Für mehrwertige Eigenschaften macht das ja Sinn, für einwertige hätte Microsoft sich da aber eine andere Lösung einfallen lassen können. Die Werte von mehrwertigen Eigenschaften können Sie übrigens über die `Clear`-Methode vor dem Neu-Schreiben löschen.

Das Schreiben alleine reicht aber nicht aus. ADSI-Objekte werden (wahrscheinlich aus Performancegründen) immer im Speicher gecacht. Wenn Sie Eigenschaften lediglich beschreiben und das Programm dann beenden, gehen Ihre Änderungen verloren. Nach dem Schreiben müssen Sie immer die `CommitChanges`-Methode des `DirectoryEntry`-Objekts aufrufen, damit die Änderungen in den Verzeichnisdienst zurückgeschrieben werden. Das Ganze sollte natürlich wieder in einer Ausnahmebehandlung erfolgen.

Das folgende Beispiel, das das Beispiel aus Listing 15 weiterführt, ändert den vollen Namen des User-Objekts:

```
try
{
    // Die Eigenschaft FullName beschreiben
    de.Properties["FullName"].Add("Zaphod Beeblebrox");

    // Die gecachten Werte in den Verzeichnisdienst schreiben
    de.CommitChanges();
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Schreiben der Eigenschaft: {0}",
        ex.Message);
}
```

Listing 17: Schreiben einer Eigenschaft in einem ADSI-Objekt

Freigeben der Ressourcen

Nach der Arbeit mit ADSI-Objekten sollten Sie die Ressourcen, die diese Objekte verwenden, explizit freigeben. Sie können sich zwar auch auf den Garbage Collector verlassen, der automatisch den Destruktor der `DirectoryEntry`-Objekte aufruft und die Ressourcen freigibt. Bis zur Ausführung des GC kann es aber einige Zeit dauern. Um sicherzugehen, rufen Sie einfach die `Close`-Methode des `DirectoryEntry`-Objekts auf:

```
// Schließen des ADSI-Objekts und Freigeben der ADSI-Ressourcen
de.Close();
```

Durchgehen von Container-Objekten

ADSI-Objekte können auch Container für andere ADSI-Objekte sein. Ein `Computer`-Objekt des WinNT-Providers ist z.B. (u.a.) ein Container für `Group`- und `User`-Objekte. Die in einem Container enthaltenen Objekte können Sie über die Eigenschaft `Children` durchgehen, die eine einfache Auflistung von `DirectoryEntry`-ist. Die einzelnen Objekte können Sie wieder genauso behandeln, wie ich es bereits gezeigt habe. Das Beispiel in Listing 18 gibt zusätzlich zum Namen des Objekts auch dessen Schemaklassen-Name aus.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.DirectoryServices` einbinden und in den Variablen `authUser` und `authPassword` die Daten eines Benutzers eintragen, der auf Ihrem System existiert.

```
// DirectoryEntry-Objekt für den Zugriff auf den lokalen Computer erzeugen
DirectoryEntry de = null;
string adsiPath = "WinNT://" + Environment.MachineName + ",computer";
string authUser = "Ford";
string authPassword = "Handtuch";

try
{
    // DirectoryEntry-Objekt für das ADSI-Objekt erzeugen
    de = new DirectoryEntry(adsiPath, authUser, authPassword);

    // Einmal auf das Objekt zugreifen, um dessen Existenz zu überprüfen
    string name = de.Name;
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    de = null;
}

// Eine Eigenschaft auslesen und die enthaltenen Objekte durchgehen
if (de != null)
{
    Console.WriteLine("Computername: {0}", de.Name);

    // Die Child-Objekte durchgehen
    foreach (DirectoryEntry child in de.Children)
    {
        // Name ausgeben
        Console.WriteLine("Name: {0}", child.Name);
        Console.WriteLine("Schema: {0}", child.SchemaClassName);
        Console.WriteLine();
    }
}
```

Listing 18: Einfaches Durchgehen eines Container-Objekts



Abbildung 20: Das Programm hat alle dem Computer untergeordneten ADSI-Objekte ausgelesen (leicht modifizierte Darstellung)

Die können jedoch auch nur die Objekte durchgehen, die einer oder mehreren Schema-Klassen angehören. Dazu fügen Sie der `SchemaFilter`-Eigenschaft der `Children`-Auflistung über deren `Add`-Methode die gewünschten Schemaklassennamen hinzu. Falls Sie zuvor bereits gefiltert haben, sollten Sie vor dem Aufruf von `Add` die `Clear`-Methode aufrufen, um alle vorhandenen Filter zu löschen. Das folgende Beispiel geht alle `User`-Objekte des Computers durch:

```
Console.WriteLine("Alle User:");

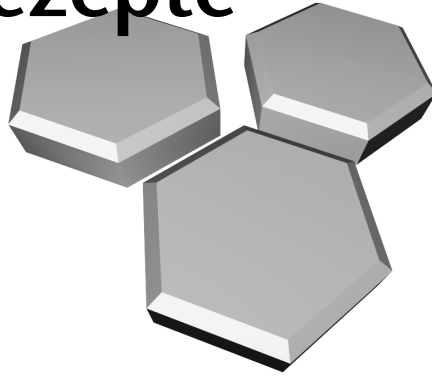
if (de != null)
{
    // Filter definieren
    de.Children.SchemaFilter.Clear();
    de.Children.SchemaFilter.Add("User");

    // Die Child-Objekte durchgehen
    foreach (DirectoryEntry child in de.Children)
    {
        // Name ausgeben
        Console.WriteLine(child.Name);
        Console.WriteLine();
    }
}
```

Listing 19: Durchgehen nur bestimmter ADSI-Objekte in einem Container

TEIL II

Rezepte



Basics

1 (Ganz-)Zahlen hexadezimal darstellen

Das hexadezimale Zahlenformat bietet besonders für Programmierer den wichtigen Vorteil, dass ein Byte mit zwei Ziffern dargestellt werden kann (00 bis FF). Es eignet sich deshalb hervorragend zur optischen Auswertung der einzelnen Bytes einer Zahl. Das ist wohl auch der Hauptgrund dafür, dass Technologien wie HTML für Farbangaben neben vordefinierten Konstanten auch hexadezimal dargestellte Zahlwerte erlauben, die sich aus drei Bytes zusammensetzen. Das linke Byte steht dabei für den Rotanteil der Farbe, das mittlere für den Grünanteil und das rechte für den Blauanteil. Die Farbe FFA000 steht z. B. für einen Orangeton.

Eine Ganzzahl können Sie sehr einfach entweder über die `ToString`-Methode eines Integer-Datentyps oder über die `Format`-Methode der `String`-Klasse in einen hexadezimalen Wert konvertieren, indem Sie als Format-Bezeichner ein `X` übergeben:

```
int i = 0xFAFBFC;
string hexNumber = String.Format("{0:X}", i);
```

Listing 1: Formatieren einer Ganzzahl als Hexadezimalwert

2 Hexadezimale Zahlen in dezimale Zahlen konvertieren

Hexadezimal dargestellte Zahlwerte können Sie über die Methoden `ToByte`, `ToSByte`, `ToInt16`, `ToUInt16`, `ToInt32`, `ToUInt32`, `ToInt64` und `ToUInt64` der `Convert`-Klasse in eine Integer-Zahl konvertieren. Am ersten Argument übergeben Sie dazu den zu konvertierenden String, am zweiten die Basis der Zahl, die in diesem Fall 16 ist. Da die Konvertierung auch fehlschlagen kann, sollten Sie die dann erzeugte Ausnahme abfangen:

```
string hexNumber = "FFA033";

// Konvertieren des Hex-Werts in eine long-Zahl
long number = 0;
try
{
    number = Convert.ToInt64(hexNumber, 16);
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Konvertieren: {0}", ex.Message);
}
```

Listing 2: Konvertieren eines hexadezimalen Werts in einen long-Wert

Basics

Datum
Zeit

Anwen-
dunge

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

Alternativ zur Ausnahmebehandlung können Sie auch vor der Konvertierung abfragen, ob der zu konvertierende String eine gültige Zahl speichert. Das Rezept 10 nimmt Ihnen diese Aufgabe ab.

3 Ganzzahlen dual darstellen

In einigen Fällen ist es hilfreich oder einfach nur informativ, die duale Repräsentation einer Zahl vor Augen zu haben. Sinn macht das allerdings nur bei ganzen Zahlen, da Fließkommazahlen auf eine recht komplexe Weise im Speicher abgelegt werden (als Vorzeichenbit, Mantisse und Exponent).

Das .NET Framework bietet dazu keine Möglichkeit. Sie müssen die Formatierung selbst programmieren. Dazu können Sie ganz einfach in einer Schleife die einzelnen Bits der Zahl durchgehen. Wenn Sie auf die Zahl ein bitweises Und mit 1 anwenden, können Sie den Zustand des jeweils rechten Bits ermitteln. In der Schleife shiften Sie die Zahl dann um ein Bit nach rechts, damit beim nächsten Durchlauf das nächste Bit verglichen wird. Das jeweilige Ergebnis hängen Sie einfach an eine String-Variable an.

Für eine `ulong`-Zahl sieht das Ganze dann so aus wie in der Methode `DualNumber` in Listing 3. Die Methode gibt das Ergebnis mit einem vorangestellten `+` zurück, da die Zahl nur positiv sein kann.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` importieren.

```
public static string DualNumber(ulong number)
{
    string result = "";

    // Die Bits 0 bis 63 durchgehen
    for (int i = 0; i < 64; i++)
    {
        // Über ein bitweises Or ermitteln, ob das Bit gesetzt ist, und das
        // Ergebnis vorne an die Stringvariable anfügen
        result = (number & 1).ToString() + result;

        // Die Bits um eine Position nach rechts shiften
        number >>= 1;
    }

    return "+" + result;
}
```

Listing 3: Methode zur Konvertierung einer `ulong`-Zahl in eine Dualzahl

Bei Zahlen mit Vorzeichen ist die Umwandlung in eine Dualzahl prinzipiell nicht allzu einfach. Bei diesen Zahlen wird das linke Bit verwendet, um das Vorzeichen zu bestimmen. Ein 0-Bit kennzeichnet eine positive, ein 1-Bit eine negative Zahl. Problematisch ist die Auswertung der einzelnen Bits, da ein Rechtsshift das Vorzeichenbit nicht mitshiftet und ein Rechtsshift von -1 immer -1 ergibt (wahrscheinlich, weil es -0 ja nicht gibt).

Einen `long`-Wert können Sie auch nicht einfach nach `ulong` casten, da bei negativen Werten Werte herauskommen, die dazu führen, dass die linken Bits, die eigentlich 0 sind, als 1 ausgewertet werden (was ich nicht erklären kann, aber ausprobiert habe).

Meine Idee dazu war nun, einfach über `Math.Abs` den Absolutwert des `long`-Werts zu berechnen, wenn dieser negativ ist. Leider führte dies bei negativen Zahlen auch zu dem Problem, dass linke 0-Bits als 1 ausgewertet wurden. Ein Versuch mit einer einfachen Negation der Zahl brachte dann aber den Erfolg.

Die überladene Variante der Methode `DualNumber` in Listing 4 führt die Umwandlung für einen `long`-Wert aus. Ist der Wert negativ, merkt sich die Methode diesen Zustand und negiert die Zahl. Der so ermittelte positive Wert wird dann wie in der `ulong`-Variante Bit für Bit ausgewertet. Der ermittelte Wert wird je nach dem Vorzeichen der übergebenen Zahl mit einem `-` oder `+` als Präfix zurückgegeben.

```
public static string DualNumber(long number)
{
    string result = "";

    // Ermitteln, ob die Zahl negativ ist, und in diesem Fall den Absolutwert
    // berechnen
    bool negative = false;
    if (number < 0)
    {
        negative = true;
        number *= -1;
    }

    // Die Bits 0 bis 64 durchgehen
    for (int i = 0; i < 64; i++)
    {
        // Über ein bitweises Or ermitteln, ob das Bit gesetzt ist, und das
        // Ergebnis vorne an die Stringvariable anfügen
        result = (number & 1).ToString() + result;

        // Die Bits um eine Position nach rechts shiften
        number >>= 1;
    }

    // Negative Zahlen auswerten
    if (negative)
        result = "-" + result;
    else
        result = "+" + result;

    return result;
}
```

Listing 4: Methode zur Konvertierung einer `long`-Zahl in eine Dualzahl

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die Konvertierung funktioniert sogar für den minimalen `long`-Wert (-9.223.372.036.854.775.808), obwohl `long` nur maximal einen positiven Wert von 9.223.372.036.854.775.807 aufnehmen kann. Die Negation des minimalen Werts führt aber zu einem Überlauf, woraus wieder der minimale Wert resultiert.

Nun fehlen nur noch Methoden zur Konvertierung der anderen Integer-Datentypen. Um die Umwandlung nicht immer neu schreiben zu müssen, habe ich diese so entwickelt, dass sie die `long`- oder `ulong`-Variante aufrufen, um den Wert zu konvertieren, und aus dem Ergebnis den rechten Teilstring extrahieren, der der Bit-Größe des jeweiligen Typs entspricht:

```
public static string DualNumber(int number)
{
    return number < 0 ?
        "-" + DualNumber((long)number).Substring(33, 32) :
        "+" + DualNumber((long)number).Substring(33, 32);
}

public static string DualNumber(uint number)
{
    return "+" + DualNumber((ulong)number).Substring(33, 32);
}

public static string DualNumber(short number)
{
    return number < 0 ?
        "-" + DualNumber((long)number).Substring(49, 16) :
        "+" + DualNumber((long)number).Substring(49, 16);
}

public static string DualNumber(ushort number)
{
    return "+" + DualNumber((ulong)number).Substring(49, 16);
}

public static string DualNumber(byte number)
{
    return "+" + DualNumber((ulong)number).Substring(57, 8);
}
```

Listing 5: Methoden zur Konvertierung von `int`, `uint`, `short`, `ushort` und `byte`-Werten in eine Dualzahl

Listing 6 zeigt eine Anwendung, die die Konvertierungs-Methoden für einige `short`- und `ushort`-Zahlen testet.

```
Console.WriteLine("short-Werte als Dualzahl:");

// Zahlen vom -5 bis +2 als Dualzahl darstellen
```

Listing 6: Test der Methoden zur Konvertierung von Integerwerten in eine Dualzahl

```

for (short i = -5; i < 5; i++)
    Console.WriteLine("{0:000}: {1}", i, DualNumber(i));

// Kleinste short-Zahl + 1 darstellen
Console.WriteLine("{0:000}: {1}", short.MinValue + 1,
    DualNumber((short)(short.MinValue + 1)));

// Kleinste short-Zahl darstellen
Console.WriteLine("{0:000}: {1}", short.MinValue, DualNumber(short.MinValue));

// Größte short-Zahl darstellen
Console.WriteLine("{0:000}: {1}", short.MaxValue, DualNumber(short.MaxValue));
Console.WriteLine();

Console.WriteLine("ushort-Werte als Dualzahl:");

// Zahlen vom 0 bis +5 als Dualzahl darstellen
for (ushort i = 0; i < 10; i++)
    Console.WriteLine("{0:000}: {1}", i, DualNumber(i));

// Größte ushort-Zahl darstellen
Console.WriteLine("{0:000}: {1}", ushort.MaxValue,
    DualNumber(ushort.MaxValue));

```

Listing 6: Test der Methoden zur Konvertierung von Integerwerten in eine Dualzahl (Forts.)



Abbildung 1: Das Testprogramm in Aktion

Die kleinste Zahl eines Integertyps mit Vorzeichen entspricht übrigens dem Wert 0 mit Vorzeichenbit (weswegen der Absolutwert des kleinsten Werts eines Integer-Typs mit Vorzeichen um 1 größer ist als der Absolutwert des größten Werts). Deswegen gibt das Programm für den kleinsten `short`-Wert (-32768) den Dualwert -1000000000000000 aus.

4 Dualzahlen in Ganzzahlen konvertieren

Ähnlich wie hexadezimal dargestellte Zahlen (vgl. Rezept 2) können Sie auch dual dargestellte Zahlen in einen Integer-Wert konvertieren. Dazu verwenden Sie dieselben Methoden der `Convert`-Klasse wie im genannten Rezept, geben als Basis aber nun 2 an:

```
string dualNumber = "00001010";

// Konvertieren des Dualwerts in eine byte-Zahl
byte number = 0;
try
{
    number = Convert.ToByte(dualNumber, 2);
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Konvertieren: {0}", ex.Message);
}
```

Listing 7: Konvertieren einer Dualzahl in eine dezimale Zahl

Die einzige Einschränkung bei der Verwendung der To-Methoden der `Convert`-Klasse ist, dass Sie keine Dualzahlen konvertieren können, die mit einem Minuszeichen beginnen (ein Pluszeichen ist allerdings erlaubt). Bei Konvertierungen mit den Methoden `ToSByte`, `ToInt16`, `ToInt32` und `ToInt64` wird allerdings das linke Bit als Vorzeichen gewertet. Der Dualwert muss dazu exakt der Bitgröße des jeweiligen Datentyps entsprechen.

5 Zahlen in verschiedenen Notationen darstellen

Zur Formatierung einer Zahl in verschiedene Notationen können Sie spezielle Formatzeichen einsetzen, die Sie z. B. der `ToString`-Methode eines numerischen Datentyps übergeben oder im Formatstring der `Format`-Methode der `String`-Klasse einsetzen können. Das .NET Framework definiert dazu die folgenden Zeichen:

- ▶ *C*: Formatierung als Währung,
- ▶ *e* oder *E*: Formatierung im wissenschaftlichen Format in der Form *MantisseEExponent*,
- ▶ *P*: Formatierung als Prozentwert,
- ▶ *G*: Formatierung als normale Zahl ohne Tausendertrennzeichen,
- ▶ *F*: Formatierung mit mindestens einer Ziffer links vom Dezimaltrennzeichen und zwei Dezimalziffern,
- ▶ *N*: Formatierung als normale Zahl mit Tausendertrennzeichen und zwei Dezimalziffern,
- ▶ *H*: Hexadezimale Formatierung.

Listing 8 zeigt einige Beispiele für die Formatierung in verschiedenen Notationen. Zum Kompilieren dieses Beispiels müssen Sie den Namensraum `System` importieren.

```
// Währung
Console.WriteLine(doubleNumber.ToString("C"));

// Wissenschaftliches Format
Console.WriteLine(doubleNumber.ToString("e"));
Console.WriteLine(doubleNumber.ToString("E"));

// Formatierung als Prozentwert
Console.WriteLine(doubleNumber.ToString("P"));

// Formatierung als normale Zahl ohne Tausendertrennzeichen
Console.WriteLine(doubleNumber.ToString("G"));

// Formatierung mit mindestens einer Ziffer links
// und zwei Ziffern rechts vom Dezimaltrennzeichen
Console.WriteLine(doubleNumber.ToString("F"));

// Formatierung als normale Zahl mit Tausendertrennzeichen
// und zwei Dezimalziffern
Console.WriteLine(doubleNumber.ToString("N"));

// Hexadezimal (nur Integer-Werte)
Console.WriteLine(intNumber.ToString("X"));
```

Listing 8: Darstellung von Zahlen in verschiedenen Notationen



Abbildung 2: Das Beispielprogramm in Aktion

Die Ausgabe des Fragezeichens an Stelle des Euro-Zeichens in Abbildung 2 resultiert übrigens daraus, dass die Windows-Konsole leider einen anderen Zeichensatz verwendet als ein .NET-Programm.

6 Zahlen kaufmännisch runden

Über die `Round`-Methode der `Math`-Klasse können Sie Zahlen mathematisch runden. Am ersten Argument geben Sie die Zahl an, am zweiten die Anzahl der Dezimalziffern, auf die gerundet werden soll:

```
double number1 = 1.15;
double number2 = 1.25;
double number3 = 1.151;
```

```
double result1 = Math.Round(number1, 1);
double result2 = Math.Round(number2, 1);
double result3 = Math.Round(number3, 1);
```

Wie bereits angedeutet rundet `Math.Round` mathematisch: Falls die Ziffer hinter der Dezimalstelle, auf die gerundet wird (im Beispiel ist das die zweite Dezimalstelle), eine Fünf ist und danach keine weiteren Dezimalstellen folgen, wird die Zahl aufgerundet, wenn die Ziffer an der Rundungs-Dezimalstelle ungerade ist, und abgerundet, wenn die Ziffer gerade ist. Im Beispiel resultiert deshalb für alle Zahlen das Ergebnis 1,2. Das mathematische Runden bewirkt, dass keine Ungleichheit beim Runden entsteht. Beim kaufmännischen Runden, bei dem immer aufgerundet wird, wenn die relevante Ziffer eine Fünf ist, werden mehr Zahlen aufgerundet als abgerundet (was den Kaufleuten aber nur recht sein kann 😊). Das liegt daran, dass die Fünf genau in der Mitte der Ziffern von Null bis Neun liegt.

Wenn Sie kaufmännisch runden wollen bzw. müssen, helfen Ihnen die beiden Varianten der Methode `RoundCommercial` in Listing 9, die mit einer `double`- oder einer `decimal`-Zahl arbeiten. Diese Methoden multiplizieren die zu rundende Zahl zunächst mit $10^{\text{Anzahl der Dezimalstellen}}$ um das Komma um die Anzahl der Rundungs-Dezimalstellen nach rechts zu verschieben. Auf diese Zahl wird dann 0,5 addiert und über `Math.Floor` (für `double`) bzw. `Decimal.Truncate` (für `decimal`) der ganzzahlige Anteil extrahiert. Durch die Addition mit 0,5 ist sichergestellt, dass alle Zwischenergebnisse mit einer 5 als erste Dezimalstelle um 1 erhöht werden. Das Zwischenergebnis wird dann nur noch durch den Dezimalstellenfaktor ($10^{\text{Anzahl der Dezimalstellen}}$) geteilt um das Komma wieder nach links zu verschieben.

Zum Kompilieren der `RoundCommercial`-Methode müssen Sie den Namensraum `System` importieren.

```
public static double RoundCommercial(double number, int decimals)
{
    // Faktor für die Kommaverschiebung berechnen
    double decimalFactor = (double)Math.Pow(10, decimals);

    // Zahl runden
    return Math.Floor((number * decimalFactor +
        (0.5F * Math.Sign(number)))) / decimalFactor;
}

public static decimal RoundCommercial(decimal number, int decimals)
{
    // Faktor für die Kommaverschiebung berechnen
    decimal decimalFactor = (decimal)Math.Pow(10, decimals);

    // Zahl runden
    return Decimal.Truncate((number * decimalFactor +
        (0.5M * Math.Sign(number)))) / decimalFactor;
}
```

Listing 9: Methoden zum kaufmännischen Runden von Zahlen

Abbildung 3 zeigt die Ausgabe eines Testprogramms.

```

C:\Codebook\Zahlen kaufmännisch runden\bin\Debug\Zahlen ka...
Math.Round:
1,15 auf zwei Dezimalstellen gerundet: 1,2
1,25 auf zwei Dezimalstellen gerundet: 1,2
1,151 auf zwei Dezimalstellen gerundet: 1,2
1,125000000001 auf drei Dezimalstellen gerundet: 1,13
1,124999999999 auf drei Dezimalstellen gerundet: 1,12

RoundCommercial:
1,15 auf zwei Dezimalstellen gerundet: 1,2
1,25 auf zwei Dezimalstellen gerundet: 1,3
1,151 auf zwei Dezimalstellen gerundet: 1,2
1,125000000001 auf drei Dezimalstellen gerundet: 1,13
1,124999999999 auf drei Dezimalstellen gerundet: 1,12
  
```

Abbildung 3: Mathematisches Runden über `Math.Round` und kaufmännisches Runden über `RoundCommercial`

7 Umstellen der aktuellen Kultur

Wenn Sie Zahlen oder Datumswerte in Strings konvertieren, werden diese entsprechend der aktuell eingestellten Kultur formatiert, sofern Sie bei der Konvertierung keine spezielle Kultur angeben. Dasselbe gilt für das Konvertieren von Strings in Zahlen oder Datumswerte, wo natürlich auch die aktuelle Kultur verwendet wird.

Die aktuelle Kultur ist definiert über ein `CultureInfo`-Objekt (aus dem Namensraum `System.Globalization`), das dem aktuellen Thread zugeordnet ist. Den aktuellen Thread erreichen Sie über die Eigenschaft `CurrentThread` der `Thread`-Klasse (aus dem Namensraum `System.Threading`), das zugeordnete `CultureInfo`-Objekt über dessen Eigenschaft `CurrentCulture`.

Die aktuelle Kultur wird beim Start einer Anwendung über die im System eingestellte Kultur ermittelt. Sie können die aktuelle Kultur jedoch auch auf eine andere setzen. Dazu müssen Sie der `CurrentThread`-Eigenschaft ein `CultureInfo`-Objekt für die gewünschte Kultur übergeben. Ein solches Objekt können Sie über die statische `CreateSpecificCulture`-Methode der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) erzeugen. Dieser Methode übergeben Sie eine der international genormten Kultur-Kurzbezeichnungen. Tabelle 1 stellt einige wichtige dieser Kurzbezeichnungen dar. Die gesamte Liste finden Sie in der Dokumentation der `CultureInfo`-Klasse.

Kurzbezeichnung	Kultur
de	Deutsch allgemein
de-DE	Deutsch in Deutschland
de-AT	Österreichisches Deutsch
de-CH	Schweizer Deutsch
fr	Französisch allgemein

Tabelle 1: Einige wichtige Kultur-Kurzbezeichnungen

Kurzbezeichnung	Kultur
it	Italienisch allgemein
es	Spanisch allgemein
en	Englisch allgemein
en-US	US-amerikanisches Englisch
en-GB	Englisch in Großbritannien

Tabelle 1: Einige wichtige Kultur-Kurzbezeichnungen (Forts.)

Nun können Sie idealerweise beim Start der Anwendung die Kultur z.B. auf die englische Variante umstellen:

```
// CultureInfo-Objekt für die US-englische Kultur erzeugen und dem aktuellen
// Thread zuweisen
Thread.CurrentThread.CurrentCulture =
    CultureInfo.CreateSpecificCulture("en-US");

// Als Beispiel einige Zahlen und Datumswerte formatiert ausgeben
double number = 1234.5678;
Console.WriteLine("{0:#,##0.00}", number);

DateTime now = DateTime.Now;
Console.WriteLine(now.ToString());
```

Listing 10: Umstellen der aktuellen Kultur

Das Beispiel erfordert den Import der Namensräume `System`, `System.Threading` und `System.Globalization`.

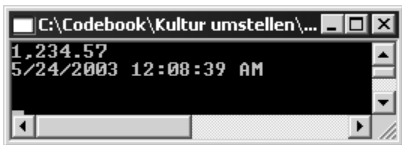


Abbildung 4: Formatierung einer Zahl und eines Datumswerts im US-englischen Format

8 Zahlen in verschiedenen Länderformaten darstellen

In einigen Fällen sollen Zahlen nur für spezielle Ausgaben in länderspezifischen Formaten ausgegeben werden. Die Umstellung der aktuellen Kultur, die ich im Rezept 7 beschrieben habe, ist dazu nicht sinnvoll geeignet.

Den Methoden zur Formatierung können Sie aber an einem Argument ein Objekt übergeben, das die Schnittstelle `IFormatProvider` implementiert. Da die Klasse `CultureInfo` diese Schnittstelle implementiert, können Sie an diesem Argument eine Instanz dieser Klasse über-

geben um Zahlen in einem bestimmten Länderformat auszugeben. Informationen zur Erzeugung eines solchen Objekts finden Sie im Rezept 7.

Das Programm in Listing 11 zeigt, wie Sie eine Zahl über deren `ToString`-Methode im US-englischen Format ausgeben. Das Programm benötigt den Import der Namensräume `System` und `System.Globalization`.

```
double number = 1234.5678;

// CultureInfo-Objekt für die englische Kultur erzeugen
CultureInfo englishCulture = CultureInfo.CreateSpecificCulture("en-US");

// Zahl in das US-englische Format formatieren
string formattedNumber = number.ToString("#,###0.00", englishCulture);
```

Listing 11: Explizites Formatieren einer Zahl im US-englischen Format

9 Einen Bytewert in ein besser lesbares Format umwandeln

Wenn Sie in einem Programm Werte ermitteln, deren Einheit Byte ist, und Sie diese im Programm ausgeben wollen, fördert es die Lesbarkeit, wenn Sie große Werte in deren KB-, MB- oder GB-Entsprechung umwandeln. Dazu müssen Sie lediglich ermitteln, ob der Bytewert jeweils kleiner ist als ein KB, MB bzw. GB, den Wert dann entsprechend teilen und das Kürzel für die Einheit anhängen.

Die Methode `ByteToReadableFormat` in Listing 12 übernimmt diese Aufgabe. Um eine möglichst hohe Flexibilität anzubieten, erwartet die Methode am zweiten Argument einen Formatstring, über den die übergebene Zahl nach der Umrechnung formatiert wird.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` einbinden.

```
public static string ByteToReadableFormat(long byteValue, string format)
{
    if (byteValue < 1024)
        return byteValue + " Byte";
    if (byteValue < 1048576) // 1048576 = 1024 * 1024
        return String.Format("{0:" + format + "} KB", byteValue / 1024F);
    if (byteValue < 1073741824) // 1073741824 = 1024 * 1024 * 1024
        return String.Format("{0:" + format + "} MB", byteValue / 1048576F);
    else
        return String.Format("{0:" + format + "} GB", byteValue / 1073741824F);
}
```

Listing 12: Methode zum Formatieren eines Bytewerts in ein besser lesbares Format

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

`ByteToReadableFormat` gibt den u. U. umgerechneten Wert mit der hinten angehängten Einheit als Zeichenkette zurück. Aus dem Wert 1023 resultiert z.B. der String »1023 Byte«, aus 2222 wird »2,1699 KB« (bei einer Formatierung mit vier Dezimalstellen). Der folgende Programmcode zeigt Beispiele für eine Anwendung dieser Methode:

```
Console.WriteLine(ByteToReadableFormat(1023, "#,##0.####"));
Console.WriteLine(ByteToReadableFormat(1024, "#,##0.00"));
Console.WriteLine(ByteToReadableFormat(2222, "#,##0.####"));
Console.WriteLine(ByteToReadableFormat(3333 * 1024, "#,##0.####"));
Console.WriteLine(ByteToReadableFormat(4444L * 1024 * 1024, "#,##0.####"));
```



Abbildung 5: Die Ausgabe des Beispielprogramms

10 Überprüfen, ob ein String eine gültige Zahl speichert

Immer dann, wenn ein Anwender in einem Programm Zahlen eingeben soll und diese für Berechnungen weiterverarbeitet werden, müssen Sie die eingegebenen Zeichenketten daraufhin überprüfen, ob diese eine gültige Zahl speichern. Die Konvertierung eines Strings, der keine gültige Zahl verwaltet (über die Methoden der `Convert`-Klasse oder die `Parse`-Methoden der numerischen Datentypen), in einen numerischen Datentyp führt ansonsten zu einer Ausnahme vom Typ `FormatException`.

Die einfachste Variante, einen String daraufhin zu überprüfen, ob er eine gültige Zahl speichert, ist, diesen einfach zu konvertieren und die Ausnahme, die beim Konvertieren ungültiger Strings erzeugt wird, abzufangen. Die Methoden `IsNumber` und `IsLong` in Listing 13 nehmen Ihnen diese Arbeit ab. Diese Methoden rufen zur Konvertierung die `Parse`-Methode der Klassen der jeweiligen Datentypen auf.

Ich habe jeweils zwei Varianten dieser Methoden entwickelt. Der ersten können Sie ein Objekt übergeben, das die `IFormatProvider`-Schnittstelle implementiert. Damit können Sie u.a. über ein `CultureInfo`-Objekt die Kultur einstellen, deren Zahlformat für die Überprüfung verwendet werden soll. Die zweite Variante besitzt nur das `string`-Argument und ruft die erste Variante unter der Übergabe der aktuellen Kultur auf.

Zur Verwendung dieser Methoden müssen Sie die Namensräume `System`, `System.Threading` und `System.Globalization` importieren.

```
/* Methoden zur Überprüfung eines Strings auf einen allgemeinen Zahlwert */
public static bool IsNumber(string value, IFormatProvider provider)
{
    try
    {
        Double.Parse(value, provider);
        return true;
    }
    catch
    {
        return false;
    }
}

public static bool IsNumber(string value)
{
    return IsNumber(value, Thread.CurrentThread.CurrentCulture);
}

/* Methoden zur Überprüfung eines Strings auf einen long-Wert */
public static bool IsLong(string value, IFormatProvider provider)
{
    try
    {
        Int64.Parse(value, provider);
        return true;
    }
    catch
    {
        return false;
    }
}

public static bool IsLong(string value)
{
    return IsLong(value, Thread.CurrentThread.CurrentCulture);
}
```

Listing 13: Methoden zur Überprüfung eines Strings auf numerische Daten

Wenn die Methoden `true` zurückgeben, können Sie den String über die `Parse`-Methode sicher konvertieren.

Da die `Parse`-Methode grundsätzlich alle zum Datentyp passenden Formate erlaubt (bei `double` z.B. neben dem normalen Format auch die Exponentialschreibweise), ist die Überprüfung einerseits sehr flexibel. Andererseits gibt `IsNumber` mit der deutschen Kultur auch `true` zurück, wenn die Zahl ein falsches Tausender-Trennzeichen enthält. Der String »1.2« ist z.B. gültig (und ergibt bei der Konvertierung die Zahl 12). Da Anwender in der Praxis häufig versehentlich oder sogar bewusst den Punkt als Dezimaltrennzeichen verwenden, ist die Überprüfung mit `IsNumber` in vielen Fällen gefährlich. Besser wäre es, Tausender-Trennzeichen von vornherein auszuschließen. Und das ist möglich.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateien

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Zur Überprüfung eines Strings auf numerische Daten können Sie auch die `TryParse`-Methode der `Double`-Struktur verwenden. Dieser Methode übergeben Sie am ersten Argument den zu überprüfenden String. Am zweiten Argument können Sie mit den Werten der Aufzählung `System.Globalization.NumberStyles` festlegen, welches Format die eingegebene Zahl besitzen darf. Dieses Argument ist ein Bitfeld, Sie können also mehrere der `NumberStyles`-Konstanten über `|` kombinieren. Über die Konstante `AllowDecimalPoint` legen Sie z.B. fest, dass ein Dezimaltrennzeichen erlaubt ist, `AllowThousands` erlaubt ein Tausender-Trennzeichen, `AllowLeadingSign` ein führendes Vorzeichen und `AllowCurrencySymbol` ein Währungssymbol. Mit diesen und den anderen Konstanten können Sie recht genau festlegen, welches Format die Zahl besitzen darf. `NumberStyles` besitzt daneben aber auch Konstanten für vordefinierte Formate, die sich größtenteils über ihren Namen erklären: `Currency`, `Float`, `HexNumber`, `Integer` und `Number` (wie `Float`, nur mit Tausendertrennzeichen).

Am dritten Argument übergeben Sie der `TryParse`-Methode ein Objekt, das die `IFormatProvider`-Schnittstelle implementiert. Hier können Sie ein `CultureInfo`-Objekt übergeben. Für die aktuelle Kultur übergeben Sie hier `System.Threading.Thread.CurrentThread.CurrentCulture`.

Am letzten Argument, das mit `out` deklariert ist, übergeben Sie schließlich eine `double`-Variable, in die `TryParse` den konvertierten Wert schreibt.

Die Methoden `IsDouble` in Listing 14 setzt diese Art der Überprüfung ein. Sie verwendet als Zahlformat einfach `NumberStyles.Float`, das eine Dezimalzahl erlaubt, allerdings Tausendertrennzeichen verbietet. Um wieder möglichst flexibel zu sein, erwartet eine Variante der Methode am zweiten Argument ein Objekt, das die `IFormatProvider`-Schnittstelle implementiert. Die andere Variante setzt die aktuelle Kultur für die Formatierung ein.

```
public static bool IsDouble(string value, IFormatProvider provider)
{
    double dummy;
    return Double.TryParse(value, NumberStyles.Float, provider, out dummy);
}

public static bool IsDouble(string value)
{
    return IsDouble(value, Thread.CurrentThread.CurrentCulture);
}
```

Listing 14: Methoden zur Überprüfung eines Strings auf einen double-Wert (ohne Tausendertrennzeichen)

Abbildung 6 zeigt ein Programm, das diese Zahl-Überprüfungsmethoden anwendet. Bei der ersten Eingabe wurde die eingegebene Zahl zuerst über `IsNumber` und dann über `IsDouble` überprüft, bei der zweiten über `IsLong`. Die letzte Überprüfung verwendet `IsDouble` mit einem `CultureInfo`-Objekt für die englische Kultur.

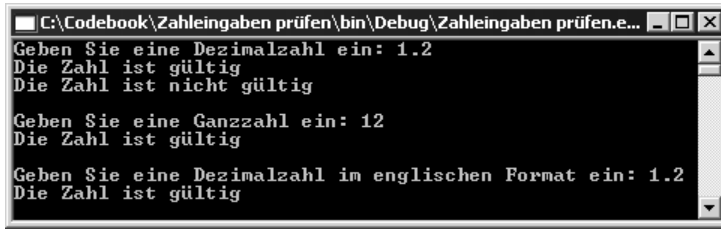


Abbildung 6: Überprüfung eingegebener Zahlen

Hinweis

Die zweite Überprüfung in Abbildung 6 ergibt eine ungültige Zahl, weil hierzu `IsDouble` verwendet und damit das Tausendertrennzeichen ausgeschlossen wird.

11 Überprüfen, ob ein String ein gültiges Datum speichert

Ähnlich wie bei eingegebenen Zahlen ist es sinnvoll, vom Anwender eingegebene Datums- werte vor der Weiterverarbeitung zu prüfen. Dazu können Sie eine ähnliche Technik verwenden wie bei der Überprüfung auf die Gültigkeit einer Zahl.

Ich stelle diese Methode `IsDate` hier nur kurz dar, weil das Kapitel »Datum und Zeit« mit dem Rezept 54 die Lösung noch einmal anbietet (und erläutert). Ich wollte die wichtige Überprüfung von Datumswerten aber an dieser Stelle nicht einfach weglassen (für den Fall, dass Sie in diesem Kapitel nach der Lösung suchen). Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```
public static bool IsDate(string dateString)
{
    // Versuch, den String in ein Datum zu konvertieren
    try
    {
        System.DateTime.Parse(dateString);
        return true;
    }
    catch
    {
        return false;
    }
}

public static bool IsDate(string dateString, CultureInfo culture)
{
    // Versuch, den String in ein Datum zu konvertieren
    try
```

Listing 15: Methoden zur Überprüfung der Gültigkeit eines Datums-Strings

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
{
    System.DateTime.Parse(dateString, culture);
    return true;
}
catch
{
    return false;
}
}
```

Listing 15: Methoden zur Überprüfung der Gültigkeit eines Datums-Strings (Forts.)

12 Zufallszahlen erzeugen

Zufallszahlen können Sie über eine Instanz der Klasse `System.Random` erzeugen. Die Methode `Next` liefert eine zufällig ermittelte `int`-Zahl zurück. Rufen Sie `Next` ohne Argument auf, liegt die Zahl im Bereich zwischen 0 und 2.147.483.647 (dem maximalen `int`-Wert). Sie können dieser Methode aber auch einen Maximalwert oder einen Minimal- und einen Maximalwert übergeben. So können Sie z.B. Zufallszahlen im Bereich von 1 bis 42 erzeugen.

```
Random random = new Random();
int number1 = random.Next(1, 42);
int number2 = random.Next(1, 42);
int number3 = random.Next(1, 42);
```

Die erzeugten Zufallszahlen sind gleich verteilt. Wenn Sie also sehr viele Zahlen erzeugen, ist die Wahrscheinlichkeit groß, dass die einzelnen Zahlen mit einer gleichen Anzahl erzeugt werden.

Die `Random`-Klasse ist für die Generierung echter Zufallszahlen allerdings nicht geeignet: `Random` verwendet eine bestimmte Anzahl festgelegter Zahlenreihen mit `double`-Werten zwischen 0 und 1 als Basis für die Erzeugung der Zufallszahlen. Die nächste Basiszahl können Sie übrigens über die Methode `NextDouble` abrufen. Ein so genannter Seed-Wert (Seed = Saat) bestimmt dabei, welche dieser Zahlenreihen verwendet wird. Im Konstruktor der `Random`-Klasse können Sie den Seed-Wert als `int`-Zahl übergeben. Nun ist es logisch, dass zwei `Random`-Instanzen mit demselben Seed-Wert exakt dieselben Basiszahlen zurückliefern. Identische `Next`-Aufrufe ergeben dann also immer (auch auf verschiedenen Rechnern) identische Zahlenreihen. Mit einem Seed-Wert von 1 erzeugt `Next(1, 42)` z.B. bei drei Aufrufen immer die Folge 11, 5 und 21.

Der Default-Konstruktor der `Random`-Klasse verwendet deswegen die aktuellen Systemticks (`Environment.TickCount`) als Seed-Wert. Damit ist die Wahrscheinlichkeit, identische Zahlenreihen zu erzeugen, einigermaßen gering, aber nicht ausgeschlossen. Wenn Sie mit mehreren `Random`-Instanzen arbeiten und diese direkt hintereinander erzeugen, kann es auf schnellen Rechnern sogar vorkommen, dass diese dieselbe Systemzeit verwenden und damit identische Zahlenreihen liefern. In diesem Fall sollten Sie dem vorbeugen und den Seed-Wert für jede Instanz selbst berechnen (z.B. indem Sie einfach eine beliebige Zahl zu den ausgelesenen Ticks addieren).

Wenn Sie sichere Zufallszahlen erzeugen wollen, können Sie die Klasse `RNGCryptoServiceProvider` aus dem Namensraum `System.Security.Cryptography` verwenden. Diese Klasse wird normalerweise für die Implementierung von speziellen Verschlüsselungsverfahren verwendet.

Die Methode `GetBytes` eines `RNGCryptoServiceProvider`-Objekts füllt ein übergebenes `byte`-Array mit »kryptografisch starken« Zufallszahlen. `GetNonZeroBytes` arbeitet ähnlich, füllt aber nur Zahlen größer 0 in die Arrayelemente. Das `byte`-Array müssen Sie zuvor mit der Größe der Anzahl der zu erzeugenden Zahlen erzeugen. Damit können Sie Zufallszahlen im Bereich von 0 bzw. 1 und 255 erzeugen:

```
Console.WriteLine("Zufallszahlen zwischen 1 und 255");
RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
byte[] randomNumbers = new byte[10];
rng.GetNonZeroBytes(randomNumbers);
for (int i = 0; i < 10; i++)
    Console.WriteLine(randomNumbers[i]);
```

Listing 16: Erzeugen echter Zufallszahlen zwischen 1 und 255

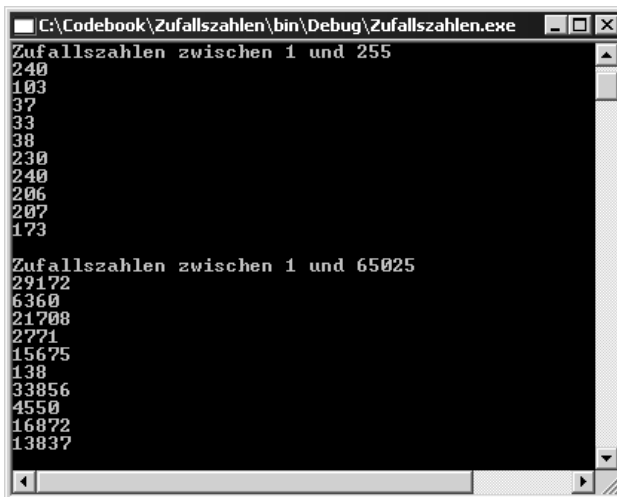


Abbildung 7: Erzeugen echter Zufallszahlen

Um größere Zahlen zu erhalten, können Sie einfach mehrere Arrays füllen und die einzelnen Elemente miteinander multiplizieren:

```
Console.WriteLine("Zufallszahlen zwischen 1 und 65025");
byte[] randomNumbers1 = new byte[10];
byte[] randomNumbers2 = new byte[10];
rng.GetNonZeroBytes(randomNumbers1);
```

Listing 17: Erzeugen echter Zufallszahlen zwischen 1 und 65025

```
rng.GetNonZeroBytes(randomNumbers2);  
for (int i = 0; i < 10; i++)  
    Console.WriteLine(randomNumbers1[i] * randomNumbers2[i]);
```

Listing 17: Erzeugen echter Zufallszahlen zwischen 1 und 65025 (Forts.)

Wenn Sie echte Zufallszahlen in einem bestimmten Bereich erhalten wollen, können Sie die ermittelte Zahl folgendermaßen umrechnen:

```
Divisor = (Zufallszahlen-Bereichsumfang / (Endzahl - Startzahl + 1))  
Zahl = (byte)((Zahl / Divisor) + Startzahl)
```

Wenn Sie Zufallszahlen im Bereich von 0 bis 255 erzeugen und Zahlen im Bereich von 10 bis 20 erhalten wollen, sieht die Berechnung der einzelnen Zahlen folgendermaßen aus:

```
double divisor = 256 / 20;  
zahl = (byte)((zahl / divisor ) + 10);
```

Die Methode `RandomByteNumber` in Listing 18 zeigt dies für Zahlen im Byte-Bereich. Diese Methode benötigt den Import der Namensräume `System` und `System.Security.Cryptography`.

```
public static byte[] RandomNumbers(int count, byte min, byte max)  
{  
    // Zufallszahlen erzeugen  
    RNGCryptoServiceProvider csp = new RNGCryptoServiceProvider();  
    byte[] numbers = new Byte[count];  
    csp.GetBytes(numbers);  
  
    // Die Zahlen umrechnen  
    double divisor = 256F / (max - min + 1);  
    if (min > 0 || max < 255)  
    {  
        for (int i = 0; i < count; i++)  
        {  
            numbers[i] = (byte)((numbers[i] / divisor) + min);  
        }  
    }  
  
    return numbers;  
}
```

Listing 18: Erzeugen echter Zufallszahlen in einem gegebenen Bereich

Ein Test dieser Methode hat bei 10000 erzeugten Zahlen eine recht gute Gleichverteilung ergeben. Sie finden diesen Test als Teil des Beispiels zu diesem Rezept auf der Buch-CD.

13 High- und Low-Word lesen

Einige Windows-API-Funktionen geben numerische Daten in einem (normalerweise vorzeichenlosen) 32-Bit-Integerwert zurück, dessen hohes und niedriges Wort (ein Wort hat 16 Bit) unterschiedliche Informationen speichern. Der `uint`-Wert `0xAAAAFFFF` speichert z. B. im High-Word den Wert `0xAAAA` und im Low-Word den Wert `0xFFFF`.

Sie können das hohe Wort eines vorzeichenlosen 32-Bit-Werts extrahieren, indem Sie den Wert um 16 Bit nach rechts shiften. Das niedrige Wort erhalten Sie über eine bitweise Addition mit `0xFFFF`.

Die Methoden `GetHighWord` und `GetLowWord` in Listing 19 gehen auf diese Weise vor um das hohe und das niedrige Wort in je einer Variante aus einem `uint`-Wert bzw. `int`-Wert zu extrahieren. Das Ergebnis der Berechnung wird bei der Rückgabe in einen `ushort`-Wert umgewandelt, da die Berechnung selbst einen `uint`- bzw. `int`-Wert ergibt.

`GetHighWord` und `GetLowWord` erfordern den Import des Namensraums `System`.

```
/* Methode zum Extrahieren des High-Word aus einem uint-Wert */
public static ushort GetHighWord(uint value)
{
    return (ushort)(value >> 16);
}

/* Methode zum Extrahieren des Low-Word aus einem uint-Wert */
public static ushort GetLowWord(uint value)
{
    return (ushort)(value & 0xFFFF);
}

/* Methode zum Extrahieren des High-Word aus einem int-Wert */
public static ushort GetHighWord(int value)
{
    return (ushort)(value >> 16);
}

/* Methode zum Extrahieren des Low-Word aus einem int-Wert */
public static ushort GetLowWord(int value)
{
    return (ushort)(value & 0xFFFF);
}
```

Listing 19: Methoden zum Extrahieren des hohen und des niedrigen Worts eines 32-Bit-Integerwerts

Abbildung 8 zeigt die Ausgabe eines Programms, das das High- und das Low-Word über die Methoden in Listing 19 für die `uint`-Zahl `0xAAAAFFFF` berechnet. Diese Zahl wurde dann über einen Typecast in eine `int`-Variable geschrieben und damit ebenfalls das High- und das Low-Word berechnet.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

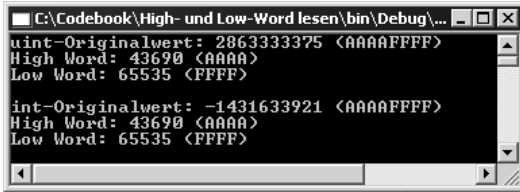


Abbildung 8: Berechnung des High- und des Low-Word einer uint- und einer int-Zahl

14 High- und Low-Word setzen

Wenn Sie Windows-API-Funktionen aufrufen, die einen 32-Bit-Integerwert erwarten, dessen Low- und High-Word eine separate Bedeutung besitzen, müssen Sie diese separat setzen. Dazu können Sie zwei `ushort`-Variablen verwenden, die die Werte des hohen und des niedrigen Worts speichern. Zum Setzen des hohen Worts shiften Sie die `ushort`-Variable um 16 Bit nach links und addieren das Ergebnis mit dem um das hohe Wort reduzierten Integerwert. Das hohe Wort des Integerwerts erhalten Sie über eine bitweise Addition mit `0xFFFF`. Damit erreichen Sie, dass nur das hohe Wort gesetzt und der Wert des niedrigen Worts nicht geändert wird. Die Methode `SetHighWord` in Listing 20, die in je einer `uint`- und einer `int`-Variante implementiert ist, zeigt, wie diese Berechnung ausgeführt wird. Das Ergebnis des Linksshift muss dabei in einen `uint`- bzw. `int`-Wert umgewandelt werden, da dieses den Typ `long` besitzt.

Um das niedrige Wort zu setzen, addieren Sie den Integerwert bitweise mit `0xFFFF0000`, um den Wert dieses Worts zu löschen, und addieren einfach das neue Wort auf das Ergebnis. Die Methode `SetLowWord` in Listing 20 zeigt, wie das ausgeführt wird. In der `int`-Variante dieser Methode ergibt die bitweise Addition einen `long`-Wert. Das liegt daran, dass der Wert `0xFFFF0000` vom Compiler nicht als `uint`, sondern als `long` ausgewertet wird, weil ein `int`-Wert in der Berechnung enthalten ist. Deshalb muss das Ergebnis der bitweisen Addition in einen `int`-Wert umgewandelt werden. In der `uint`-Variante ist diese Umwandlung nicht notwendig, da der Compiler den Wert `0xFFFF0000` hier als `uint` auswertet, weil die Addition ebenfalls mit einem `uint`-Wert erfolgt.

Zum Kompilieren dieser Methoden müssen Sie den Namensraum `System` importieren.

```

/* Methode zum Setzen des High-Word in einer uint-Variablen */
public static uint SetHighWord(uint value, ushort highWord)
{
    return (value & 0xFFFF) + (uint)(highWord << 16);
}

/* Methode zum Setzen des Low-Word in einer uint-Variablen */
public static uint SetLowWord(uint value, ushort lowWord)
{
    return (value & 0xFFFF0000) + lowWord;
}

```

Listing 20: Methoden zum Setzen des High- und des Low-Word in einem `uint`- oder `int`-Wert

```

/* Methode zum Setzen des High-Word in einer int-Variablen */
public static int SetHighWord(int value, ushort highWord)
{
    return (value & 0xFFFF) + (int)(highWord << 16);
}

/* Methode zum Setzen des Low-Word in einer uint-Variablen */
public static int SetLowWord(int value, ushort lowWord)
{
    return (int)(value & 0xFFFF0000) + lowWord;
}

```

Listing 20: Methoden zum Setzen des High- und des Low-Word in einem uint- oder int-Wert (Forts.)

15 Beep ausgeben

Eigenartigerweise besteht in den Klassen des .NET Framework keine Möglichkeit, einen einfachen Piepston (Beep) auszugeben. In vielen Fällen ist es aber sinnvoll, dem Anwender akustisch mitzuteilen, dass irgendetwas passiert ist.

Für die Ausgabe eines Piepstons muss das Windows-API herangezogen werden. Die Funktion `MessageBeep` ist in der Lage einfache, aber auch spezielle akustische Meldungen auszugeben. Im einzigen Argument übergeben Sie dieser Funktion dazu einen `uint`-Wert. Die verwendeten Konstanten sind in der Datei `winuser.h` deklariert und besitzen den Präfix `MB_`. Wenn Sie sich die Konstanten in Listing 21 anschauen, werden Sie erkennen, dass diese bis auf `MB_SIMPLE_BEEP` eng mit den Icons einer MessageBox zusammenhängen. Die Töne dieser Konstanten sind im System festgelegt und werden normalerweise als Wave-Datei über die Soundkarte abgespielt. Die von mir definierte Konstante `MB_SIMPLE_BEEP` existiert übrigens nicht in der Headerdatei. Den Wert dieser Konstante habe ich der Dokumentation vom `MessageBeep` entnommen.

Um nun möglichst flexibel zu sein, arbeitet die Methode `MessageBeep` in Listing 21 mit einem Wert der Aufzählung `BeepType`. Diese Aufzählung setzt die API-Konstanten in eigene Konstanten um. Beim Aufruf der `MessageBeep`-Funktion wird der Konstantenwert dann einfach in den `uint`-Wert umgewandelt, damit die Funktion den passenden Meldungston erzeugt. `MessageBeep` gibt einen Wert ungleich Null zurück, wenn die Ausführung erfolgreich war. Da Fehler recht unwahrscheinlich sind, gibt die `MessageBeep`-Methode in Listing 21 lediglich einen booleschen Wert zurück, der mit `true` aussagt, dass die Ausführung erfolgreich war.

Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```

/* Deklaration der API-Funktion MessageBeep und benötigter Konstanten */
[DllImport("User32.dll")]
private static extern int MessageBeep(uint uType);

private const uint MB_SIMPLE_BEEP = 1;

```

Listing 21: Methode zur Ausgabe eines Meldungstons

```

private const uint MB_ICONHAND = 0x00000010;
private const uint MB_ICONQUESTION = 0x00000020;
private const uint MB_ICONEXCLAMATION = 0x00000030;
private const uint MB_ICONASTERISK = 0x00000040;

/* Aufzählung für die Art des Piepstons */
public enum BeepType: uint
{
    Simple = MB_SIMPLE_BEEP,
    Error = MB_ICONHAND,
    Exclamation = MB_ICONEXCLAMATION,
    Question = MB_ICONQUESTION,
    Information = MB_ICONASTERISK
}

/* Methode zur Ausgabe eines Piepstons */
public static bool MessageBeep(BeepType beepType)
{
    return (MessageBeep((uint)beepType) != 0);
}

```

Listing 21: Methode zur Ausgabe eines Meldungs-Tons (Forts.)

Das Windows-API bietet übrigens über die Funktion `Beep` noch eine weitere Möglichkeit, einen Meldungston auszugeben. Dieser Funktion können Sie eine Frequenz und einen Wert übergeben, der die Länge des auszugebenden Tons in Millisekunden angibt. Leider funktioniert `Beep` nur dann, wenn keine Soundkarte installiert ist, da diese Funktion den Lautsprecher des Computers anspricht.

Hinweis

In den Rezepten 214, 215 und 216 finden Sie Methoden bzw. Klassen zum Abspielen von Wave- und anderen Multimedia-Dateien. Darüber können Sie dem Anwender auch akustisch mitteilen, dass ein besonderer Zustand eingetreten ist.

16 Strings ohne Berücksichtigung der Groß-/Kleinschreibung vergleichen

Beim direkten Vergleich zweier Strings berücksichtigt der Vergleichsoperator der `string`-Klasse die Schreibweise der einzelnen Zeichen. Der String »Hallo« ist bei einem solchen Vergleich ungleich dem String »hallo«. Wenn Sie zwei Strings ohne Berücksichtigung der Groß-/Kleinschreibung vergleichen wollen, können Sie diese einfach mit der `CompareTo`-Methode der `String`-Klasse vergleichen: Geben Sie neben den zu vergleichenden Strings am dritten Argument (`ignoreCase`) `true` an. `Compare` gibt 0 zurück wenn beide Strings gleich sind.

```

string string1 = "Hallo";
string string2 = "hallo";

```

Listing 22: Case-insensitiver Vergleich von zwei Zeichenketten


```
if (String.Compare(string1, string2, true) == 0)
    Console.WriteLine("'{0}' und '{1}' sind gleich", string1, string2);
else
    Console.WriteLine("'{0}' und '{1}' sind nicht gleich", string1, string2);
```

Listing 22: Case-insensitiver Vergleich von zwei Zeichenketten (Forts.)

Hinweis

Der ebenfalls denkbare Vergleich über ein Umwandeln der Strings in Klein- oder Großschreibung (über `ToLower` bzw. `ToUpper`) ist übrigens nicht zu empfehlen, da `Compare` wesentlich effizienter arbeitet. Ein Vergleich mit `Compare` ist nur unwesentlich langsamer als ein direkter Vergleich.

17 Teilstrings flexibel ersetzen

Strings bieten über die Methode `Replace` eine einfache Möglichkeit, einzelne Zeichen oder Teilstrings im String zu ersetzen:

```
string source = "abc AAA abc aaa abc AAA";
string find = "aaa";
string replacement = "xyz";
string source = "abc aaa abc aaa abc AAA";
string result = source.Replace(find, replacement);
```

Ein wesentlicher Nachteil dieser Methode ist, dass Sie nicht bestimmen können, dass die Groß-/ Kleinschreibung nicht berücksichtigt werden soll. Das Ergebnis der Ersetzung im obigen Code ist z.B. »abc xyz abc xyz abc AAA«. Die großgeschriebenen As werden nicht ersetzt. Ein (je nach Anwendung) eventueller anderer Nachteil ist, dass Sie nicht bestimmen können, ab welchem Suchstring ersetzt werden soll und wie viele Ersetzungen maximal vorgenommen werden sollen.

Die `Replace`-Methode eines `StringBuilder`-Objekts erlaubt hingegen auch die Angabe einer Startposition und der Länge des Teilstrings, in dem gesucht werden soll. So können Sie z.B. einen zehn Zeichen langen Teilstring ab dem sechsten Zeichen ersetzen:

```
string source = "abc AAA abc aaa abc AAA";
string find = "aaa";
string replacement = "xyz";
int start = 5;
int count = 10;
System.Text.StringBuilder sb = new System.Text.StringBuilder(source);
string result = sb.Replace(find, replacement, start, count).ToString();
```

Dieses Beispiel ergibt den String »abc aaa abc xyz abc AAA«.

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die `Replace`-Methode eines regulären Ausdrucks in Form einer Instanz der `Regex`-Klasse (aus dem Namensraum `System.Text.RegularExpressions`) erlaubt zudem das Ersetzen ohne Berücksichtigung der Groß-/Kleinschreibung. Am Argument `count` wird hier, anders als beim `StringBuilder`-Objekt, angegeben, wie viele Ersetzungen maximal erfolgen sollen.

```
string source = "abc AAA abc aaa abc AAA";
string find = "aaa";
string replacement = "xyz";
int start = 0;
int count = 2;
Regex re = new Regex(find, RegexOptions.IgnoreCase);
string result = re.Replace(source, replacement, count, start);
```

Das Ergebnis der `Replace`-Methode ist in diesem Fall »abc xyz abc xyz abc AAA«, da die Groß-/Kleinschreibung nicht berücksichtigt wurde und zwei Ersetzungen erfolgt sind. Wenn Sie am dritten Argument `-1` übergeben, werden alle Fundstellen ersetzt. Das letzte Argument der im Beispiel verwendeten Variante der `Replace`-Methode definiert übrigens die Zeichenposition, ab der gesucht werden soll. Leider können Sie hier nicht definieren, ab welcher Fundstelle ersetzt werden soll.

Abgesehen davon besitzt die `Replace`-Methode eines `Regex`-Objekts aber zwei gravierende Nachteile: Zum einen müssen Sie dafür sorgen, dass das Suchmuster korrekt ist. Reguläre Ausdrücke verwenden den Backslash als Escape-Zeichen und einige andere Sonderzeichen wie z.B. geschweifte und runde Klammern. Wenn diese Sonderzeichen in Ihrem Suchstring vorkommen, müssen Sie einen Backslash vor das Zeichen setzen, um die Sonderbedeutung aufzuheben.

Zum anderen ist die Ausführung der `Regex.Replace`-Methode gegenüber den anderen Möglichkeiten sehr langsam. In einem Test benötigte diese Methode für 10.000 Ersetzungen etwa 0,3 Sekunden, die `Replace`-Methode einer `string`-Instanz hingegen nur ca. 0,0077 Sekunden. Einen Vorteil will ich aber nicht verschweigen: Sie können hier natürlich reguläre Ausdrücke einsetzen, was die Suche sehr flexibel macht.

Eine weitere Variante zum Ersetzen von Strings ist die Verwendung der `Visual-Basic-Replace`-Methode. Zur Verwendung dieser Methode müssen Sie die Assembly *Microsoft.VisualBasic.dll* referenzieren. In der Verweisliste von Visual Studio wird diese Assembly unter ihrem langen Namen *Microsoft Visual Basic .NET Runtime* aufgeführt. Die `Replace`-Methode finden Sie dann im Modul `Strings` im Namensraum `Microsoft.VisualBasic`.

Am ersten Argument übergeben Sie den Quellstring, am zweiten den Suchstring und am dritten den Ersatzstring. Das vierte Argument gibt die Position an, ab der im Quellstring gesucht wird. Der Index beginnt hier anders als in C# mit 1. Dummerweise gibt die `VB-Replace`-Methode den String ab der Startposition zurück (und nicht komplett), wenn Sie eine Startposition größer 1 angeben (was ein alter Visual-Basic-Bug ist).

Dafür können Sie am fünften Argument bestimmen, wie viele Ersetzungen maximal vorgenommen werden sollen. Wenn Sie hier `-1` angeben, werden alle Vorkommen des Suchstrings ab der Startposition ersetzt.

Das letzte Argument steuert, ob die Groß-/Kleinschreibung berücksichtigt werden soll. Hier können Sie die Werte der `CompareMethod`-Aufzählung angeben. Der Wert `Binary` führt zu einem Vergleich unter Berücksichtigung der Groß-/Kleinschreibung, der Wert `Text` zu einem Textvergleich. Das folgende Beispiel ersetzt in einem String maximal zwei Vorkommen eines Teilstrings ohne Berücksichtigung der Groß-/Kleinschreibung:

```
string source = "abc AAA abc aaa abc AAA";
string find = "aaa";
string replacement = "xyz";
int start = 1;
int count = 2;
result = Strings.Replace(source, find, replacement, start, count,
    CompareMethod.Text);
```

Das Ergebnis des `Replace`-Aufrufs ist hier »abc xyz abc xyz abc AAA«.

Da ich nicht davon begeistert bin, Anleihen bei Visual Basic zu machen, und da ich in einigen Fällen statt der Anfangs-Suchposition angeben wollte, ab welchem Suchstring ersetzt werden soll, habe ich dann eine eigene `Replace`-Methode geschrieben (Listing 23). `Replace` verwendet die Methoden `IndexOf` und `Substring` des `string`-Objekts um den gesuchten Teilstring zu suchen und durch den neuen Teilstring zu ersetzen. Am Argument `ignoreCase` können Sie angeben, dass die Groß-/Kleinschreibung nicht berücksichtigt werden soll. Zur Umsetzung schreibt `Replace` den übergebenen String in eine lokale Kopie, die in Kleinschrift umgewandelt wird wenn `ignoreCase true` ist. In diesem Fall wird auch der Suchstring in Kleinschrift umgewandelt.

`Replace` sucht dann einfach ab einer gegebenen Position, die bei 0 startet, in einer Schleife nach dem Suchstring. Findet es diesen, hängt `Replace` den aktuellen Teilstring ab der letzten Startposition bis zur Position des Teilstrings an eine Ergebnis-Stringvariable an. Die Startposition wird dann für den nächsten Durchlauf auf die Position hinter dem Suchstring gesetzt und erneut nach dem Suchstring gesucht. Die Schleife läuft so lange wie noch ein weiterer Suchstring gefunden wird.

In der Schleife werden jede Fundstelle und jeder Ersatz gezählt. Mit diesen Werten vergleicht `Replace` dann die Angaben, die an den Argumenten `start` und `count` übergeben wurden. Das Ersetzen wird nur dann ausgeführt, wenn der aktuelle Suchzähler größer oder gleich `start` ist. Ist `count` mit einem Wert größer -1 angegeben, wird die Schleife unterbrochen, wenn die Anzahl der Ersetzungen `count` entspricht. Unter der Schleife wird dann noch der eventuell übrig gebliebene rechte Teilstring an das Ergebnis und dieses zurückgegeben.

```
public static string Replace(string source, string find,
    string replacement, bool ignoreCase, int start, int count)
{
    int pos1 = 0, pos2 = 0;
    int findStringLength = find.Length;
```

Listing 23: Methode zum flexiblen Ersetzen von Strings

```
string baseString;
string result = "";

// Den Sonderfall abhandeln, dass count mit 0 angegeben ist
if (count == 0)
{
    return source;
}

// Wenn der Vergleich ohne Berücksichtigung der Groß-/Kleinschreibung
// erfolgen soll, werden der zu durchsuchende und der Suchstring einfach in
// Kleinschreibung umgewandelt
if (ignoreCase)
{
    baseString = source.ToLower();
    find = find.ToLower();
}
else
    baseString = source;

// Erstes Vorkommen des Suchstrings suchen
pos2 = baseString.IndexOf(find);

// Den String durchgehen, so lange der Suchstring gefunden wurde
int findCount = 0;
int replaceCount = 0;
while (pos2 != -1)
{
    findCount++;
    if (findCount >= start)
    {
        // Das Ergebnis zusammensetzen und die Such-Indizes auf die neuen
        // Positionen setzen
        result += source.Substring(pos1, pos2 - pos1) + replacement;
        pos1 = pos2 + findStringLen;
        pos2 = baseString.IndexOf(find, pos1);

        // Wenn count größer -1 definiert ist, überprüfen, ob die Anzahl
        // erreicht ist
        replaceCount++;
        if (count > -1 && replaceCount >= count)
            break;
    }
    else
    {
        result += source.Substring(pos1, pos2 - pos1 + findStringLen);
        pos1 = pos2 + findStringLen;
        pos2 = baseString.IndexOf(find, pos1);
    }
}
```

Listing 23: Methode zum flexiblen Ersetzen von Strings (Forts.)

```
    if (pos1 < source.Length)
    {
        // Wenn nach dem letzten Suchstring noch ein Teilstring gespeichert ist,
        // diesen hinten anhängen
        result += source.Substring(pos1, source.Length - pos1);
    }

    return result;
}
```

Listing 23: Methode zum flexiblen Ersetzen von Strings (Forts.)

Replace benötigt den Import des Namensraums `System.Text.RegularExpressions`.

Listing 24 zeigt eine Anwendung dieser Methode.

```
string source = "abc aaa abc aaa abc AAA";
Console.WriteLine("Quelle: {0}", source);

string result = Replace(source, "aaa", "xyz", true, 1, -1);
Console.WriteLine("Alle Fundstellen ersetzt: {0}", result);

result = Replace(source, "aaa", "xyz", true, 2, -1);
Console.WriteLine("Ab der zweiten alle weiteren ersetzt: {0}", result);

result = Replace(source, "aaa", "xyz", true, 2, 1);
Console.WriteLine("Ab der zweiten nur eine ersetzt: {0}", result);
```

Listing 24: Beispiel zur Anwendung der Replace-Methode

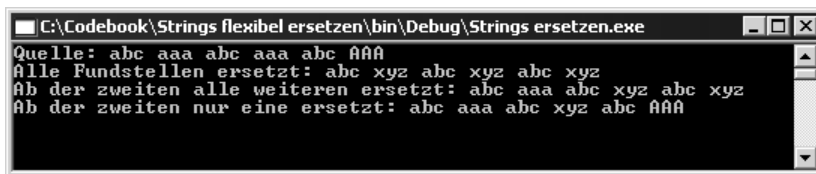


Abbildung 9: Das Beispielprogramm in Aktion

Ich habe natürlich (mit Hilfe der Klasse `HighResStopClock` aus dem Rezept 58) Performance-Vergleiche durchgeführt um meine Version der `Replace`-Methode abzusichern. Tabelle 2 zeigt die Ergebnisse für 10.000 Aufrufe auf einem Rechner mit 2,4 GHz Prozessor, 512 MB Arbeitsspeicher und Windows XP. Zur Messung habe ich das Programm ohne Debuginformationen kompiliert und direkt (ohne Visual Studio) aufgerufen. Das Programm finden Sie in den Beispielen zu diesem Kapitel neben diesem Rezept.

Methode	Benötigte Zeit für 10.000 Aufrufe in Sekunden
<code>string.Replace</code>	0,00693216595964012
<code>StringBuilder.Replace</code>	0,0142213605360458
<code>Regex.Replace</code> mit binärem Vergleich	0,0879824111723697
<code>Regex.Replace</code> mit Textvergleich	0,0764404414527545
<code>Visual-Basic-Replace</code> mit binärem Vergleich	0,0268902891289256
<code>Visual-Basic-Replace</code> mit Textvergleich	0,0301678006562287
Eigene <code>Replace</code> -Methode mit binärem Vergleich	0,0265237620982555
Eigene <code>Replace</code> -Methode mit Textvergleich	0,0408199924850784

Tabelle 2: Performancevergleich der verschiedenen `Replace`-Methoden

Der absolute Gewinner ist die `Replace`-Methode der `string`-Klasse. Immer wenn es auf Performance ankommt und Sie die erweiterten Möglichkeiten der anderen Methoden nicht benötigen, sollten Sie diese Methode anwenden.

Die Verwendung eines `StringBuilder`-Objekts ist trotz der in der Schleife vorgenommenen Instanzierung (um einen korrekten Vergleich zu erhalten habe ich das `StringBuilder`-Objekt immer wieder neu instanziiert) erstaunlich performant.

Die `Regex`-Variante ist deutlich langsamer als die anderen und sollte nur eingesetzt werden, wenn Sie zum Vergleich einen regulären Ausdruck einsetzen müssen.

Was mich erstaunte, war allerdings die Tatsache, dass meine `Replace`-Methode mit der von Visual Basic ganz gut mithalten kann. Es besteht also eigentlich kein Grund, die `Visual-Basic`-Variante zu verwenden 😊.

18 Mehrere (Leer-)Zeichen in ein (Leer-)Zeichen konvertieren

Strings enthalten manchmal mehrere Leerzeichen, was besonders dann gilt, wenn die Eingaben von einem Anwender stammen. In vielen Fällen sind diese mehrfachen Leerzeichen störend und müssen durch ein Leerzeichen ersetzt werden. Diese Aufgabe ist einfach zu lösen. Ich habe einige Varianten der Programmierung verglichen (u. a. eine eigene Programmierung, die eine vereinfachte Variante der `Replace`-Methode aus dem Rezept 17 darstellte). Die schnellste Variante war gleichzeitig auch die einfachste. Die `ToSingleSpace`-Methode in Listing 25 setzt diese Variante um, die einfach nur in einer Schleife die `Replace`-Methode des `string`-Objekts aufruft, bis keine doppelten Leerzeichen mehr gefunden werden. Um noch ein wenig mehr Flexibilität zu besitzen, habe ich zudem die (nur unwesentlich langsamere) Methode `ToSingleChar` entwickelt, der Sie das Zeichen übergeben können, dessen mehrfaches Vorkommen in ein Vorkommen konvertiert werden soll. `ToSingleChar` erzeugt dazu einfach die zwei Stringvariablen `find` und `replace` mit zwei bzw. einem der übergebenen Zeichen und ruft damit die `Replace`-Methode des `string`-Objekts auf.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` importieren.

```
/* Methode zum Konvertieren mehrfacher Leerzeichen in ein Leerzeichen */
public static string ToSingleSpace(string source)
{
    while (source.IndexOf(" ") > -1)
    {
        source = source.Replace(" ", " ");
    }

    return source;
}

/* Methode zur Konvertierung mehrfacher beliebiger Zeichen in ein Zeichen */
public static string ToSingleChar(string source, char c)
{
    string find = new string(c, 2);
    string replace = new string(c, 1);
    while (source.IndexOf(find) > -1)
    {
        source = source.Replace(find, replace);
    }

    return source;
}
```

Listing 25: Methoden zum Ersetzen mehrfacher Leer- und anderer Zeichen

19 Erstes Zeichen aller Wörter im String groß- und die restlichen Zeichen kleinschreiben

Um Strings in eine normalisierte Form zu bringen, ist es manchmal notwendig, die ersten Zeichen aller Wörter im String groß- und die folgenden kleinzuschreiben. Damit können Sie die oft unterschiedlichen Eingaben von Anwendern einigermaßen in eine gleichmäßige Form bringen. Wenn Sie dann noch mehrfache Leerzeichen entfernen, wie ich es im Rezept 18 beschreibe, erhalten Sie einen recht gut formatierten String.

Bei der vorliegenden Aufgabe ist das Problem eigentlich nur, die einzelnen Wörter zu extrahieren. Sie könnten dazu die `Split`-Methode eines Strings verwenden. Da Sie dieser Methode ein Zeichen-Array übergeben können, können Sie die unterschiedlichen Wort-Trennzeichen wie z.B. ein Leerzeichen oder einen Bindestrich berücksichtigen. Problematisch ist daran allerdings das korrekte Wieder-Zusammensetzen des Strings (mit den richtigen Wort-Trennzeichen).

Wenn Sie nur die ersten Zeichen jedes Worts im String großschreiben wollen, können Sie die Methode `TextInfo.ToTitleCase` eines `CultureInfo`-Objekts verwenden. Laut der Dokumentation konvertiert diese Methode zwar angeblich jedes Wort normalerweise in eine Kombination aus großgeschriebenem Anfangsbuchstaben und kleingeschriebenem Rest, im Test mit der deutschen und der englischen Kultur war das aber leider nicht der Fall. Nur das erste Zei-

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

chen wurde großgeschrieben, eventuell großgeschriebene folgende Zeichen wurden nicht verändert.

Die Konvertierung erfolgt hierbei kulturabhängig. Wenn Sie die aktuelle Kultur berücksichtigen wollen, können Sie die `CurrentCulture`-Eigenschaft der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) verwenden:

```
string source = "das IST ein TEST-string";
string result = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(source);
```

Dieses Beispiel ergibt (auf einem Rechner, der die deutsche Kultur verwendet) den String »Das IST Ein TEST-String«.

Da `ToTitleCase` scheinbar nicht wie in der Dokumentation beschrieben funktioniert, habe ich eine eigene Methode `NormalizeWords` entwickelt, die alle Wörter im übergebenen String mit großgeschriebenem Anfangsbuchstaben und kleingeschriebenem Rest formatiert.

`NormalizeWords` nutzt dabei den regulären Ausdruck `\w{1,}`, der die Stellen im String ermittelt, die beliebig viele direkt aufeinander folgende Wortzeichen (das sind alle Buchstaben, Zahlen und der Unterstrich) enthalten. Dieses Muster wird über die `Replace`-Methode einer `Regex`-Instanz (aus dem Namensraum `System.Text.RegularExpressions`) ersetzt. Für das Ersetzen habe ich eine Methode `NormalizeWordMatchEvaluator` geschrieben, deren Signatur dem Delegate `MatchEvaluator` entspricht. Diese Methode, die der `Replace`-Methode am zweiten Argument über den Delegate übergeben wird, wird von `Replace` für jede Fundstelle aufgerufen. Dabei bekommt sie ein `Match`-Objekt (»Treffer-Objekt«) übergeben, in dessen Eigenschaft `Value` der String gespeichert ist, der dem Muster entspricht. Diesen String wandelt `NormalizeWordMatchEvaluator` dann so um, dass er mit einem Großbuchstaben beginnt und mit Kleinschrift weitergeht, und gibt das Ergebnis zurück (womit `Replace` den gefundenen String durch den neuen ersetzt).

`NormalizeWords` benötigt zum Kompilieren den Import der Namensräume `System` und `System.Text.RegularExpressions`.

```
/* Methode für den MatchEvaluator-Delegate der Regex-Instanz */
private static string NormalizeWordMatchEvaluator(Match match)
{
    int length = match.Value.Length;
    if (length > 1)
        return match.Value.Substring(0, 1).ToUpper() +
            match.Value.Substring(1, length - 1).ToLower();
    else
        return match.Value.ToUpper();
}

/* Methode zur Normalisierung aller Wörter in einem String */
public static string NormalizeWords(string source)
```

Listing 26: Methode zur Normalisierung von Wörtern in einem String


```
{  
    // Regex-Objekt mit einem Muster erzeugen, das alle Wörter  
    // im String ermittelt  
    Regex re = new Regex(@"\w{1,}");  
  
    // Die Fundstellen durch die Rückgabe der MatchEvaluator-Methode  
    // ersetzen  
    return re.Replace(source,  
        new MatchEvaluator(NormalizeWordMatchEvaluator));  
}
```

Listing 26: Methode zur Normalisierung von Wörtern in einem String (Forts.)

Diese Methode ist eventuell nicht sehr performant. Durch die Verwendung des `Regex`-Objekts mit dem Muster `\w{1,}` ist aber sichergestellt, dass immer alle Wörter im String umgewandelt werden. Eine eigene Programmierung wäre dagegen sehr aufwändig und fehlerträchtig.

Listing 27 zeigt eine Beispiel-Anwendung, die auch die Variante mit der `ToTitleCase`-Methode beinhaltet.

```
string source = "das IST ein TEST-string";  
Console.WriteLine("Quelle: {0}", source);  
  
// Ersten Buchstaben über die TextInfo.ToTitleCase-Methode eines  
// CultureInfo-Objekts großschreiben  
string result = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(source);  
Console.WriteLine("ToTitleCase: {0}", result);  
  
// Wörter über NormalizeWords normalisieren  
result = NormalizeWords(source);  
Console.WriteLine("NormalizeWords: {0}", result);
```

Listing 27: Beispiel zum Normalisieren von Wörtern in einem String



Abbildung 10: Das Beispiel in Aktion

20 Anzahl der Wörter in einem String ermitteln

Die Anzahl der Wörter in einem String können Sie über den regulären Ausdruck `\w{1,}` ermitteln. `\w` steht hier für ein beliebiges Wort-Zeichen (Buchstaben, Zahlen und der Unterstrich). `{1,}` steht dafür, dass das zuvor definierte Zeichen mindestens einmal hintereinander vorkommen muss. Gegenüber einer eigenen Programmierung bietet die Verwendung eines

Basics

Datum
Zeit

Anwen-
dungen

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

regulären Ausdrucks den Vorteil, dass er sehr einfach und auch in Bezug auf die Unicode-Zeichen der Sprachen mit einer anderen als der lateinischen Schriftsprache (Griechisch, Kyrillisch etc.) sicher ist.

Die Methode `WordCount1` in Listing 28 setzt dies um. Das Suchmuster wird neben dem Quellstring der statischen `Matches`-Methode der `Regex`-Klasse (aus dem Namensraum `System.Text.RegularExpressions`) übergeben. Diese Methode gibt eine Referenz auf ein `MatchCollection`-Objekt zurück, das für alle Fundstellen im Quellstring `Match`-Objekte verwaltet. Diese Objekte interessieren uns für unsere Aufgabenstellung nicht. `WordCount1` liest einfach die `Count`-Eigenschaft aus und gibt damit die Anzahl der Treffer zurück.

```
public static long WordCount1(string source)
{
    return Regex.Matches(source, @"\\w{1,}").Count;
}
```

Listing 28: Sichere (und langsame) Methode zur Ermittlung der Anzahl der Wörter in einem String

Die Verwendung des regulären Ausdrucks und der `MatchCollection` führt dazu, dass die Methode relativ langsam ausgeführt wird (was besonders beim Überprüfen von Strings mit mehreren Kilobyte Zeichen negativ auffällt) und dass viel Speicher benötigt wird (die `Match`-Objekte werden ja schließlich angelegt).

Deshalb habe ich eine weitere Methode `WordCount2` geschrieben. Diese Methode geht den String Zeichen für Zeichen durch und überprüft, ob es sich um ein Nicht-Wort-Zeichen handelt.

Strings werden in einem .NET-Programm als 16-Bit-Unicode-Zeichenketten gespeichert¹. Unicode ist aufgeteilt in einen allgemeinen, lateinischen Bereich und folgende sprachspezifische (z.B. Griechisch und Kyrillisch), Symbol-, mathematische und andere Teiltabellen. Diese Teiltabellen finden Sie auf der Unicode-Webseite an der Adresse www.unicode.org/charts.

`WordCount2` vergleicht den Unicode-Wert jedes Zeichens mit den Werten, die ich als Wort-Zeichen im lateinischen Bereich der Unicode-Tabelle erkannt habe. Unterstützt wird aufgrund der Performance und der Komplexität nur der Bereich bis zum Zeichen 0x2FF (lateinische Zeichen, IPA-Erweiterungen und Abstands-Modifizier-Zeichen).

Mit Hilfe der Tabelle *DerivedCoreProperties* der offiziellen Unicode-Datenbank (www.unicode.org/Public/UNIDATA/DerivedCoreProperties.txt) habe ich dazu zunächst die alphabetischen Zeichen ermittelt und den Zahlbereich und den Unterstrich hinzugefügt.

1. Wenn Sie Unicode-Zeichen im Quellcode verwenden (z.B. über die Betätigung der Alt-Taste mit dem Unicode-Zeichencode im numerischen Tastaturblock), müssen Sie die Quellcodedatei explizit mit Codierung speichern, damit die Unicode-Zeichen in der originalen Codierung bestehen bleiben. Dazu wählen Sie den Befehl `SPEICHERN UNTER` und klicken auf den kleinen Pfeil rechts am `SPEICHERN`-Schalter. Wählen Sie dann den Befehl `MIT CODIERUNG SPEICHERN` und im folgenden Dialog die Codierung `UNICODE (UTF-8 OHNE SIGNATUR) – CODEPAGE 650001`.

Dabei sind die folgenden Bereiche herausgekommen:

- ▶ Zeichen im Bereich von 0x0030 und 0x0039: Ziffern,
- ▶ Zeichen im Bereich von 0x0041 und 0x005A: Normale Großbuchstaben,
- ▶ das Zeichen 0x005F: Unterstrich,
- ▶ Zeichen im Bereich von 0x0061 bis 0x007A: Kleinbuchstaben,
- ▶ die Zeichen 0x00AA, 0x00B5 und 0x00BA: hochgestelltes, unterstrichenes a, Mü-Zeichen, hochgestelltes, unterstrichenes o,
- ▶ Zeichen im Bereich von 0x00C0 bis 0x00D6, 0x00D8 bis 0x00F6, 0x00F8 bis 0x0236: Länderspezifische Sonderzeichen,
- ▶ Zeichen im Bereich von 0x0250 bis 0x02C1, 0x02C6 bis 0x02D1, 0x02E0 bis 0x02E4 und das Zeichen 0x02EE: Wort-Zeichen der IPA-Erweiterung.

Wird das Zeichen als Wort-Zeichen erkannt und die Variable `newWord` ist `false`, beginnt mit diesem Zeichen ein neues Wort und `WordCount2` inkrementiert die Zählvariable `count` und setzt `newWord` auf `false`. Ist das Zeichen kein Wort-Zeichen, setzt `WordCount2` die Variable `newWord` auf `false`. Der Wert von `count` wird schließlich zurückgegeben. Zur Sicherheit wirft `WordCount2` eine Ausnahme, wenn der String ein Zeichen im nicht unterstützten Bereich enthält.

`WordCount2` erfordert den Import des Namensraums `System`.

```
public static long WordCount2(string source)
{
    bool newWord = false;
    long count = 0;

    // Alle Zeichen des Strings durchgehen
    for (int i = 0; i < source.Length; i++)
    {
        char c = source[i];

        // Überprüfen, ob das Zeichen größer ist als das letzte Zeichen im
        // unterstützten (lateinischen) Bereich der Unicode-Tabelle.
        if (c > '\u02FF')
            throw new Exception("WordCount2 unterstützt keine Strings " +
                                "mit Unicode-Zeichen größer 0x02FF");

        // Überprüfen, ob es ein Wort-Zeichen ist
        if ((c >= '\u0030' && c <= '\u0039') ||
            (c >= '\u0041' && c <= '\u005A') ||
            (c == '\u005F') ||
            (c >= '\u0061' && c <= '\u007A') ||
            (c == '\u00AA') ||
            (c == '\u00B5') ||
```

Listing 29: Schnelle und speichersparende Methode zum Zählen der Wörter in einem String

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        (c == '\u00BA') ||
        (c >= '\u00C0' && c <= '\u00D6') ||
        (c >= '\u00D8' && c <= '\u00F6') ||
        (c >= '\u00F8' && c <= '\u0236') ||
        (c >= '\u0250' && c <= '\u02C1') ||
        (c >= '\u02C6' && c <= '\u02D1') ||
        (c >= '\u02E0' && c <= '\u02E4') ||
        (c == '\u02EE'))
    {
        // Wort-Zeichen
        if (newWord == false)
            count++;
        newWord = true;
    }
    else
    {
        // Kein Wort-Zeichen
        newWord = false;
    }
}

return count;
}

```

Listing 29: Schnelle und speichersparende Methode zum Zählen der Wörter in einem String (Forts.)

In einem Performancevergleich (der in das Beispielprogramm auf der Buch-CD integriert ist) habe ich für 10.000 Aufrufe mit dem String »ab cd ef« auf einem Pentium-4-Rechner mit 2,4 GHz für `WordCount1` die (gerundete) Zeit 0,1375 Sekunden und für `WordCount2` die Zeit 0,0009208 Sekunden ermittelt. `WordCount2` ist also deutlich schneller.

21 Alle Wörter eines Strings extrahieren

Das Extrahieren aller Wörter aus einem String ist über den regulären Ausdruck `\w{1,}`, der alle Teile eines Strings ermittelt, die aus einem oder beliebig vielen Wort-Zeichen (Buchstaben, Zahlen und der Unterstrich) bestehen, recht einfach. Die Verwendung dieses Ausdrucks besitzt den Vorteil, dass die Wort-Zeichen aller in der Unicode-Zeichentabelle verwalteter Schriftsprachen korrekt erkannt werden. So können Sie auch z. B. griechische Texte korrekt auswerten.

Über die statische `Matches`-Methode der `Regex`-Klasse (aus dem Namensraum `System.Text.RegularExpressions`) erhalten Sie eine Referenz auf eine `MatchCollection`-Auflistung, die in Form von `Match`-Objekten alle Fundstellen im übergebenen String enthält, die dem ebenfalls übergebenen Muster entsprechen. Die Eigenschaft `Value` der `Match`-Objekte speichert den gefundenen Teilstring.

Die Methode `GetWords` in Listing 30 setzt diese Technik ein. `GetWords` kopiert die Werte der erhaltenen `Match`-Objekte in ein String-Array und gibt dieses zurück.

GetWords erfordert den Import der Namensräume System und System.Text.RegularExpressions.

```
public static string[] GetWords(string source)
{
    // Alle Wörter abfragen
    MatchCollection matches = Regex.Matches(source, @"\\w{1,}");

    // Die MatchCollection in ein String-Array kopieren und zurückgeben
    string[] words = new string[matches.Count];
    for (int i = 0; i < matches.Count; i++)
        words[i] = matches[i].Value;
    return words;
}
```

Listing 30: Methode zum Extrahieren aller Wörter aus einem String

Abbildung 11 zeigt das Ergebnis eines Test-Programms, bei dem der String » Das ist ein [Test-]String« zerlegt wurde.

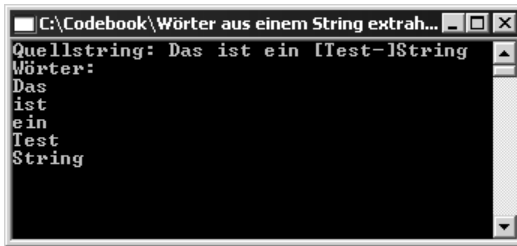


Abbildung 11: Ein Test-Programm hat den String korrekt zerlegt

22 String auf die ersten Wörter bis zu einer Maximallänge kürzen

Oftmals ist es sinnvoll, einen langen String zur Darstellung auf dem Bildschirm so einzukürzen, dass dieser in einem gegebenen Raum dargestellt werden kann. Ein Beispiel dafür sind Internet-Suchmaschinen wie Google, die eine verkürzte Form der Beschreibung der gefundenen Webseiten anzeigen. Dabei ist es aber nicht ideal, den String einfach zu kürzen, da dieser dann mit großer Wahrscheinlichkeit unschön mitten in einem Wort abgetrennt wird.

Die Methode `AbbreviateString` in Listing 31 kürzt einen String so, dass dieser der gegebenen Maximallänge an Zeichen entspricht, aber nicht mitten in einem Wort abgeschnitten wird. Als Wort-Trennzeichen wird nur das Leerzeichen verwendet, um zu erreichen, dass nicht mitten in mit Bindestrichen zusammengesetzten Wörtern oder in Wörtern mit angehängten Klammern oder Ähnlichem gekürzt wird.

`AbbreviateString` splittet den String einfach an den Leerzeichen auf und setzt die einzelnen Teilstrings in einer Schleife so lange zusammen, bis die maximal mögliche Länge erreicht ist.

Bei der maximalen Länge werden das Leerzeichen und die drei Punkte mit eingerechnet, die `AbbreviateString` an einen gekürzten String anhängt. `AbbreviateString` berücksichtigt aber auch die Sonderfälle, dass schon das erste Wort zu lang ist oder dass der String insgesamt kürzer ist als die Maximallänge.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` einbinden.

```
public static string AbbreviateString(string source, int maxCharCount)
{
    string result = "";

    // String am Leerzeichen splitten
    string[] words = source.Split(' ');

    // Die Sonderfälle abhandeln, dass der gesamte String kürzer oder das erste
    // Wort schon zu lang ist
    if (source.Length <= maxCharCount)
        return source;
    if (words.Length > 0 && words[0].Length > maxCharCount)
        return words[0].Substring(0, maxCharCount - 4) + " ...";

    // Die Wörter durchgehen und in das Ergebnis schreiben, bis die
    // Maximallänge erreicht ist
    for (int i = 0; i < words.Length; i++)
    {
        if (result.Length + words[i].Length + 5 > maxCharCount)
            return result + " ...";
        else
            result += ' ' + words[i];
    }

    return source;
}
```

Listing 31: Methode zum Kürzen eines Strings unter Berücksichtigung der Wörter

Listing 32 zeigt einen beispielhaften Aufruf der `AbbreviateString`-Methode.

```
string source = "Die Antwort auf die Frage aller Fragen ist 42";
string result = AbbreviateString(source, 20);
Console.WriteLine("Quellstring: {0}", source);
Console.WriteLine("Gekürzter String: {0}", result);

source = "Formular-Authentifizierung";
result = AbbreviateString(source, 20);
Console.WriteLine("Quellstring: {0}", source);
Console.WriteLine("Gekürzter String: {0}", result);
```

Listing 32: Beispiel-Aufruf der `AbbreviateString`-Methode

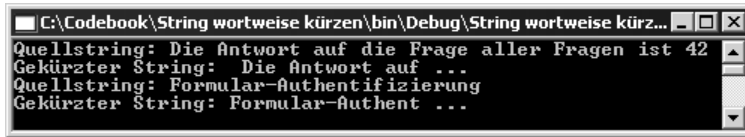


Abbildung 12: Wortgerechtes Kürzen eines Strings auf maximal 20 Zeichen

23 Teilstring nur dann ersetzen, wenn dieser am Anfang eines Strings gespeichert ist

Bei der Entwicklung einiger Methoden für dieses Buch musste ich einen Teilstring in einem String ersetzen, wenn dieser am Anfang des Strings gespeichert war. Ein Beispiel dafür ist das Rezept 107, das zwei Ordner auf einen identischen Inhalt vergleicht.

Dieses Problem habe ich über einen regulären Ausdruck gelöst, der mit einem `^`-Zeichen begann und mit der gesuchten Zeichenkette endete. Das `^`-Zeichen besagt, dass das folgende Muster am Anfang der Zeichenkette stehen muss. Um den am Anfang stehenden Teilstring zu ersetzen, rufen Sie die `Replace`-Methode der `Regex`-Klasse aus dem Namensraum `System.Text.RegularExpressions` auf. Dabei müssen Sie allerdings beachten, dass reguläre Ausdrücke einer Vielzahl an Zeichen eine besondere Bedeutung geben. Der Backslash wird z. B. als Einleitung einer Escape-Sequenz erkannt, runde Klammern werden für Gruppierungen verwendet.

Die Methode `ReplaceLeadingString` in Listing 33 stellt deswegen über die `Escape`-Methode der `Regex`-Klasse allen Sonderzeichen der regulären Ausdrücke im Suchstring einen Backslash voran, der die Bedeutung des Zeichens aufhebt.

`ReplaceLeadingString` ruft dann `Regex.Replace` auf, wobei der Aufruf entsprechend dem Argument `ignoreCase` entweder mit dem Wert `RegexOptions.IgnoreCase` am vierten Argument erfolgt oder ohne diesen Wert. Damit ist (als kleiner Zusatz) auch das Ersetzen ohne Berücksichtigung der Groß-/Kleinschreibung möglich.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Text.RegularExpressions` importieren.

```
public static string ReplaceLeadingString(string source, string find,
    string replacement, bool ignoreCase)
{
    // Muster für die Suche zusammenstellen, dabei den Sonderzeichen im
    // Suchstring einen Backslash voranstellen
    string pattern = "^" + Regex.Escape(find);

    // Muster ersetzen
    if (ignoreCase)
        return Regex.Replace(source, pattern, replacement,
            RegexOptions.IgnoreCase);
    else
```

Listing 33: Ersetzen eines Teilstrings am Anfang eines Strings

```
        return Regex.Replace(source, pattern, replacement);
    }

```

Listing 33: Ersetzen eines Teilstrings am Anfang eines Strings (Forts.)

Dass `ReplaceLeadingString` wunderbar funktioniert, beweist das folgende Testprogramm, bei dem der Basis- und der Suchstring bewusst mit einigen Sonderzeichen der regulären Ausdrücke beginnen:

```
string source = @"^$ {}[] \1 \2 # \w. C:\Codebook\Basics\Strings (Test)";
string find = @"^$ {}[] \1 \2 # \w. C:\Codebook";
string replacement = @"C:\Books\C#\Codebook";
string result = ReplaceLeadingString(source, find, replacement, true);

```

Listing 34: Testprogramm für die `ReplaceLeadingString`-Methode

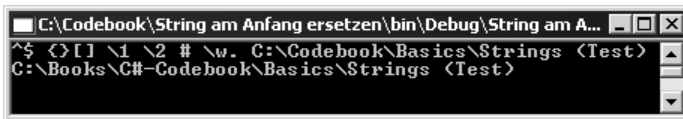


Abbildung 13: Die Ersetzung des Anfangs-Teilstrings war erfolgreich

24 Linken und rechten Teilstring extrahieren

Bei der Stringverarbeitung kommt es immer wieder vor, dass ein linker oder rechter Teilstring extrahiert werden muss. `string`-Objekte bieten zwar eine Methode `Substring`, die das Extrahieren beliebiger Teilstrings erlaubt. Das Auslesen des linken und mehr noch eines rechten Teilstrings ist damit aber für die Praxis ein wenig zu aufwändig. Hinzu kommt, dass `Substring` wenig tolerant ist und die Anfangs- und Endposition sich immer auf eine Position im String beziehen muss.

Wenn Sie nur einzelne String-Extraktionen programmieren und die Performance der Extraktion keine Rolle spielt, können Sie dazu auch die Methoden `Left` und `Right` in Listing 35 verwenden. Diese Methoden erlauben einen einfacheren Aufruf und arbeiten toleranter als `Substring`. Wenn Sie an `count` eine Anzahl Zeichen übergeben, die größer ist als der String selbst, geben beide Methoden einfach den übergebenen String unverändert zurück.

Zum Kompilieren dieser Methoden müssen Sie den Namensraum `System` importieren.

```
/* Methode zum Extrahieren eines linken Teilstrings */
public static string Left(string source, int count)
{
    if (source.Length >= count)
        return source.Substring(0, count);
    else

```

Listing 35: Methoden zum Extrahieren des linken und rechten Teilstrings


```
        return source;
    }

    /* Methode zum Extrahieren eines rechten Teilstrings */
    public static string Right(string source, int count)
    {
        int length = source.Length;
        if (length >= count)
            return source.Substring(length - count, count);
        else
            return source;
    }
}
```

Listing 35: Methoden zum Extrahieren des linken und rechten Teilstrings (Forts.)

Die Performance beider Methoden ist recht gut. In Tabelle 3 finden Sie einen Vergleich mit Substring.

Methode	Zeit für 10.000 Aufrufe in Sekunden (Pentium 4, 2,4 GHz)
<i>Left</i>	0,00112025411050846
Substring(0, <i>count</i>)	0,000840050900323924
<i>Right</i>	0,00112137157096782
Substring(<i>Length</i> – <i>count</i> , <i>count</i>)	0,000844520742161364

Tabelle 3: Performance der Left- und Right-Methoden im Vergleich mit einem direkten Substring-Aufruf

Den Performancetest finden Sie im Beispielprogramm zu diesem Rezept auf der CD.

25 (Ganz-)Zahlen aus einem String extrahieren

Bei der Verarbeitung von Strings kommt es vor, dass Zahlen, die im String neben anderen Zeichenfolgen gespeichert sind, extrahiert werden müssen. Das ist z.B. dann der Fall, wenn Sie eine eingegebene Telefonnummer normalisieren müssen. Anwender geben Telefonnummern in verschiedenen Formaten ein, wobei die einzelnen Nummernteile meist durch Leerzeichen, Bindestriche oder Klammern getrennt werden. Eine solche Eingabe können Sie so zerlegen, dass nur die eingegebenen Zahlen herauskommen, und diese mit vordefinierten Trennzeichen dann wieder zusammensetzen.

Wenn ein String eine oder mehrere Ganzzahlen enthält und zwischen diesen Zahlen beliebige Zeichen gespeichert sind, können Sie diese Zahlen recht einfach über einen regulären Ausdruck extrahieren. Dazu können Sie die `Matches`-Methode der `Regex`-Klasse aus dem Namensraum `System.Text.RegularExpressions` verwenden. Dieser Methode übergeben Sie am ersten Argument den Quellstring und am zweiten das Suchmuster. Zum Suchen aller Zahlen können Sie das Muster `\d{1,}` verwenden. Dieses Muster bedeutet, dass eine Folge von Ziffern mit beliebig vielen Ziffern gesucht wird. `Matches` gibt eine Referenz auf eine Instanz

der Klasse `MatchCollection` zurück, die eine Auflistung von `Match`-Objekten ist. Aus der Eigenschaft `Value` dieser Objekte können Sie den Teilstring auslesen, der dem Muster entspricht.

Die Methode `ExtractNumbers` im folgenden Listing geht auf diese Weise vor um alle Zahlen aus einem String in ein `long`-Array zu kopieren. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Text.RegularExpressions` importieren.

```
public static long[] ExtractNumbers(string source)
{
    // Ganzzahlen über einen regulären Ausdruck extrahieren
    MatchCollection matches = Regex.Matches(source, @"\d{1,}");

    // Das Ergebnis in ein long-Array kopieren
    long[] result = new long[matches.Count];
    for (int i = 0; i < matches.Count; i++)
    {
        result[i] = Convert.ToInt64(matches[i].Value);
    }

    return result;
}
```

Listing 36: Methode zum Extrahieren aller Ganzzahlen aus einem String

26 Überprüfen, ob ein String mit einer Ganzzahl beginnt

Wenn Sie überprüfen wollen, ob ein String mit einer Ganzzahl beginnt, können Sie ähnlich wie im vorhergehenden Rezept einen regulären Ausdruck einsetzen. Das Muster `^\d{1,}` trifft nur für beliebige Ziffernfolgen zu, die am Anfang der Zeichenkette stehen. Über die `Match`-Methode der `Regex`-Klasse können Sie überprüfen, ob ein String dem Muster entspricht. Dieser Methode übergeben Sie am ersten Argument den Quellstring und am zweiten das Muster. `Match` gibt eine Referenz auf ein `Match`-Objekt zurück. Aus dessen Eigenschaft `Success` können Sie auslesen, ob ein Teilstring gefunden wurde, der dem Muster entspricht.

Die Methode `NumberAtStart` in Listing 37 setzt diese Technik ein. `NumberAtStart` gibt im Erfolgsfall die Länge des gefundenen Teilstrings zurück, die aus der Eigenschaft `Length` des `Match`-Objekts ausgelesen wird. Beginnt der übergebene String nicht mit einer Zahl, gibt `NumberAtStart` 0 zurück.

`NumberAtStart` erfordert den Import der Namensräume `System` und `System.Text.RegularExpressions`.

```
public static int NumberAtStart(string source)
{
    // Match-Instanz erzeugen und überprüfen, ob ein Treffer erzielt wurde
```

Listing 37: Methode, die überprüft, ob ein String mit einer Zahl beginnt, und die die Zeichenlänge der Zahl zurückgibt

```
Match match = Regex.Match(source, @"^\d{1,}");
if (match.Success)
    // Die Länge des gefundenen Teilstrings zurückgeben
    return match.Length;
else
    return 0;
}
```

Listing 37: Methode, die überprüft, ob ein String mit einer Zahl beginnt, und die die Zeichenlänge der Zahl zurückgibt (Forts.)

`NumberAtStart` setzt voraus, dass Sie die Namensräume `System` und `System.Text.RegularExpressions` importieren.

Über die Rückgabe der Länge des Strings können Sie die Zahl später sehr einfach aus dem String extrahieren:

```
string source = "1234 abc";
int result = NumberAtStart(source);
if (result > 0)
{
    long number = Convert.ToInt64(source.Substring(0, result));
    Console.WriteLine("{0} beginnt mit einer Zahl: {1}", source, number);
}
else
    Console.WriteLine("{0} beginnt nicht mit einer Zahl", source);
```

Listing 38: Ermitteln, ob ein String mit einer Zahl beginnt, und extrahieren dieser Zahl

27 Zufalls-String berechnen

Die Berechnung einer zufälligen Zeichenkette ist besonders wichtig für automatisch generierte Passwörter, so wie diese von vielen Webseiten generiert werden, die eine Registration erfordern. Das Berechnen eines zufälligen Strings ist recht einfach.

Die Methode `RandomString` erzeugt dazu für jedes Zeichen des Strings eine Zufallszahl im Bereich der Unicode-Codes der normalen Kleinbuchstaben a bis z (97 bis 122) und hängt das jeweilige Zeichen an einen Ergebnisstring an.

Da die `Random`-Klasse keine echten Zufallszahlen erzeugt (siehe Rezept 12), verwendet `RandomString` eine Instanz der Klasse `RNGCryptoServiceProvider`, deren Methode `GetNonZeroBytes` »kryptografisch starke« Zufallszahlen im Bereich von 1 bis 255 erzeugt. Um Zahlen im Bereich von 97 bis 122 zu erhalten, teilt `RandomString` jede ermittelte Zahl durch 9.81 (das entspricht etwa $255 / 26$), addiert 97 hinzu und wandelt das `double`-Ergebnis in einen `byte`-Wert um. Der berechnete Wert wird schließlich in ein Zeichen umgewandelt und an den Ergebnisstring angehängt.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Security.Cryptography` importieren.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
public static string RandomString(int count)
{
    string randomString = "";

    // Echte Zufallszahlen im Bereich von 1 bis 255 erzeugen
    RNGCryptoServiceProvider rngCSP = new RNGCryptoServiceProvider();
    byte[] numbers = new byte[count];
    rngCSP.GetNonZeroBytes(numbers);

    // Die Zahlen so umrechnen, dass Werte zwischen 97 und 122 herauskommen,
    // und die daraus resultierenden Zeichen an den Ergebnisstring anhängen
    for (int i = 0; i < count; i++)
    {
        numbers[i] = (byte)(numbers[i] / 9.81 + 97);
        randomString += (char)numbers[i];
    }

    return randomString;
}
```

Listing 39: Methode zur Erzeugung eines Zufalls-Strings

Abbildung 14 zeigt die Ausgabe eines Programms, das mit Hilfe der `RandomString`-Methode einen Zufallsstring erzeugt hat.



Abbildung 14: Erzeugen eines Zufalls-Strings

28 Konstantennamen einer Aufzählung als String auslesen

In Anwendungen ist es häufig sinnvoll, die symbolischen Konstanten, die eine Aufzählung (Enumeration) zur Verfügung stellt, als String darzustellen. So können Sie dem Anwender z.B. in einer Listbox alle Werte einer Aufzählung zur Auswahl anbieten.

Die Klasse `Enum` bietet mit der statischen Methode `GetNames` eine einfache Möglichkeit, die Konstanten einer Aufzählung als String darzustellen. Dieser Methode übergeben Sie den Typ der Aufzählung und erhalten ein `string`-Array zurück.

Wenn Sie nur einzelne Konstantennamen auslesen wollen, können Sie die `GetName`-Methoden verwenden, der Sie neben dem Typ der Aufzählung am zweiten Argument den Wert übergeben. Das macht allerdings nur dann Sinn, wenn Sie den Wert nicht über die symbolische Konstante, sondern direkt angeben.

Listing 40 zeigt die Konstantennamen für die Aufzählung `System.DayOfWeek`. Die Namen dieser Aufzählung werden in diesem Beispiel im `Load`-Ereignis eines Formulars in eine `ListBox` gelesen. Der Name der Konstante mit dem Wert 0 wird in ein Label ausgegeben.

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // Alle Werte einer Aufzählung als String auslesen
    string[] enumNames = Enum.GetNames(typeof(DayOfWeek));
    for (int i = 0; i < enumNames.Length; i++)
        this.daysListBox.Items.Add(enumNames[i]);

    // Einen bestimmten Wert auslesen
    string name = Enum.GetName(typeof(DayOfWeek), 0);
    this.infoLabel.Text = "Der Name der Konstante mit dem Wert 0 ist " + name;
}
```

Listing 40: Auslesen der Konstantennamen einer Aufzählung

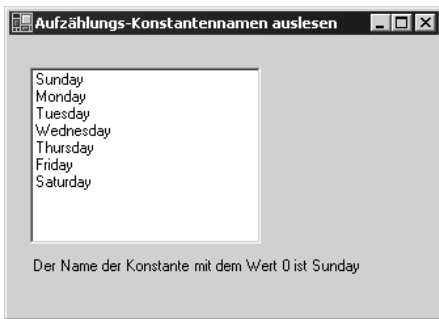


Abbildung 15: Das Programm hat die Konstantennamen der `DayOfWeek`-Aufzählung ausgelesen

29 Strings in Aufzählungswerte konvertieren

Im Rezept 28 habe ich gezeigt, wie Sie die Konstantennamen einer Aufzählung als String auslesen können um diese dem Anwender zur Auswahl anbieten zu können. Etwas problematisch ist allerdings die Auswahl der Anwender-Selektion. Als Problemlösung können Sie in der `ListBox` statt einfachen Strings eigene Objekte für die einzelnen Konstantenwerte verwalten, deren `ToString`-Methode den Namen der Konstante zurückgibt (sodass dieser angezeigt wird) und die in einer separaten Eigenschaft den Wert der Konstante verwalten. Wenn Ihnen das zu aufwändig ist, können Sie aber auch einfach die `Parse`-Methode der `Enum`-Klasse verwenden. Dieser Methode übergeben Sie den Typ der Aufzählung und den String, den Sie umwandeln wollen. Am dritten Argument können Sie mit `true` bestimmen, dass die Groß-/Kleinschreibung nicht berücksichtigt werden soll.

`Parse` gibt den Aufzählungswert als `object` zurück, Sie müssen die Rückgabe also entsprechend umwandeln. Natürlich erzeugt `Parse` eine Ausnahme, wenn der übergebene String keinem Konstantennamen entspricht. Diese Ausnahme sollten Sie abfangen, wenn der String vom Anwender eingegeben wurde.

Zum Kompilieren des Beispiels müssen Sie den Namensraum `System` importieren.

```
string value = "friday";
try
{
    // Konstantenwert auslesen und in den Typ DayOfWeek casten
    DayOfWeek day = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), value, true);

    Console.WriteLine("Wert von '{0}': {1}", value, (int)day);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

Listing 41: Umwandeln eines String in einen Aufzählungswert

30 String in Byte-Array umwandeln

Einige spezielle Techniken des .NET Frameworks arbeiten mit Byte-Arrays, die die einzelnen Zeichen eines Strings verwalten. Dazu gehören z.B. einige Klassen des Namensraums `System.Security.Cryptography`, über deren Methoden Sie Daten ver- und entschlüsseln können, und die Klasse `System.IO.MemoryStream`, die einen Stream im Arbeitsspeicher erzeugt (und die für einige Tricks in diesem Buch eingesetzt wird).

Einen String können Sie nun sehr einfach in ein Byte-Array konvertieren. Dazu verwenden Sie die `GetBytes`-Methode eines `Encoding`-Objekts, der Sie in einer Variante einen String übergeben können. Diese Methode setzt die einzelnen Zeichen des übergebenen Strings in deren zur Codierung passende Byte-Werte um. Sie müssen nun nur noch die richtige Codierung herausfinden und ein entsprechendes `Encoding`-Objekt erzeugen. Wenn Sie, was wohl für die meisten Anwendungsfälle ausreicht, die Standard-8-Bit-Codierung ISO-8859-1 (ISO Latin 1) verwenden wollen, erzeugen Sie dieses Objekt über die statische Methode `GetEncoding` der `Encoding`-Klasse, der Sie den Namen der Codierung übergeben. Listing 42 zeigt, wie Sie aus einem String auf diese Weise ein Byte-Array erzeugen. Das Programm benötigt den Import der Namensräume `System` und `System.Text`.

```
string source = "abc äöü";

// String in byte-Array in der ISO-8859-1-Codierung umwandeln
byte[] result = System.Text.Encoding.GetEncoding(
    "ISO-8859-1").GetBytes(source);

// Byte-Array testweise ausgeben
for (int i = 0; i < result.Length; i++)
    Console.WriteLine("{0}: {1}", result[i], (char)result[i]);
```

Listing 42: Umwandeln eines Strings in ein Byte-Array in der Codierung ISO-8859-1



Abbildung 16: Das Programm hat die Zeichen des Teststrings korrekt in Bytes umgewandelt

Sie müssen bei der Umwandlung allerdings eines beachten: Strings werden in C# intern als 16-Bit-Unicode-Zeichen gespeichert. Wurde der String z. B. aus einer Unicode-Textdatei eingelesen, kann es sein, dass Zeichen gespeichert sind, deren Wert größer ist als 0x00FF (der größte 8-Bit-Wert). Eine Konvertierung mit einer 8-Bit-Codierung führt dann dazu, dass alle für diese Codierung unbekannten Zeichen in ein Fragezeichen (mit dem Code 0x003F) konvertiert werden. Dasselbe gilt natürlich auch, wenn Sie einen String, der lediglich 8-Bit-Sonderzeichen wie unsere Umlaute enthält, in die 7-Bit-ASCII-Codierung umwandeln (z. B. über das `Encoding`-Objekt, das Sie über die `ASCII`-Eigenschaft der `Encoding`-Klasse erreichen). Eine Rück-Umwandlung des Byte-Arrays führt dann natürlich zu einem anderen String als dem ursprünglichen. Das Beispiel in Listing 43 zeigt, was ich meine.

```
// String in byte-Array in der ASCII-Codierung umwandeln
string source = "abc-äöü";
byte[] result = System.Text.Encoding.ASCII.GetBytes(source);
Console.WriteLine("Originaler String: {0}", source);

// Bytes ausgeben
for (int i = 0; i < result.Length; i++)
    Console.WriteLine("{0}: {1} ", result[i], (char)result[i]);

// String zurückkonvertieren
source = System.Text.Encoding.ASCII.GetString(result);
Console.WriteLine("Zurückkonvertierter String: {0}", source);
```

Listing 43: Umwandeln eines Unicode-Strings mit Zeichen größer 0x00ff (dem größten 7-Bit-ASCII-Wert) in ein Byte-Array und wieder zurück

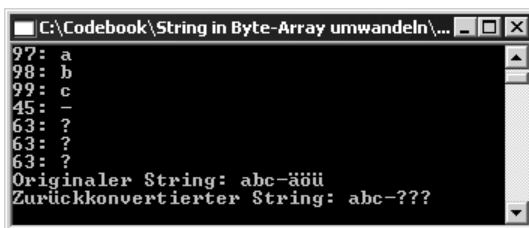


Abbildung 17: Die Umlaute wurden in ein Fragezeichen umgewandelt

Wenn Sie Strings mit Zeichen größer als 0x00FF umwandeln wollen, müssen Sie die Unicode-Codierung verwenden:

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
string source = "abc-äöü \u0400\u0410"; // String mit kyrillischen Zeichen
byte[] result = System.Text.Encoding.Unicode.GetBytes(source);
for (int i = 0; i < result.Length; i++)
    Console.WriteLine("{0}: {1} ", result[i], (char)result[i]);
```

Listing 44: Umwandeln eines Strings in ein Byte-Array mit der Unicode-Codierung

Wenn Sie die Unicode-Codierung verwenden, werden pro Zeichen zwei Bytes gespeichert. Das erste Byte verwaltet den niedrigen und das zweite den hohen 8-Bit-Wert eines Zeichens. Bei Zeichen im 8-Bit-Bereich ist das zweite Byte natürlich immer mit 0 belegt. In Abbildung 18 sehen Sie aber, dass die kyrillischen Zeichen beide Bytes belegen.

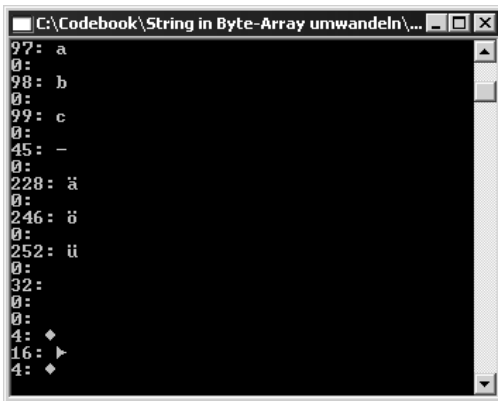


Abbildung 18: Ein Byte-Array in der Unicode-Codierung (die Zeichen-Ausgabe der kyrillischen Zeichen entspricht in der Abbildung den einzelnen Bytes dieser Zeichen)

31 Byte-Array in String lesen

Im Rezept 30 habe ich beschrieben, wie Sie einen String in ein Byte-Array umwandeln. Der Weg zurück ist natürlich auch möglich (und wird im genannten Rezept bereits eingesetzt 😊).

Dazu verwenden Sie die `GetString`-Methode eines zur im Byte-Array verwendeten Codierung passenden `Encoding`-Objekts. Grundlagen dazu habe ich bereits im genannten Rezept erläutert. Sie müssen beim Umwandeln lediglich beachten, dass Sie eine passende Codierung einsetzen.

Das Beispiel in Listing 45 erzeugt ein Unicode-Byte-Array und wandelt dies in einen String um. Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.Text` einbinden.

```
byte[] unicodeCharCodes = {0x41, 0, 0x42, 0, 0x20, 0, 0xE4, 0, 0xF6, 0};

// Byte-Array in String umwandeln
```

Listing 45: Umwandeln eines Byte-Arrays in einen String

```
string result = Encoding.Unicode.GetString(unicodeCharCodes);  
  
Console.WriteLine(result);
```

Listing 45: Umwandeln eines Byte-Arrays in einen String (Forts.)

Das Ergebnis dieses Programms ist der String »AB äö«. Würden Sie statt des Unicode-Encoding-Objekts eines für die ISO-8859-1-Codierung verwenden, würde der String bei einer Ausgabe folgendermaßen aussehen: »A B ä ö«. Die zusätzlichen Leerzeichen resultieren aus den 0-Bytes der Unicode-Daten, die bei einer Ausgabe als unsichtbares Zeichen erscheinen.

32 Arrays eigener Objekte sortieren

Arrays können Sie über die statische `Sort`-Methode der `Array`-Klasse sortieren:

```
byte[] numbers = {10, 1, 3, 19, 33, 5, 7};  
Array.Sort(numbers);
```

Die `ArrayList`-Klasse aus dem Namensraum `System.Collections` bietet ebenfalls eine (Instanz-)Methode `Sort` zur Sortierung der aufgelisteten Wert- oder Referenztypen:

```
ArrayList names = new ArrayList();  
names.Add("Zaphod");  
names.Add("Ford");  
names.Add("Trillian");  
names.Add("Arthur");  
names.Sort();
```

`Sort` verwendet laut der Dokumentation des .NET Frameworks 1.1 den besonders schnellen Quicksort-Algorithmus, der wie jeder andere Sortier-Algorithmus jeweils zwei Elemente gegeneinander vergleicht und gegebenenfalls vertauscht, dabei aber das Array jeweils in Hälften aufteilt und diese durchgeht.

Mit den Standard-Typen und Instanzen einiger Klassen wie z.B. `String` und `DateTime` ist das Sortieren kein Problem. Wenn Sie jedoch Instanzen eigener Strukturen oder Klassen in einem Array oder einer `ArrayList`-Auflistung sortieren wollen, müssen Sie in diesen die `Comparable`-Schnittstelle implementieren. `Comparable` besitzt lediglich die Methode `CompareTo`. Dieser Methode übergibt `Sort` das andere Objekt, das mit dem aktuellen verglichen werden soll. `CompareTo` muss die folgenden Werte zurückgeben, wenn Sie aufsteigend sortieren wollen:

- ▶ -1, wenn das aktuelle Objekt kleiner ist als das übergebene,
- ▶ 0, wenn beide Objekte gleich groß sind,
- ▶ 1, wenn das aktuelle Objekt größer ist als das übergebene.

Die Klasse `Person` in Listing 46 zeigt ein Beispiel für die Implementierung der `CompareTo`-Methode. In dieser Klasse vergleicht `CompareTo` den Nach- und den Vornamen der Person um zu ermitteln, welche Instanz größer ist.

```
public class Person: IComparable
{
    public string FirstName;
    public string LastName;
    public Person(string firstName, string lastName)
    {
        this.FirstName = firstName;
        this.LastName = lastName;
    }

    /* Implementierung der CompareTo-Methode */
    public int CompareTo(object o)
    {
        Person otherPerson = (Person)o;
        string firstName1 = this.FirstName.ToLower();
        string firstName2 = otherPerson.FirstName.ToLower();
        string lastName1 = this.LastName.ToLower();
        string lastName2 = otherPerson.LastName.ToLower();

        if (lastName1 == lastName2 && firstName1 == firstName2)
            // Beide Instanzen sind gleich
            return 0;
        else if (lastName1.CompareTo(lastName2) < 0 ||
            (lastName1 == lastName2 &&
            firstName1.CompareTo(firstName2) < 0))
            // Diese Instanz ist kleiner als die andere
            return -1;
        else
            // Diese Instanz ist größer als die andere
            return 1;
    }
}
```

Listing 46: Klasse, die die `IComparable`-Schnittstelle für ein korrektes Sortieren implementiert

Wenn Sie nun ein Array von `Person`-Instanzen erzeugen und sortieren, erfolgt die Sortierung nach der `CompareTo`-Methode:

```
Person[] persons = new Person[4];
persons[0] = new Person("Arthur", "Miller");
persons[1] = new Person("John", "Dent");
persons[2] = new Person("Arthur", "Dent");
persons[3] = new Person("Tricia", "McMillan");

Array.Sort(persons);
```

```
for (int i = 0; i < persons.Length; i++)
    Console.WriteLine(persons[i].FirstName + " " + persons[i].LastName);
```



Abbildung 19: Sortieren eigener Objekte

33 StringCollection (und andere Auflistungen) sortieren

Die für das Speichern von Stringlisten sehr interessante Klasse `StringCollection` aus dem Namensraum `System.Collections.Specialized` besitzt keine Methode zum Sortieren. Wenn Sie eine solche Auflistung sortieren wollen, können Sie ihre Werte in eine `ArrayList`-Instanz oder in ein Array kopieren, dort sortieren und wieder zurückkopieren, wie dies die Methode `Sort` in Listing 47 ausführt.

Sort benötigt den Import der Namensräume `System` und `System.Collections.Specialized`.

```
public static void Sort(StringCollection col)
{
    // StringCollection in ein Array kopieren und dieses sortieren
    string[] values = new string[col.Count];
    col.CopyTo(values, 0);
    Array.Sort(values);

    // Das Ergebnis wieder zurückkopieren
    for (int i = 0; i < values.Length; i++)
        col[i] = values[i];
}
```

Listing 47: Methode zum Sortieren einer `StringCollection`

Dieses Verfahren funktioniert natürlich auch für andere Auflistungen, die keine eigene `Sort`-Methode besitzen und prinzipiell sortierbar sind (was für Auflistungen gilt, die die Schnittstelle `IList` implementieren). Auflistungen, die die Schnittstelle `IDictionary` implementieren (z.B. `Hashtable`-Instanzen), sind prinzipiell nicht sortierbar (es sei denn, es handelt sich um Auflistungen wie `SortedList`, die ihren Inhalt automatisch sortieren). Bei solchen Auflistungen können Sie aus den Daten aber über die `Values`-Eigenschaft eine Auflistung der Werte auslesen, die Sie dann wie in Listing 47 über die `CopyTo`-Methode in ein Array kopieren und ebenfalls sortieren können.

Listing 48 zeigt eine Beispielanwendung der `Sort`-Methode, Abbildung 20 das Ergebnis dieses Programms.

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// StringCollection erzeugen und füllen
StringCollection col = new StringCollection();
col.Add("Zaphod");
col.Add("Trillian");
col.Add("arthur");
col.Add("ford");
col.Add("Marvin");
col.Add("Slartibartfaß");

Console.WriteLine("Original-Reihenfolge:");
for (int i = 0; i < col.Count; i++)
    Console.WriteLine(col[i]);

// StringCollection über die eigene Sort-Methode sortieren
Sort(col);

Console.WriteLine();
Console.WriteLine("Sortiert:");
for (int i = 0; i < col.Count; i++)
    Console.WriteLine(col[i]);
```

Listing 48: Beispielprogramm zur Anwendung der Sort-Methode

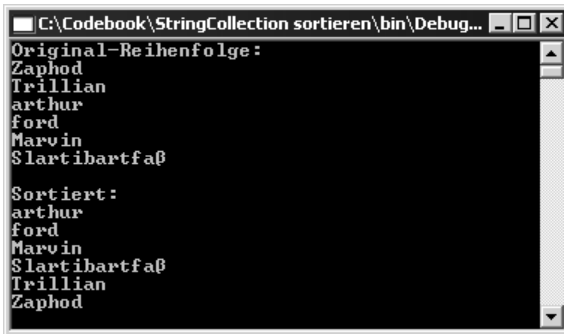


Abbildung 20: Das Programm hat die StringCollection sortiert

Erstaunlicherweise benötigt ein solches Sortieren trotz des doppelten Kopierens sehr wenig Zeit. In einem Performancetest, der einen (selbst entwickelten) Quicksort-Algorithmus, einen (im Internet recherchierten) Heapsort-Algorithmus² und die `Sort`-Methode miteinander verglich, ergaben sich für das Sortieren von 1000 zufällig erzeugten Strings auf einem Pentium-4-Rechner mit 2,4 GHz die in Tabelle 4 dargestellten mittleren Zeitwerte. Das Programm finden Sie neben dem Beispiel zu diesem Rezept.

2. Der zum Vergleich verwendete Heapsort-Algorithmus, der im Internet an vielen Stellen zu finden ist, ist allerdings leicht fehlerhaft, denn das erste Element wird bei der Sortierung nicht berücksichtigt.

Sortiervverfahren	Zeit in Sekunden
Quicksort	3,70421305873605
Heapsort	0,00338283217559774
Sort	0,00274597283546745

Tabelle 4: Vergleich verschiedener Sortiervverfahren

Das auf wundersame Weise sogar bessere Ergebnis der Sort-Methode beweist, dass sich die Implementierung eines speziellen Sortieralgorithmus nicht lohnt.

34 Arrays, ArrayList- und andere Auflistungen durchsuchen

In Arrays können Sie über die `BinarySearch`-Methode der `Array`-Klasse suchen. Eine `ArrayList`-Instanz besitzt ebenfalls eine solche Methode. `BinarySearch` sucht nach dem übergebenen Objekt und gibt den Index des gefundenen Elements zurück. Wird kein Element gefunden, das zum Suchobjekt passt, gibt diese Methode -1 zurück.

`BinarySearch` führt eine effiziente binäre Suche in den Elementen aus, was allerdings für eine optimale Performance voraussetzt, dass die Liste sortiert ist. Sie sollten das Array bzw. die Auflistung also zuvor gegebenenfalls sortieren.

Das Suchen nach Werttypen funktioniert immer ohne Probleme. Beim Suchen nach Referenztypen ist eine wichtige Voraussetzung für die korrekte Ausführung der `BinarySearch`-Methode, dass die Klasse die `CompareTo`-Methode der `IComparable`-Schnittstelle implementiert. Diese Methode wird von `BinarySearch` aufgerufen, um die Objekte in der Liste mit dem Vergleichsobjekt zu vergleichen. Näheres dazu finden Sie im Rezept 32.

Listing 49 zeigt eine Klasse, die Kreisdaten speichert und das Sortieren/Suchen nach dem Radius ermöglicht.

```
public class Circle: IComparable
{
    public int x;
    public int y;
    public int Radius;
    public Circle(int x, int y, int radius)
    {
        this.x = x;
        this.y = y;
        this.Radius = radius;
    }

    /* Implementierung der CompareTo-Methode */
    public int CompareTo(object o)
    {
```

Listing 49: Klasse zur Speicherung von Kreisdaten

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
        Circle otherCircle = (Circle)o;
        return this.Radius.CompareTo(otherCircle.Radius);
    }
}
```

Listing 49: Klasse zur Speicherung von Kreisdaten (Forts.)

Die `CompareTo`-Methode ist in diesem Beispiel sehr einfach, weil zum Vergleich die gleichnamige Methode des `int`-Typs aufgerufen werden kann.

Wenn Sie nun Instanzen dieser Klasse erzeugen, können Sie nach einem Kreis mit einem bestimmten Radius suchen. Der `BinarySearch`-Methode müssen Sie dazu eine neue `Circle`-Instanz mit dem gesuchten Radius übergeben. Vor dem Suchen sollten Sie sortieren, was allerdings nur dann sinnvoll ist, wenn das Array bzw. die Auflistung sehr viele Objekte verwaltet.

```
// Array von Circle-Objekten erzeugen
Circle[] circles = new Circle[4];
circles[0] = new Circle(10, 20, 100);
circles[1] = new Circle(15, 25, 80);
circles[2] = new Circle(5, 10, 120);
circles[3] = new Circle(35, 45, 70);

// Nach einem Kreis mit dem Radius 120 suchen
Array.Sort(circles);
int index = Array.BinarySearch(circles, new Circle(0, 0, 120));
if (index > -1)
    Console.WriteLine("Gefunden: Radius: {0}, x: {1}, y: {2}", circles[index].Radius,
        circles[index].x, circles[index].y);
else
    Console.WriteLine("Nicht gefunden");
```

Listing 50: Suchen in einem Array

Das Suchen in Auflistungen ist zunächst nur mit `ArrayList`-Instanzen möglich. Wenn Sie in anderen Auflistungen suchen wollen, ist das wie beim Sortieren prinzipiell nur mit Auflistungen möglich, die die `IList`-Schnittstelle implementieren (und deren Elemente folglich über einen `int`-Index angesprochen werden können). Dazu können Sie die Elemente der Auflistung ähnlich dem Rezept 33 in ein Array kopieren und suchen. Alternativ können Sie die Auflistung natürlich auch einfach sequentiell durchsuchen, bei großen Auflistungen kann das aber so viel Zeit in Anspruch nehmen, dass der Zeitverlust durch das Umkopieren nicht ins Gewicht fällt. Beachten Sie dann aber, dass der Index der Elemente im sortierten Array nicht mehr mit dem Index der Elemente in der Auflistung korrespondiert.

35 Typisierte Auflistungen erzeugen

Bei der Arbeit mit Objekten, die in einer Auflistung verwaltet werden sollen, ist es immer wieder störend, dass die Standard-Auflistungen des .NET Frameworks object-Referenzen verwalten, die beim Zugriff in den korrekten Objekttyp umgewandelt werden müssen.

Viel einfacher und eleganter ist hingegen eine Auflistung, deren Eigenschaften, Methoden und Indizierer bereits mit den gewünschten Typen arbeiten. Solche Auflistungen können Sie erzeugen, indem Sie eine Klasse entwickeln, die die entsprechenden Schnittstellen (z.B. `IEnumerable`, `IList`, `ICollection`) implementiert. Das macht aber viel Arbeit und erfordert einige Tricks (wie z.B. dass die generischen Schnittstellen über private Methoden implementiert werden, die von typsicheren öffentlichen Methoden aufgerufen werden).

Wesentlich einfacher ist die Ableitung einer Auflistungsklasse von der Basisklasse `CollectionBase` oder `DictionaryBase` aus dem Namensraum `System.Collection`. Die neue Version 2.0 von C# wird für solche Zwecke übrigens wesentlich geeignetere Klassen-Templates unterstützen, die in ähnlicher Form auch in C++ verwendet werden. Aber bis zum Release von C# 2.0 wird es noch eine Weile dauern. Also leiten Sie Ihre eigenen Auflistungen von den dafür vorgesehenen Basisklassen ab.

Für das Beispiel habe ich zunächst eine Klasse `Person` geschrieben, deren Instanzen aufgelistet werden sollen. `Person` implementiert neben einigen Eigenschaften und dem Konstruktor die `CompareTo`-Methode der `IComparable`-Schnittstelle, damit das Suchen und Sortieren möglich wird (siehe Rezepte 32 und 34). Außerdem wird die `ToString`-Methode überschrieben, damit die Ausgabe der Daten des Objekts vereinfacht wird. Das Beispiel benötigt den Import der Namensräume `System` und `System.Collections`.

```
public class Person: IComparable
{
    public int Id;
    public string FirstName;
    public string LastName;

    public Person(int id, string firstName, string lastName)
    {
        this.Id = id;
        this.LastName = lastName;
        this.FirstName = firstName;
    }

    public override string ToString()
    {
        return this.Id + " " + this.FirstName + " " + this.LastName;
    }

    public int CompareTo(object o)
```

Listing 51: Klasse, deren Instanzen aufgelistet werden sollen

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
{  
    return this.Id.CompareTo(((Person)o).Id);  
}  
}
```

Listing 51: Klasse, deren Instanzen aufgelistet werden sollen (Forts.)

Die Klasse `PersonCollection` in Listing 52 implementiert eine `Collection`-Auflistung von `Person`-Objekten. Eine solche Auflistung ermöglicht u.a. das Durchgehen der Liste mit `foreach` und den Zugriff auf die Elemente über einen `int`-Index.

`PersonCollection` wird von der Klasse `CollectionBase` abgeleitet, die die Basis für eine `Collection` bildet. `CollectionBase` schließt dazu eine `ArrayList`-Auflistung in der geschützten Eigenschaft `InnerList` ein, die Sie für das Speichern der Objektreferenzen verwenden. Einige allgemeine Methoden und Eigenschaften sind in `CollectionBase` bereits öffentlich vorhanden (wobei ich die von `object` geerbten nicht mit aufliste):

- ▶ `Count`: Anzahl der gespeicherten Elemente,
- ▶ `Clear`: Löschen aller Elemente,
- ▶ `RemoveAt`: Entfernen eines Elements am angegebenen Index.

Die Möglichkeit des Durchgehens der Liste mit `foreach` oder einem `Enumerator` besteht ebenfalls bereits.

`CollectionBase` implementiert die Schnittstellen `IEnumerable`, `IList` und `ICollection`, wobei einige der Methoden und Eigenschaften dieser Schnittstellen privat implementiert sind (sodass die Forderung der Schnittstelle erfüllt ist, die Methoden aber nicht nach außen zugänglich sind). Der Grund dafür ist, dass einige dieser Elemente mit dem generischen `object`-Typ arbeiten und andere nicht veröffentlicht wurden, weil sie für die Implementierung einer eigenen Auflistung u.U. nicht erwünscht sind.

Sie können diese privat implementierten Methoden nun in Ihrer Klasse öffentlich und vor allen Dingen typsicher implementieren. Die wichtigen Elemente hierbei sind die Methoden `CopyTo`, `Add`, `Contains`, `IndexOf`, `Insert`, `Remove` und der `Indizierer`. Natürlich können Sie auch zusätzliche Methoden implementieren, wobei Sie auf vorhandene Funktionalität der `ArrayList`-Instanz zurückgreifen können. Interessant sind hier die Methoden `Sort` und `BinarySearch`.

Listing 52 zeigt dies für eine Auflistung von `Person`-Objekten, deren Klasse Sie mit einer Suchen-/Ersetzen-Aktion und einer kleinen Änderung der `BinarySearch`-Methode problemlos in eine Auflistung anderer Objekte umwandeln können. Die meisten Elemente dieser Klasse leiten den Aufruf einfach an die entsprechende Methode der geschützten `ArrayList`-Instanz weiter, sind aber typsicher implementiert.

Der Konstruktor ist in zwei Varianten implementiert. Der zweiten Variante können Sie die Initialkapazität der Liste übergeben, für den Fall, dass Sie wissen, wie viele Objekte gespeichert werden sollen und um dann zu erreichen, dass die Liste nicht allzu häufig vergrößert

werden muss. Da die Basisklasse nur einen parameterlosen Konstruktor implementiert, setzt der Konstruktor die Eigenschaft `Capacity` der inneren Liste³.

Die `BinarySearch`-Methode habe ich gegenüber dem Original ein wenig abgeändert, sodass die `Id` der `Person` übergeben werden kann. `BinarySearch` erzeugt mit der `Id` ein `Person`-Objekt, weil die entsprechende `ArrayList`-Methode dieses benötigt um den Vergleich mit den gespeicherten Objekten ausführen zu können (der ja über die `CompareTo`-Methode der `Comparable`-Schnittstelle der `Person`-Instanzen erfolgt). Anders als die `BinarySearch`-Methode der `ArrayList`-Klasse gibt meine Variante eine Referenz auf das gefundene `Person`-Objekt zurück oder `null`, was die Arbeit mit dieser Methode erleichtert.

```
public class PersonCollection: CollectionBase
{
    /* Konstruktoren */
    public PersonCollection()
    {
    }

    public PersonCollection(int capacity)
    {
        this.InnerList.Capacity = capacity;
    }

    /* Add-Methode */
    public void Add(Person p)
    {
        base.InnerList.Add(p);
    }

    /* Remove-Methode */
    public void Remove(Person p)
    {
        base.InnerList.Remove(p);
    }

    /* Indizierer */
    public Person this[int index]
    {
        get {return (Person)base.InnerList[index];}
        set {base.InnerList[index] = value;}
    }

    /* CopyTo-Methode */
    public void CopyTo(Person[] persons, int startIndex)
    {

```

Listing 52: Typisierte Auflistung vom Typ `Collection` für `Person`-Objekte

-
3. Ganz optimal ist das nicht, da die innere `ArrayList`-Instanz beim ersten Zugriff auf die `InnerList`-Eigenschaft erzeugt wird (was ich über den Reflektor von Lutz Röder herausgefunden habe) und in diesem Fall bereits mit der Default-Initialkapazität initialisiert wurde.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        base.InnerList.CopyTo(persons, startIndex);
    }

    /* Contains-Methode */
    public bool Contains(Person p)
    {
        return base.InnerList.Contains(p);
    }

    /* Sort-Methoden */
    public void Sort()
    {
        base.InnerList.Sort();
    }

    public void Sort(IComparer comparer)
    {
        base.InnerList.Sort(comparer);
    }

    public void Sort(int index, int count, IComparer comparer)
    {
        base.InnerList.Sort(index, count, comparer);
    }

    /* BinarySearch-Methode */
    public Person BinarySearch(int Id)
    {
        int index = base.InnerList.BinarySearch(new Person(Id, null, null));
        if (index > -1)
            return (Person)base.InnerList[index];
        else
            return null;
    }
}

```

Listing 52: Typisierte Auflistung vom Typ Collection für Person-Objekte (Forts.)

Zur Erzeugung einer Auflistung vom Typ Hashtable leiten Sie Ihre Klasse von `DictionaryBase` ab. Eine solche Auflistung ermöglicht u. a. den Zugriff auf die Elemente über einen Schlüssel (dessen Hashcode mit den gespeicherten Elementen assoziiert wird) und das Auslesen der gespeicherten Werte und Schlüssel als Collection-Auflistung.

`DictionaryBase` implementiert die Schnittstellen `IEnumerable`, `ICollection` und `IDictionary`, wobei wie bei `CollectionBase` einige der Elemente dieser Schnittstellen privat implementiert sind. Die folgenden für Auflistungen wichtigen Elemente sind aber bereits öffentlich vorhanden: `Count`, `Clear`, `CopyTo` und `GetEnumerator`. Alle weiteren können bzw. müssen Sie selbst implementieren.

`DictionaryBase` schließt eine Instanz der Klasse `Hashtable` in der geschützten Eigenschaft `InnerHashtable` ein. An diese Instanz geben Sie die Aufrufe Ihrer Methoden weiter. Sie sollten

wie in der Klasse `PersonHashtable` in Listing 53 die Methoden `Add`, `Contains`, `Remove`, `CopyTo`, den Indizierer, die `Values`-Eigenschaft und die `Keys`-Eigenschaft implementieren, damit Ihre `Hashtable`-Auflistung flexibel verwendet werden kann. Die meisten dieser Elemente habe ich zwar typsicher implementiert, aber den Aufruf einfach an die innere `Hashtable`-Instanz weitergeleitet. Die `CopyTo`-Methode arbeitet etwas anders als bei einer normalen `Hashtable`. Bei dieser kopiert `CopyTo` Referenzen auf Instanzen der in der Auflistung verwalteten `Dictionary-Entry`-Instanzen in das Array. Meine Variante kopiert aber einfach Referenzen auf die verwalteten `Person`-Instanzen, was die Arbeit mit dem Array erheblich vereinfacht.

Ein Konstruktor ist übrigens nicht implementiert. Sinnvoll wäre dieser zur Definition der Initialkapazität der inneren `Hashtable`-Instanz. Leider besitzt `DictionaryBase` nur einen parameterlosen Konstruktor. Daneben erlaubt die `Hashtable`-Instanz nicht das Definieren der Kapazität (außer im Konstruktor, aber an den kommen wir nicht heran).

```
public class PersonHashtable: DictionaryBase
{
    /* Add-Methode */
    public void Add(Person p)
    {
        base.InnerHashtable.Add(p.Id, p);
    }

    /* Remove-Methode */
    public void Remove(int id)
    {
        base.InnerHashtable.Remove(id);
    }

    /* Indizierer */
    public Person this[int id]
    {
        get {return (Person)base.InnerHashtable[id];}
        set {base.InnerHashtable[id] = value;}
    }

    /* Contains-Methode */
    public bool Contains(int id)
    {
        return base.InnerHashtable.Contains(id);
    }

    /* Values-Eigenschaft */
    public ICollection Values
    {
        get {return base.InnerHashtable.Values;}
    }

    /* Keys-Eigenschaft */
    public ICollection Keys
```

Listing 53: Typsichere `Hashtable`-Auflistung

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    {
        get {return base.InnerHashtable.Keys;}
    }

    /* CopyValues */
    public void CopyTo(Person[] persons, int startIndex)
    {
        this.InnerHashtable.Values.CopyTo(persons, startIndex);
    }
}

```

Listing 53: Typsichere Hashtable-Auflistung (Forts.)

Listing 54 zeigt eine Beispiel-Anwendung der `PersonCollection`-Klasse. Auf die Demonstration der `PersonHashtable`-Klasse habe ich aus Platzgründen verzichtet. Sie finden diese aber im Beispiel zu diesem Rezept auf der Buch-CD.

```

// Collection mit Person-Objekten erzeugen
PersonCollection pc = new PersonCollection();
pc.Add(new Person(1003, "Tricia", "McMillan"));
pc.Add(new Person(1001, "Zaphod", "Beeblebrox"));
pc.Add(new Person(1004, "Arthur", "Dent"));
Person ford = new Person(1002, "Ford", "Prefect");
pc.Add(ford);

// Collection sortieren
pc.Sort();

// Collection durchgehen
Console.WriteLine("Sortierte Auflistung:");
foreach (Person p in pc)
    Console.WriteLine(p.ToString());
Console.WriteLine();

// Ermitteln, ob in der Collection ein bestimmtes Objekt existiert
if (pc.Contains(ford))
    Console.WriteLine("Ford existiert in der Auflistung");
else
    Console.WriteLine("Ford existiert nicht in der Auflistung");
Console.WriteLine();

// Nach einem Objekt mit einer Id suchen
Person person = pc.BinarySearch(1003);
if (person != null)
    Console.WriteLine("Gefunden: {0}", person.ToString());
else
    Console.WriteLine("Nicht gefunden");
Console.WriteLine();

```

Listing 54: Beispiel-Anwendung der `PersonCollection`-Klasse

```
// Ein Objekt an einem bestimmten Index löschen
pc.RemoveAt(2);

// Ein bestimmtes Objekt löschen
pc.Remove(ford);

// Collection durchgehen
Console.WriteLine("Auflistung nach dem Löschen:");
for (int i = 0; i < pc.Count; i++)
    Console.WriteLine(pc[i].ToString());
Console.WriteLine();
```

Listing 54: Beispiel-Anwendung der PersonCollection-Klasse (Forts.)

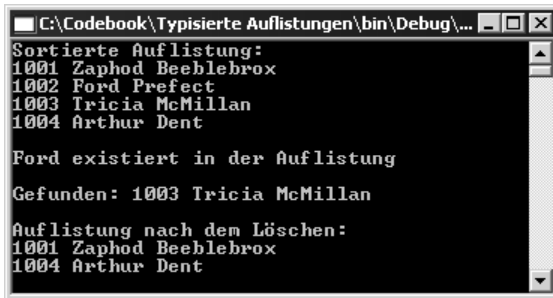


Abbildung 21: Die Beispiel-Anwendung in Aktion

36 Das Programm für eine definierte Zeit anhalten

Manchmal ist es sinnvoll, ein Programm für eine kurze Zeit anzuhalten. Wenn Sie z.B. in einer Schleife eine Berechnung ausführen und das jeweilige Ergebnis in einem Label auf einem Windows-Formular ausgeben wollen, ist die Berechnung meist so schnell, dass Windows mit der Aktualisierung des Labels nicht nachkommt. Wenn Sie allerdings eine kleine Pause einbauen, erfolgt die Ausgabe in den durch die Pause bestimmten Abständen. Eine andere (wenn auch nicht allzu galante) Einsatzmöglichkeit ist das Pausieren eines Info- oder Splash-Formulars beim Start der Anwendung, damit der Anwender genügend »Zeit« hat, die dort ausgegebenen Informationen zu lesen.

Eine definierte Pause können Sie einfach über die statische `Sleep`-Methode der `Thread`-Klasse aus dem Namensraum `System.Threading` erzeugen. Dazu übergeben Sie ein `TimeSpan`-Objekt oder einfach einen `int`-Wert mit der Angabe der Pausenzeit in Millisekunden:

```
Thread.Sleep(1000);
```

`Sleep` hält den aktuellen Thread an (also den Thread, in dem die aktuelle Methode gerade ausgeführt wird). Wenn Sie keine speziellen Threads einsetzen, werden Ihre Methoden im Hauptthread der Anwendung ausgeführt. `Sleep` hält dann also die gesamte Anwendung an. Dieses Pausieren bewirkt zwar, dass der Anwender mit Windows problemlos weiterarbeiten

kann, das Programm kann aber innerhalb der Pause nicht weiter bedient werden. Sogar das Verschieben des Fensters ist innerhalb der Pause nicht möglich. Für viele Anwendungen ist dieser Effekt eigentlich nicht geeignet, da dem Anwender das Verschieben des Formulars und oft auch ein Bedienen erlaubt sein sollten. Um dieses Problem zu lösen, müssen Sie einen separaten Thread für die eigentliche Aufgabe einsetzen, was ich hier aber nicht weiter behandle.

Datum und Zeit

37 Aktuelles Datum und aktuelle Uhrzeit ermitteln

Das aktuelle Systemdatum (inklusive Uhrzeit) erhalten Sie über die `Now`-Eigenschaft der `DateTime`-Struktur (aus dem Namensraum `System`):

```
// Aktuelles Datum ermitteln
DateTime now = DateTime.Now;

// Datum und Zeit ausgeben
Console.WriteLine("Datum: {0}", now.ToShortDateString());
Console.WriteLine("Zeit: {0}", now.ToLongTimeString());
```

Listing 55: Ermitteln des aktuellen Systemdatums inklusive der Zeit



Abbildung 22: Ermitteln des aktuellen Systemdatums und der aktuellen Systemzeit

38 Datumswerte formatieren

Datumswerte können Sie über verschiedene Methoden eines `DateTime`-Objekts im Format der aktuell eingestellten Kultur formatieren.

- ▶ `ToString` gibt das Datum in einer kurzen Datumsform mit einer kurzen Zeitangabe zurück,
- ▶ `ToShortDateString` gibt den Datumsanteil in einer kurzen Form zurück (z.B. »19.05.2003«),
- ▶ `ToLongDateString` gibt den Datumsanteil in einer langen Form zurück (z.B. »Montag, 19 Mai 2003«),
- ▶ `ToShortTimeString` gibt den Zeitanteil in einer kurzen Form zurück (z.B. »16:14«),
- ▶ `ToLongTimeString` gibt den Zeitanteil in einer langen Form zurück (z.B. »16:14:49«).

Die Muster, die zur Formatierung verwendet werden, werden in Eigenschaften der `DateTimeFormat`-Eigenschaft des `CultureInfo`-Objekts verwaltet, das dem aktuellen Thread zugeordnet ist (siehe Rezept 39). Diese Eigenschaften heißen `FullDateTimePattern`, `ShortDatePattern`, `LongDatePattern`, `ShortTimePattern` und `LongTimePattern`.

Neben den `DateTime`-Methoden können Sie auch die `Format`-Methode der `String`-Klasse verwenden, der Sie für ein Datum (u.a.) die folgenden Formatzeichen übergeben können:

- *d*: Formatierung im kurzen Datumsformat,
- *D*: Formatierung im langen Datumsformat,
- *t*: Formatierung im kurzen Zeitformat,
- *T*: Formatierung im langen Zeitformat,
- *f*: Formatierung im vollständigen Datumsformat mit kurzer Zeitangabe,
- *F*: Formatierung im vollständigen Datumsformat mit langer Zeitangabe,
- *d*, *dd*, *ddd*, *dddd*: definiert den Tag des Monats als Zahl ohne bzw. mit führender 0 oder in abgekürzter (Mo, Di, Mi) oder voll ausgeschriebener Schreibweise (Montag, Dienstag etc.),
- *M*, *MM*, *MMM*, *MMMM*: definiert den Monat als Zahl ohne bzw. mit führender 0 oder in abgekürzter (Jan, Feb, Mrz) oder voll ausgeschriebener Schreibweise (Januar, Februar etc.),
- *yy*, *yyyy*: bestimmt, dass das Jahr zwei- bzw. vierstellig ausgegeben wird,
- *H*, *HH*: bestimmt, dass die Stunde im 24-Stunden-Format ohne oder mit führender 0 ausgegeben wird. Beachten Sie, dass ein klein geschriebenes *h* die Stunde im 12-Stunden-Format formatiert,
- *m*, *mm*: definiert, dass die Minute ohne oder mit führender 0 ausgegeben wird,
- *s*, *ss*: bestimmt, dass die Sekunde ohne oder mit führender 0 ausgegeben wird.

Listing 56 zeigt einige typische Datumsformatierungen. Das Programm erfordert den Import des Namensraums `System`.

```
DateTime now = DateTime.Now;

Console.WriteLine("ToString:\r\n{0}\r\n", now.ToString());
Console.WriteLine("ToShortDateString:\r\n{0}\r\n",
    now.ToShortDateString());
Console.WriteLine("ToLongDateString:\r\n{0}\r\n",
    now.ToLongDateString());
Console.WriteLine("ToShortTimeString:\r\n{0}\r\n",
    now.ToShortTimeString());
Console.WriteLine("ToLongTimeString:\r\n{0}\r\n",
    now.ToLongTimeString());
Console.WriteLine(String.Format("dd.MM.yy:\r\n{0:dd.MM.yy}\r\n", now));
Console.WriteLine(String.Format("dddd, dd. MMMM yy:\r\n" +
    "{0:dddd, dd. MMMM yyyy}\r\n", now));
Console.WriteLine(String.Format("ddd, dd. MMM yy:\r\n" +
```

Listing 56: Einige typische Datumsformatierungen


```
"{0:ddd, dd. MMM yyyy}\r\n", now));  
Console.WriteLine(String.Format("HH:mm:ss\r\n" +  
    "{0:HH:mm:ss}", now));
```

Listing 56: Einige typische Datumsformatierungen (Forts.)

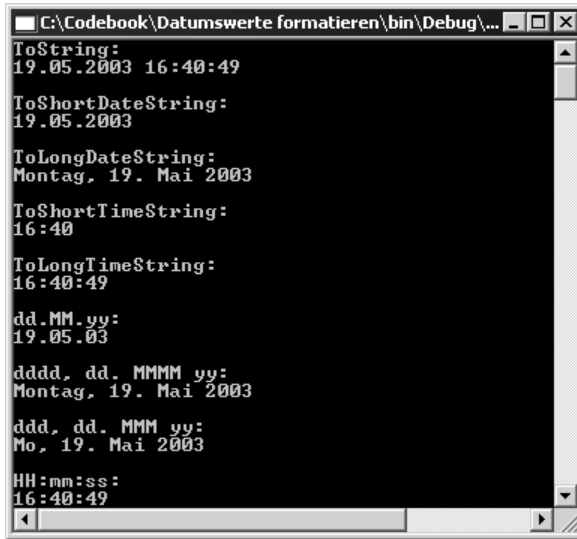


Abbildung 23: Typische Datumsformatierungen auf einem deutschen Rechner

39 Datumswerte in verschiedenen Länderformaten ausgeben

Wenn Sie die To-Methoden eines `DateTime`-Objekts (`ToString`, `ToShortDateString` etc.) aufrufen, werden Datumswerte immer entsprechend der aktuellen Kultur formatiert. Leider besitzen diese Methoden kein Argument, an dem ein `CultureInfo`-Objekt für eine spezielle Kultur übergeben werden kann.

Die `Format`-Methode der `String`-Klasse arbeitet zwar prinzipiell auf dieselbe Weise, einer Variante dieser Methode können Sie aber am ersten Argument ein Objekt übergeben, das die `IFormatProvider`-Schnittstelle implementiert. Wenn Sie kein eigenes Formatier-Objekt implementieren wollen (was wohl nur selten Sinn macht), können Sie dazu ein `CultureInfo`-Objekt verwenden, das die Formatierung dann entsprechend seiner Kultur bestimmt. Da die `Format`-Methode auch Formatierzeichen für festgelegte Formate besitzt (z.B. *d* für ein kurzes Datumsmuster, siehe bei der Dokumentation der Formatzeichenfolgen für Datum und Uhrzeit), können Sie sehr flexibel formatieren.

Wenn Sie die To-Methode eines `DateTime`-Objekts verwenden wollen, müssen Sie die aktuelle Kultur umstellen. Die aktuelle Kultur ist definiert über ein `CultureInfo`-Objekt, das der Eigenschaft `CurrentCulture` des aktuellen Thread zugewiesen ist. Das Thread-Objekt des aktuellen Thread können Sie über `System.Threading.Thread.CurrentThread` auslesen. Um ein

Datum nun über die `To`-Methoden einer `DateTime`-Instanz in einem bestimmten Länderformat auszugeben, müssen Sie dem aktuellen Thread ein spezielles `CultureInfo`-Objekt zuweisen.

Ein `CultureInfo`-Objekt für eine spezielle Kultur können Sie über die statische `CreateSpecificCulture`-Methode der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) erzeugen. Dieser Methode übergeben Sie eine der international genormten Kultur-Kurzbezeichnungen. Tabelle 5 stellt einige wichtige dieser Kurzbezeichnungen dar. Die gesamte Liste finden Sie in der Dokumentation der `CultureInfo`-Klasse.

Kurzbezeichnung	Kultur
de	Deutsch allgemein
de-DE	Deutsch in Deutschland
de-AT	Österreichisches Deutsch
de-CH	Schweizer Deutsch
fr	Französisch allgemein
it	Italienisch allgemein
es	Spanisch allgemein
en	Englisch allgemein
en-US	US-amerikanisches Englisch
en-GB	Englisch in Großbritannien

Tabelle 5: Einige wichtige Kultur-Kurzbezeichnungen

Listing 57 zeigt, wie Sie ein Datum z.B. im US-englischen Format ausgeben. Das Programm verwendet zuerst die `ToLongDateString`-Methode eines `DateTime`-Objekts zur Formatierung. Es speichert dazu die aktuell eingestellte Kultur in der Variablen `currentCulture` zwischen, damit diese später wieder zurückgesetzt werden kann. Dann erzeugt das Programm eine `CultureInfo`-Instanz für die US-englische Kultur und weist diese dem aktuellen Thread zu. Nach der Formatierung des Datums werts wird die zuvor eingestellte Kultur dann wieder zurückgeschrieben.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.Globalization` und `System.Threading` importieren.

```
// Aktuell eingestellte Kultur merken
CultureInfo currentCulture = Thread.CurrentThread.CurrentCulture;

// Aktuelle Kultur auf die englische umschalten
Thread.CurrentThread.CurrentCulture =
    CultureInfo.CreateSpecificCulture("en-US");

// Datumswert formatieren und ausgeben
string formattedDate = DateTime.Now.ToLongDateString();
```

```
Console.WriteLine(formattedDate);

// Aktuelle Kultur wieder zurücksetzen
Thread.CurrentThread.CurrentCulture = currentCulture;

// Datum über String.Format formatieren
formattedDate = String.Format(CultureInfo.CreateSpecificCulture(
    "en-US"), "{0:F}", DateTime.Now);
Console.WriteLine(formattedDate);
```

Listing 57: Ausgabe des aktuellen Datums im US-englischen Format

Zu Kompilierung dieses Programms müssen Sie die Namensräume `System.Globalization` und `System.Threading` importieren. Abbildung 24 zeigt die Ausgabe an der Konsole.

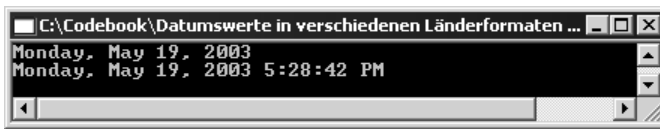


Abbildung 24: Das aktuelle Datum im US-englischen Format

40 ISO-Datum ermitteln

Das Schweizer ISO-Institut (International Organization for Standardization), das ähnlich dem deutschen DIN-Institut (internationale) Standards entwickelt und veröffentlicht, hat in der ISO-Norm 8601 (www.iso.ch/iso/en/prods-services/popstds/datesandtime.html) u.a. einen Standard für die Übermittlung von Datumswerten geschaffen. Dieser Standard ist sehr interessant für die Angabe von Datumswerten in SQL-Anweisungen, da viele Datenbanksysteme (z.B. MySQL und der Microsoft SQL Server) neben einem proprietären Format auch das ISO-Format unterstützen. Wenn Sie dieses Format verwenden, müssen Sie sich keine Gedanken um das spezielle Format des Datenbanksystems machen. Aber auch bei der Übertragung von Daten in Form von XML- oder Textdateien kann das ISO-Datum dabei helfen, dass unterschiedliche Systeme keine Probleme mit Datumswerten haben.

ISO 8601 sieht vor, dass ein Datum im folgenden Format angegeben wird:

```
YYYY-MM-DD
```

Alternativ können Sie das folgende Format verwenden, bei dem die Kalenderwoche an Stelle des Monats und der Wochentag an Stelle des Tags steht:

```
YYYY-Www-D
```

Eine Zeitangabe ist folgendermaßen definiert:

hh:mm:ss

Sie können beide Angaben kombinieren, wobei nach der Norm ein *T* zwischen Datum und Zeit stehen muss (viele Systeme nehmen an Stelle des *T* aber auch ein Leerzeichen an). Das Datum 1.6.2005 10:50:00 wird z.B. so angegeben:

2005-06-01T10:50:00

Die Methode `DateTime2ISO` in Listing 58 setzt die `Format`-Methode der `String`-Klasse ein um das übergebene Datum in das ISO-8601-Format zu formatieren und zurückzugeben. Das Argument `includeTime` bestimmt dabei, ob die Zeit mit eingeschlossen werden soll.

`DateTime2ISO` erfordert den Import des Namensraums `System`.

```
public static string DateTime2ISO(DateTime date, bool includeTime)
{
    if (includeTime)
        return String.Format("{0:yyyy-MM-ddThh:mm:ss}", date);
    else
        return String.Format("{0:yyyy-MM-dd}", date);
}
```

Listing 58: Methode zur Ermittlung eines ISO-Datums

41 Datumswerte vergleichen

`DateTime`-Instanzen können Sie direkt miteinander vergleichen, da die `DateTime`-Struktur die Vergleichsoperatoren überlädt. Diese Operatoren berücksichtigen neben dem Datumsteil natürlich auch den Zeiteil. Das Datum 01.01.2004 10:00:00 ist demnach größer als das Datum 01.01.2004 09:59:59. Listing 59 zeigt einige typische Vergleiche von Datumswerten. Das Programm erfordert den Import des Namensraums `System`.

```
// DateTime-Instanzen erzeugen
DateTime d1 = new DateTime(2003, 12, 31);
DateTime d2 = new DateTime(2004, 1, 1);
DateTime d3 = new DateTime(2004, 1, 1);
DateTime d4 = new DateTime(2004, 1, 1, 12, 0, 0);

if (d1 < d2)
    Console.WriteLine("{0} ist kleiner als {1}", d1, d2);
if (d2 == d3)
    Console.WriteLine("{0} ist gleich {1}", d2, d3);
```

Listing 59: Vergleichen von Datumswerten

```

if (d3 != d4)
    Console.WriteLine("{0} ist ungleich {1}", d3, d4);
if (d4 > d3)
    Console.WriteLine("{0} ist größer als {1}", d4, d3);

```

Listing 59: Vergleichen von Datumswerten (Forts.)



Abbildung 25: Das Beispielprogramm zum Vergleichen von Datumswerten

42 Die Differenz zwischen zwei Datumswerten in Tagen und deren Teilen berechnen

Die Differenz zwischen zwei Datumswerten können Sie einfach über eine Subtraktion des kleineren vom größeren Datumswert ermitteln. Eine Subtraktion zweier `DateTime`-Werte (aus dem Namensraum `System`) ergibt ein `TimeSpan`-Objekt. Über die Eigenschaften `Ticks` (Anzahl der 100-Nanosekunden-Intervalle zwischen den Datumswerten), `Days`, `Hours`, `Minutes`, `Seconds` und `Milliseconds` können Sie die Differenz zwischen den beiden Datumswerten auslesen:

```

// Zwei DateTime-Objekte erzeugen
DateTime birthDate = new DateTime(2000, 1, 2);
DateTime now = DateTime.Now;

// Berechnen der Differenz
TimeSpan dateDiff = now - birthDate;

// Ausgabe der Differenz
Console.WriteLine("Differenz zwischen dem {0}\r\nund dem {1}:",
    birthDate.ToString(), now.ToString());
Console.WriteLine("Ticks: {0}", dateDiff.Ticks);
Console.WriteLine("Tage: {0}", dateDiff.Days);
Console.WriteLine("Stunden: {0}", dateDiff.Hours);
Console.WriteLine("Minuten: {0}", dateDiff.Minutes);
Console.WriteLine("Sekunden: {0}", dateDiff.Seconds);
Console.WriteLine("Millisekunden: {0}", dateDiff.Milliseconds);

```

Listing 60: Berechnen der Differenz zwischen zwei Datumswerten



Abbildung 26: Einfache Datumsdifferenzberechnung

43 Die Differenz zwischen zwei Datumswerten in höheren Datumseinheiten berechnen

Wenn Sie die Differenz zwischen zwei Datumswerten in Einheiten berechnen wollen, die größer sind als Tage (also in Wochen, Monaten, Quartalen oder Jahren), können Sie dazu kein `TimeSpan`-Objekt einsetzen. Der Grund dafür ist, dass eine solche Berechnung davon abhängt, an welchem Datum diese beginnt und endet und wie lang das jeweilige Intervall ist. Bei der Berechnung der Anzahl der Monate zwischen zwei Datumswerten muss z.B. die unterschiedliche Länge der einzelnen Monate berücksichtigt werden.

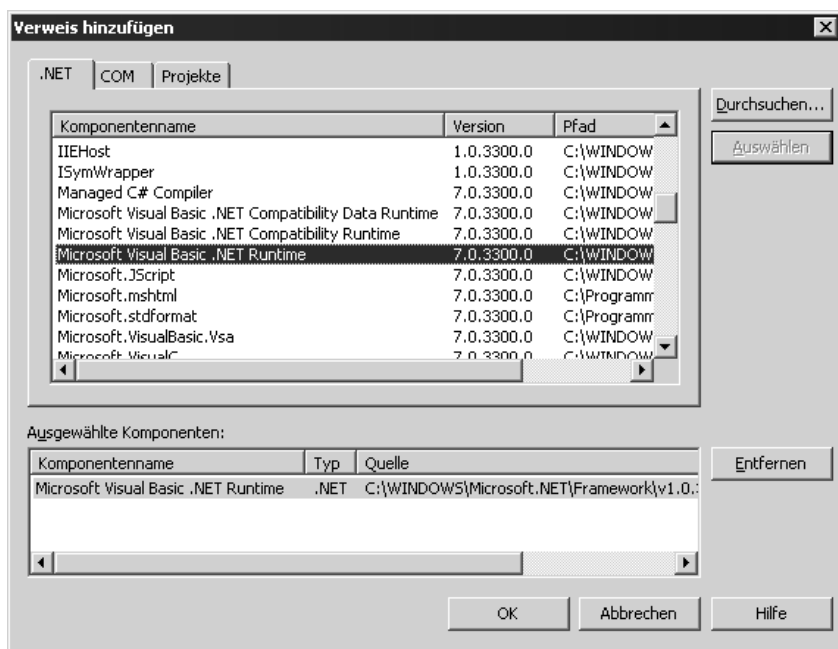


Abbildung 27: Hinzufügen eines Verweises auf die Microsoft Visual Basic .NET Runtime-Assembly in Visual Studio .NET

Für eine solche Berechnung können Sie eine kleine Anleihe bei Visual Basic machen. Im Modul `DateAndTime` ist nämlich die Methode `DateDiff` implementiert, die die Differenz zwischen zwei Datumswerten in Einheiten eines gegebenen Intervalls berechnet. Diese Funktionalität steht in den »normalen« .NET-Klassen leider nicht zur Verfügung, was auch daran erkannt werden kann, dass (bewiesen über den .NET-Disassembler *ildasm*) `DateDiff` eine in `DateAndTime` hart-codierte Methode ist.

Referenzieren Sie also die Assembly *Microsoft.VisualBasic.dll* um diese Klasse verwenden zu können. In Visual Studio .NET finden Sie diese Assembly unter dem Namen *Microsoft Visual Basic .NET Runtime*.

Nun können Sie die Differenz zwischen zwei Datumswerten sehr flexibel berechnen. Die (leider zu Visual Basic 6 kompatibel) `DateDiff`-Methode erwartet am ersten Argument die Angabe des Intervalls als String. Sie können einen Wert der Aufzählung `DateInterval` einsetzen. Tabelle 6 listet die für unsere Zwecke wichtigen auf.

Intervallzeichen	Berechnung in
<code>DateInterval.Month</code>	Monaten
<code>DateInterval.Quarter</code>	Quartalen
<code>DateInterval.Year</code>	Jahren
<code>DateInterval.Weekday</code>	Wochen

Tabelle 6: Die wichtigen Intervallzeichen der `DateDiff`-Methode

Am zweiten Argument übergeben Sie den kleineren, am dritten Argument den größeren Datumswert. Da diese Argumente in einer Variante als `DateTime` und in der anderen als `object` deklariert sind, können Sie die Datumswerte über `DateTime`-Instanzen oder als String (im aktuellen Systemformat) übergeben.

Das vierte Argument bestimmt mit den Werten der `FirstDayOfWeek`-Aufzählung, mit welchem Tag die Woche beginnt (in den USA beginnt die Woche z.B. mit dem Sonntag). Sie sollten hier immer `FirstDayOfWeek.System` angeben, damit die aktuelle Systemeinstellung verwendet wird. Ähnliches gilt für das letzte Argument, das vom Typ `FirstWeekOfYear` ist und die Woche im Januar bestimmt, die als Kalenderwoche 1 gerechnet wird. Übergeben Sie hier den Wert `FirstWeekOfYear.System`.

Listing 61 zeigt eine Anwendung der `DateDiff`-Methode zur Berechnung der Differenz zwischen zwei Datumswerten in Wochen, Monaten, Quartalen und Jahren. Das Programm erfordert die Referenzierung der Assembly *Microsoft.VisualBasic.dll* und den Import der Namensräume `System` und `Microsoft.VisualBasic`.

```
// Zwei DateTime-Objekte erzeugen
DateTime birthDate = new DateTime(2000, 1, 2);
DateTime now = DateTime.Now;
```

Listing 61: Berechnen der Differenz zwischen zwei Datumswerten in höheren Einheiten

```
// Berechnen der Differenz
long weeks = DateAndTime.DateDiff(DateInterval.Weekday, birthDate,
    now, FirstDayOfWeek.System, FirstWeekOfYear.System);
long months = DateAndTime.DateDiff(DateInterval.Month, birthDate,
    now, FirstDayOfWeek.System, FirstWeekOfYear.System);
long quarters = DateAndTime.DateDiff(DateInterval.Quarter,
    birthDate, now, FirstDayOfWeek.System, FirstWeekOfYear.System);
long years = DateAndTime.DateDiff(DateInterval.Year, birthDate, now,
    FirstDayOfWeek.System, FirstWeekOfYear.System);

// Ausgeben der berechneten Werte
Console.WriteLine("Differenz zwischen dem {0}\r\nund dem {1}:",
    birthDate.ToString(), now.ToString());
Console.WriteLine("Wochen: {0}", weeks);
Console.WriteLine("Monate: {0}", months);
Console.WriteLine("Quartale: {0}", quarters);
Console.WriteLine("Jahre: {0}", years);
```

Listing 61: Berechnen der Differenz zwischen zwei Datumswerten in höheren Einheiten (Forts.)

Abbildung 28 zeigt das Ergebnis des Programms.

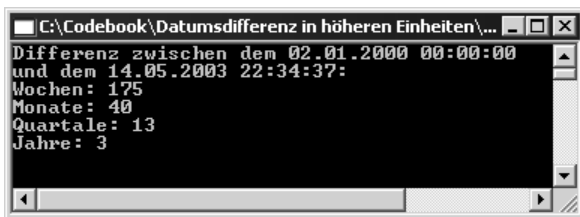


Abbildung 28: Datumsberechnung in höheren Einheiten

Hinweis

Beachten Sie, dass `DateDiff` bei allen Intervallen, die größer sind als Wochen, immer berechnet, wie viele Intervalle zwischen zwei Datumswerten liegen, wenn nur die Intervall-Teile des Datums berücksichtigt werden. Wenn Sie z.B. die Differenz zwischen dem 31.12.2002 und dem 1.1.2003 berechnen, erhalten Sie als Ergebnis ein Jahr, weil zwischen 2002 und 2003 genau ein Jahr liegt. Genauso erhalten Sie als Ergebnis ein Quartal und einen Monat, weil zwischen dem Quartal 4 und dem Quartal 1 ein Quartal bzw. zwischen dem Monat 12 und dem Monat 1 ein Monat liegt. Wenn Sie die *ganzen* Jahre, Quartale oder Monate zwischen zwei Datumswerten berechnen wollen, müssen Sie leider selbst rechnen.

44 Ein Alter in Jahren berechnen

Wenn Sie das Alter einer Person oder eines Gegenstands in Jahren berechnen wollen, können Sie dazu leider nicht die Visual-Basic-Methode `DateDiff` aus dem Rezept 43 verwenden, da diese Methode bei Berechnungen in Jahren immer nur den Jahresanteil des Datums berück-

sichtigt (die Differenz zwischen dem 31.12.2002 und dem 1.1.2003 ist demnach ein Jahr). Eine Altersangabe müssen Sie selbst berechnen.

Die Methode `GetAge` in Listing 62 implementiert eine solche Altersberechnung. `GetAge` berechnet dazu zunächst die Differenz der Jahresangabe in beiden Datumswerten. Danach wird überprüft, ob das größere Datum bezogen auf den Tag und den Monat größer oder gleich dem kleineren Datum ist. Ist dies nicht der Fall, zieht `GetAge` von der Jahresdifferenz ein Jahr ab, da der Geburtstag noch nicht erreicht ist.

`GetAge` erfordert den Import des Namensraums `System`.

```
public static int GetAge(DateTime lowerDate, DateTime higherDate)
{
    // Basis-Alter als Differenz zwischen den Jahren ermitteln
    int age = higherDate.Year - lowerDate.Year;

    // Wenn der Monat des größeren Datums kleiner oder wenn der Monat gleich
    // und der Tag kleiner ist, muss ein Jahr abgezogen werden
    if ((higherDate.Month < lowerDate.Month) ||
        (higherDate.Month == lowerDate.Month &&
         higherDate.Day < lowerDate.Day))
        age--;

    return age;
}
```

Listing 62: Berechnen eines Alter in Jahren

45 Datumswerte addieren und subtrahieren

Häufig ist es notwendig, zu einem gegebenen Datum eine bestimmte Anzahl Datumseinheiten (Minuten, Stunden, Tage, Wochen etc.) zu addieren oder davon abzuziehen. Die `DateTime`-Struktur bietet dazu entsprechende Methoden (Tabelle 7).

Methode	Addiert bzw. subtrahiert ...
<code>DateTime.Add(TimeSpan value)</code>	einen <code>TimeSpan</code> -Wert
<code>DateTime.AddDays(double value)</code>	Tage
<code>DateTime.AddHours(double value)</code>	Stunden
<code>DateTime.AddMilliseconds(double value)</code>	Millisekunden
<code>DateTime.AddMinutes(double value)</code>	Minuten
<code>DateTime.AddMonths(int months)</code>	Monate
<code>DateTime.AddSeconds(double value)</code>	Sekunden
<code>DateTime.AddTicks(long value)</code>	Ticks (100ns-Intervalle)
<code>DateTime.AddYears(int value)</code>	Jahre

Tabelle 7: Die Methoden der `DateTime`-Struktur zum Addieren und Subtrahieren von Datumseinheiten

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Da Sie den Methoden auch negative Werte übergeben können, können Sie einem Datum auch Einheiten abziehen. Die meisten Add-Methoden arbeiten mit `double`-Argumenten, sodass Sie auch Bruchteile der Einheit addieren oder subtrahieren können. Bei Monaten und Jahren (die eine unterschiedliche Anzahl an Tagen besitzen) und bei Ticks ist das prinzipiell nicht möglich, weswegen am Argument dieser Methoden `int`- bzw. `long`-Werte erwartet werden.

Listing 63 zeigt die Anwendung dieser Methoden. Zum Kompilieren des Beispiels müssen Sie den Namensraum `System` importieren.

```
// Aktuelles Datum ermitteln
DateTime now = DateTime.Now;
Console.WriteLine("Jetzt: {0}", now.ToString());

// 12,5 Stunden abziehen
DateTime date = now.AddHours(-12.5);
Console.WriteLine("Jetzt - 12,5 Stunden: {0}", date.ToString());

// 10,5 Tage hinzuaddieren
date = now.AddDays(10.5);
Console.WriteLine("Jetzt + 10,5 Tage: {0}", date.ToString());

// Sechs Monate hinzuaddieren
date = now.AddMonths(6);
Console.WriteLine("Jetzt + 6 Monate: {0}", date.ToString());

// Zwei Jahre abziehen
date = now.AddYears(-2);
Console.WriteLine("Jetzt - 2 Jahre: {0}", date.ToString());

// Mein Alter als TimeSpan-Wert berechnen (natürlich mit
// gelogenem Geburtsdatum ;-))
DateTime birthDate = new DateTime(1972, 9, 27, 2, 30, 0);
Console.WriteLine("Mein (gelogenes) Geburtsdatum: {0}",
    birthDate.ToString());
TimeSpan age = now - birthDate;

// Damit errechnen, welches Datum meinem doppelten Alter entspricht
date = now.Add(age);
Console.WriteLine("Dann bin ich doppelt so alt wie jetzt: {0}",
    date.ToString());
```

Listing 63: Addieren und Subtrahieren von Datumseinheiten

46 Die Anzahl der Tage eines Monats berechnen

Die Anzahl der Tage eines Monats können Sie über die Methode `DaysInMonth` der `DateTime`-Struktur berechnen. Dieser Methode übergeben Sie am ersten Argument das Jahr und am zweiten den Monat. Das Beispiel in Listing 64, das den Import des `System`-Namensraums erfordert, berechnet die Anzahl der Tage des aktuellen Monats.

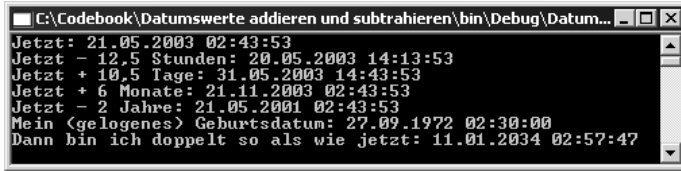


Abbildung 29: Die Datumsberechnung

```
DateTime now = DateTime.Now;
int daysInMonth = DateTime.DaysInMonth(now.Year, now.Month);

Console.WriteLine("Tage des Monats {0}/{1}: {2}",
    now.Month, now.Year, daysInMonth);
```

Listing 64: Berechnen der Anzahl der Tage des aktuellen Monats



Abbildung 30: Das Beispielprogramm in Aktion

47 Die Anzahl der Tage eines Jahres berechnen

Die Anzahl der Tage eines Jahres können Sie nicht wie bei der Anzahl der Tage eines Monats über eine Methode der `DateTime`-Struktur berechnen. Dazu müssen Sie ein Objekt einer von `Calendar` abgeleiteten Klasse verwenden, das den Kalender des Systems repräsentiert (bei uns ist das der gregorianische Kalender). Den aktuellen Kalender können Sie über die `Calendar`-Eigenschaft des `CultureInfo`-Objekts, das die aktuelle Kultur darstellt (`System.Globalization.CultureInfo.CurrentCulture.Calendar`), ermitteln. Die Anzahl der Tage in einem Jahr erhalten Sie dann über die Methode `GetDaysInYear`, der Sie das Jahr übergeben, dessen Tage Sie ermitteln wollen.

Listing 65 zeigt dies für das aktuelle Jahr. Das Beispiel erfordert den Import der Namensräume `System` und `System.Globalization`.

```
// GregorianCalendar-Objekt für den aktuellen Kalender erzeugen
GregorianCalendar calendar = new GregorianCalendar();

// Anzahl der Tage im aktuellen Jahr berechnen
DateTime now = DateTime.Now;
int daysInYear = calendar.GetDaysInYear(now.Year);

Console.WriteLine("Tage des Jahres {0}: {1}",
    now.Year, daysInYear);
```

Listing 65: Berechnen der Anzahl der Tage eines Jahres



Abbildung 31: Das Beispielprogramm in Aktion

48 Ermitteln, ob ein Jahr ein Schaltjahr ist

Über die statische Methode `IsLeapYear` der `DateTime`-Struktur können Sie ermitteln, ob ein Jahr ein Schaltjahr ist. `IsLeapYear` gibt `true` zurück, wenn es sich um ein Schaltjahr handelt.

Listing 66 ermittelt, ob das aktuelle Jahr ein Schaltjahr ist. Das Beispiel erfordert den Import des Namensraums `System`.

```
// Aktuelles Datum ermitteln
DateTime now = DateTime.Now;

// Ermitteln, ob es sich um ein Schaltjahr handelt
if (DateTime.IsLeapYear(now.Year))
    Console.WriteLine("Schaltjahr");
else
    Console.WriteLine("Kein Schaltjahr");
```

Listing 66: Ermitteln, ob das aktuelle Jahr ein Schaltjahr ist

49 Ermitteln der ISO-Kalenderwoche

Die ISO¹-Norm 8601, nach der in Deutschland die Kalenderwoche berechnet wird, definiert die erste Woche im Jahr als die Woche, in die mindestens vier der ersten sieben Januartage fallen (also die Woche, die den ersten Donnerstag im Jahr enthält). Danach ergeben sich die folgenden Aussagen:

- ▶ Beginnt das Jahr mit einem Montag, so entspricht der 1.1. der ersten Kalenderwoche und das Vorjahr schließt mit dem Ende der letzten Kalenderwoche ab.
- ▶ Beginnt das Jahr mit einem Dienstag, Mittwoch oder Donnerstag, werden die ersten Tage der Woche, die im Vorjahr liegen, der ersten Kalenderwoche des neuen Jahres zugerechnet.
- ▶ Beginnt das Jahr mit einem Freitag, Samstag oder Sonntag, werden diese Tage der letzten Kalenderwoche des Vorjahres zugerechnet.
- ▶ Je nachdem, wann die erste Woche im Jahr startet und die letzte endet, kann das Jahr 52 oder 53 Wochen umfassen.

1. ISO (International Organization for Standardization) ist ein Schweizer Institut, das internationale Standards entwickelt und veröffentlicht

Die Kalenderwoche eines Datums können Sie (mit einem kleinen Fehler) über eine Instanz der von `Calendar` abgeleiteten Klasse ermitteln, die den aktuellen Kalender des Systems darstellt. Diese Instanz ermitteln Sie über die `Calendar`-Eigenschaft der Klasse `System.Globalization.CultureInfo.CurrentCulture`. Über die Methode `GetWeekOfYear` berechnen Sie die Kalenderwoche eines Datums, das Sie am ersten Argument übergeben. Am zweiten Argument definieren Sie mit einem Wert der Aufzählung `CalendarWeekRule`, welche Woche als erste Kalenderwoche gezählt wird. Dazu stehen Ihnen die Werte der `CalendarWeekRule`-Aufzählung zur Verfügung. Der Wert `FirstDay` bestimmt, dass die Woche mit dem ersten Januar die erste Woche ist, `FirstFourDayWeek`, dass die Woche die erste ist, die mindestens vier Tage im neuen Jahr besitzt (Standard für Deutschland), über den Wert `FirstFullWeek` legen Sie die erste im neuen Jahr komplette Woche als Woche 1 fest. Am letzten Argument spezifizieren Sie mit den Werten der `DayOfWeek`-Aufzählung (deren Konstanten die Wochentagsnamen sind), mit welchem Tag die Woche beginnt (Sonntag in den USA, Montag in Deutschland). An den beiden letzten Argumenten sollten Sie die aktuellen Kultureinstellungen übergeben, die Sie aus der Eigenschaft `DateTimeFormat` eines `CultureInfo`-Objekts auslesen können.

Listing 67 implementiert dies in Form der Methode `GetCalendarWeek1`, die die Kalenderwoche eines übergebenen Datums ermittelt und in einer Instanz der Klasse `CalendarWeek` zurückgibt.

Hinweis

Die `GetWeekOfYear`-Methode eines `Calendar`-Objekts berechnet die Kalenderwoche leider nicht genau nach dem ISO-8601-Standard. Am Ende des Jahres treten häufig Fehler auf. Für den 30.12.2002 (der ein Montag war) ergibt die Berechnung z.B. 53, obwohl diese Woche fünf Tage (Mittwoch bis Sonntag) im Jahr 2003 besitzt und deshalb eigentlich die Woche 1 sein sollte. Die Berechnung für den 1.1.2003 ergibt dann auch wie erwartet 1. Microsoft dokumentiert dieses Verhalten im Knowledge-Base-Artikel 200299 (für Visual Basic 6). Um diesen »Bug« (der leider auch im Visual Basic-Äquivalent `DatePart` vorhanden ist) zu umschiffen, überprüft `GetCalendarWeek1`, ob die ermittelte Kalenderwoche größer als 52 ist und ob die für das Datum addiert mit einer Woche ermittelte Kalenderwoche 2 ergibt. In diesem Fall wird die Kalenderwoche 1 zurückgegeben.

Dieser kleine Bugfix hat sich als zuverlässig erwiesen. In einem Test wurde die Kalenderwoche für die Jahre 2001 bis 2010 korrekt berechnet (was ich über den Kalender von Outlook überprüft habe).

`GetCalendarWeek1` gibt neben der Kalenderwoche auch das Jahr zurück, da es vorkommen kann, dass ein Datum in der letzten Woche eines Jahres der ersten Kalenderwoche des nächsten Jahres oder ein Datum in der ersten Woche eines Jahres der letzten Kalenderwoche des Vorjahres zugeordnet wird. Das Jahr wird deshalb zunächst aus dem übergebenen Datum ausgelesen. Dann überprüft `GetCalendarWeek1`, ob die Woche 1 ermittelt wurde und der Monat des Datums 12 ist. In diesem Fall wurde das Datum der ersten Kalenderwoche des neuen Jahres zugeordnet und das Jahr wird folglich um 1 erhöht. Dann wird noch überprüft, ob die Kalenderwoche 52 oder 53 und der Monat des Datums 1 ist. In diesem Fall wurde das Datum der letzten Kalenderwoche des Vorjahres zugeordnet und das Jahr wird um 1 erniedrigt.

Mit den ermittelten Daten erzeugt `GetCalendarWeek1` schließlich ein `CalendarWeek`-Objekt (dessen Klasse in Listing 67 deklariert ist) und gibt eine Referenz darauf zurück.

Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```

/* Klasse für die Rückgabe der Kalenderwoche */
public class CalendarWeek
{
    public int Year;
    public int Week;
    public CalendarWeek(int year,int week)
    {
        this.Year = year;
        this.Week = week;
    }
}

/* Methode zur Ermittlung der Kalenderwoche eines Datums nach der
 * GetWeekOfYear-Methode eines Calendar-Objekts */
public static CalendarWeek GetCalendarWeek1(DateTime date)
{
    // Aktuelle Kultur ermitteln
    CultureInfo currentCulture = CultureInfo.CurrentCulture;

    // Aktuelle Kalender ermitteln
    Calendar calendar = currentCulture.Calendar;

    // Kalenderwoche über das Calendar-Objekt ermitteln
    int calendarWeek = calendar.GetWeekOfYear(date,
        currentCulture.DateTimeFormat.CalendarWeekRule,
        currentCulture.DateTimeFormat.FirstDayOfWeek);

    // Überprüfen, ob eine Kalenderwoche größer als 52 ermittelt wurde und ob
    // die Kalenderwoche des Datums in einer Woche 2 ergibt: In diesem Fall hat
    // GetWeekOfYear die Kalenderwoche nicht nach ISO 8601 berechnet (Montag,
    // der 30.12.2003 wird z. B. fälschlicherweise als KW 53 berechnet).
    // Die Kalenderwoche wird dann auf 1 gesetzt
    if (calendarWeek > 52)
    {
        date = date.AddDays(7);
        int testCalendarWeek = calendar.GetWeekOfYear(date,
            currentCulture.DateTimeFormat.CalendarWeekRule,
            currentCulture.DateTimeFormat.FirstDayOfWeek);
        if (testCalendarWeek == 2)
            calendarWeek = 1;
    }

    // Das Jahr der Kalenderwoche ermitteln
    int year = date.Year;

```

Listing 67: Methode zur Ermittlung der Kalenderwoche eines Datums über ein `Calendar`-Objekt

```

    if (calendarWeek == 1 && date.Month == 12)
        year++;
    if (calendarWeek >= 52 && date.Month == 1)
        year--;

    // Die ermittelte Kalenderwoche zurückgeben
    return new CalendarWeek(year, calendarWeek);
}

```

Listing 67: Methode zur Ermittlung der Kalenderwoche eines Datums über ein Calendar-Objekt (Forts.)

Eine andere Möglichkeit, die Kalenderwoche zu berechnen, ist ein eigener Algorithmus. Im Internet kursieren verschiedene Algorithmen, die aber teilweise Kalenderwochen falsch berechnen. Ich habe selbst versucht, eine Berechnungs-Methode zu entwickeln, bin aber an einigen wenigen Sonderfällen gescheitert. Die Berechnung ist nicht trivial, auch wenn es auf den ersten Blick so erscheint.

Der Algorithmus, auf dem die Methode `GetCalendarWeek2` in Listing 68 basiert und den ich in einem Newsgroup-Beitrag gefunden habe, berechnet die Kalenderwoche anscheinend (und nach meinen Tests) sehr genau. Ein Vergleich mit der Methode `GetCalendarWeek1` hat auf jeden Fall ergeben, dass beide Methoden die Kalenderwochen der Jahre 1899 bis 2999 gleich berechnen. Ich habe den Algorithmus nahezu unverändert übernommen, aber um die Berechnung des Jahres für die Kalenderwoche erweitert. Das Jahr wird auf dieselbe Weise berechnet wie in der Methode `GetCalendarWeek1`.

`GetCalendarWeek2` erfordert den Import des Namensraums `System`.

```

/* Methode zur Berechnung der Kalenderwoche nach dem C++-Algorithmus von
 * Ekkehard Hess aus einem Beitrag vom 29.7.1999 in der Newsgroup
 * borland.public.cppbuilder.language (freigegeben zur allgemeinen
 * Verwendung) */
public static CalendarWeek GetCalendarWeek2(DateTime date)
{
    double a = Math.Floor((14 - (date.Month)) / 12);
    double y = date.Year + 4800 - a;
    double m = (date.Month) + (12 * a) - 3;

    double jd = date.Day + Math.Floor(((153 * m) + 2) / 5) +
        (365 * y) + Math.Floor(y / 4) - Math.Floor(y / 100) +
        Math.Floor(y / 400) - 32045;

    double d4 = (jd + 31741 - (jd % 7)) % 146097 % 36524 %
        1461;
    double L = Math.Floor(d4 / 1460);
    double d1 = ((d4 - L) % 365) + L;

    // Kalenderwoche ermitteln

```

Listing 68: Methode zur Ermittlung der Kalenderwoche eines Datums über einen komplexen Algorithmus

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

int calendarWeek = (int) Math.Floor(d1 / 7) + 1;

// Das Jahr der Kalenderwoche ermitteln
int year = date.Year;
if (calendarWeek == 1 && date.Month == 12)
    year++;
if (calendarWeek >= 52 && date.Month == 1)
    year--;

// Die ermittelte Kalenderwoche zurückgeben
return new CalendarWeek(year, calendarWeek);
}

```

Listing 68: Methode zur Ermittlung der Kalenderwoche eines Datums über einen komplexen Algorithmus (Forts.)

50 Ermitteln, mit welchem Datum eine Kalenderwoche beginnt

Startdatum einer deutschen Kalenderwoche ermitteln

Das Datum des Montags einer beliebigen Kalenderwoche können Sie für Deutschland eigentlich recht einfach berechnen. Der DIN-Norm für Datumsberechnungen (DIN 1355) können Sie entnehmen, dass der 4.1. eines Jahres immer in die Woche 1 fällt (Dank an die Kollegen, die das Access-Codebook geschrieben haben ☺). Mit diesem Datum können Sie das Datum ermitteln, das den Montag dieser Woche repräsentiert. Um das Startdatum einer Kalenderwoche zu ermitteln, können Sie dem Startdatum der ersten Kalenderwoche ganz einfach die Anzahl der Kalenderwochen -1 multipliziert mit 7 addieren, wie es Listing 69 zeigt. Der Montag der Woche, in die der 4.1. fällt, wird in diesem Beispiel über eine Ermittlung der Wochentagszahl berechnet. Die `DayOfWeek`-Aufzählung eines `DateTime`-Objekts verwaltet den Wochentag als Zahl von 0 (Sonntag) bis 6 (Samstag). Da die 0 den Sonntag darstellt, muss dieser extra behandelt werden. Ansonsten wird der Montag einfach berechnet, indem die Wochentagsnummer -1 vom Datum abgezogen wird. Mit diesem Basisdatum wird dann einfach über eine Addition der Tage der Kalenderwochen bis zur abgefragten Woche das Startdatum dieser Woche ermittelt.

Die Methode `GetGermanCalendarWeekStartDate` erfordert die Einbindung der Namensräume `System` und `System.Globalization`.

```

public static DateTime GetGermanCalendarWeekStartDate(int calendarWeek,
    int year)
{
    // Datum für den 4.1. des Jahres ermitteln
    DateTime baseDate = new DateTime(year, 1, 4);

    // Den Montag dieser Woche ermitteln
    int dayOfWeek = (int)baseDate.DayOfWeek;

```

Listing 69: Ermitteln des Startdatums einer deutschen Kalenderwoche

```

    if (dayOfWeek > 0)
        // Montag bis Samstag
        baseDate = baseDate.AddDays((dayOfWeek - 1) * -1);
    else
        // Sonntag
        baseDate = baseDate.AddDays(-6);

    // Das Ergebnisdatum ermitteln
    return baseDate.AddDays((calendarWeek - 1) * 7);
}

```

Listing 69: Ermitteln des Startdatums einer deutschen Kalenderwoche (Forts.)

Startdatum einer internationalen Kalenderwoche ermitteln

Die Methode `GetGermanCalendarWeekStartDate` ist nicht geeignet für Länder, bei denen die erste Kalenderwoche anders gerechnet wird als in Deutschland und/oder bei denen die Woche mit einem anderen Tag als dem Montag beginnt. Die USA, wo die Woche mit dem Sonntag beginnt, ist ein Beispiel dafür. Um auch für diese Länder das Startdatum einer Kalenderwoche berechnen zu können, habe ich die (komplexere und deutlich langsamere) Methode `GetCalendarWeekStartDate` in Listing 70 entwickelt. Diese Methode ermittelt zunächst das Startdatum des Jahres (der 1.1. des übergebenen Jahres). Ausgehend von diesem Datum, das ja z.B. ein Mittwoch sein kann, wird dann das Datum des ersten Wochentags dieser Woche ermittelt. Dazu überprüft das Programm in einer Schleife, ob der Wochentag des Basisdatums dem aktuell eingestellten Wochenstart-Wochentag entspricht (der aus der Eigenschaft `CurrentCulture.DateTimeFormat.FirstDayOfWeek` der `CultureInfo`-Klasse aus dem Namensraum `System.Globalization` ausgelesen werden kann) und reduziert das Basisdatum in der Schleife jeweils um einen Tag (eine einfachere Lösung habe ich nicht gefunden, da die Wochen international ja mit unterschiedlichen Tagen beginnen können).

Mit dem ermittelten Basisdatum wird dann über die Methode `GetCalendarWeek2` aus dem Rezept 49 die Kalenderwoche berechnet. Handelt es sich nicht um die Woche 1, addiert `GetCalendarWeekStartDate` einfach sieben Tage auf das Basisdatum, um den Wochenbeginn der ersten Kalenderwoche zu finden. Auf das so ermittelte Datum werden dann noch die Tage bis zur angegebenen Kalenderwoche addiert.

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```

/* Die Implementierung der Klasse CalendarWeek und der Methoden
 * GetCalendarWeek1 und GetCalendarWeek2 aus dem vorhergehenden Rezept
 * wird aus Übersichtsgründen hier nicht dargestellt */

public static DateTime GetCalendarWeekStartDate(int calendarWeek, int year)
{
    // Basisdatum (1.1. des angegebenen Jahres) ermitteln
    DateTime startDate = new DateTime(year, 1, 1);

```

Listing 70: Berechnen des Startdatums einer Kalenderwoche

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

// Das Datum des ersten Wochentags dieser Woche ermitteln
while (startDate.DayOfWeek !=
    CultureInfo.CurrentCulture.DateTimeFormat.FirstDayOfWeek)
    startDate = startDate.AddDays(-1);

// Die Kalenderwoche ermitteln: Wenn es sich um die Woche 1 handelt,
// ist dies das Basisdatum für die Berechnung, wenn nicht, müssen
// sieben Tage aufaddiert werden
CalendarWeek cw = GetCalendarWeek2(startDate);
if (cw.Week != 1)
    startDate = startDate.AddDays(7);

// Das Ergebnisdatum ermitteln
return startDate.AddDays((calendarWeek - 1) * 7);
}

```

Listing 70: Berechnen des Startdatums einer Kalenderwoche (Forts.)

Hinweis

Die Berechnung des Startdatums ist aufgrund eines unerwarteten Nebeneffekts sehr tolerant. So können Sie als Kalenderwoche auch 53 übergeben, wenn das Jahr nur 52 Wochen besitzt. `GetCalendarWeekStartDate` sieht diese Woche dann einfach als Woche 1 des Folgejahres an.

Um die Berechnung der Kalenderwoche zu testen, habe ich eine kleine Konsolenanwendung geschrieben, die für alle Kalenderwochen der Jahre 1999 bis 2009 das Startdatum berechnet, ausgibt und in eine Textdatei schreibt.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Windows.Forms` importieren.

```

// Textdatei öffnen
string fileName = Path.Combine(Application.StartupPath, "Kalenderwochen.txt");
StreamWriter sr = new StreamWriter(fileName, false, Encoding.Default);

// Startdatums-Werte für die Kalenderwochen zwischen 1999 und 2009 berechnen
for (int year = 1999; year < 2010; year++)
{
    for (int calendarWeek = 1; calendarWeek < 54;
        calendarWeek++)
    {
        // Das Startdatum dieser Kalenderwoche ermitteln und ausgeben
        DateTime date = GetGermanCalendarWeekStartDate(calendarWeek, year);

        Console.WriteLine("KW {0}/{1}: {2}", calendarWeek, year,
            date.ToLongDateString());
        sr.WriteLine("KW {0}/{1}: {2}", calendarWeek, year,
            date.ToLongDateString());
    }
}

```

Listing 71: Testweises Berechnen der Kalenderwochen und der Startdatums-Werte dieser Wochen

```

}

// StreamWriter schließen
sr.Close();

```

Listing 71: Testweises Berechnen der Kalenderwochen und der Startdatums-Werte dieser Wochen (Forts.)

Abbildung 32 und eine stichprobenartige Überprüfung der erzeugten Daten beweisen, dass die Berechnung in Ordnung ist 😊.

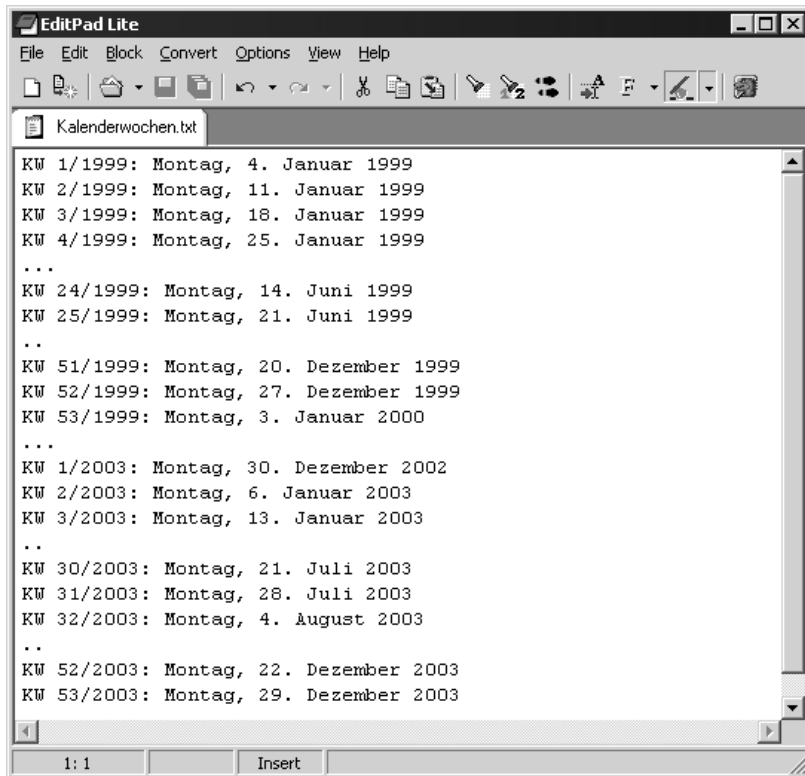


Abbildung 32: Die vom Testprogramm erzeugte Textdatei in einer verkürzten Fassung

51 Anzahl der Kalenderwochen eines Jahres bestimmen

Die Anzahl der Kalenderwochen eines Jahres können Sie über die Methode `GetCalendarWeek2` aus dem Rezept 49 ganz einfach bestimmen, indem Sie die Kalenderwoche des 31.12. des Jahres ermitteln. Handelt es sich um die Woche 1, ermitteln Sie die Kalenderwoche des Datums, das eine Woche zurückliegt. Die Methode `GetCalendarWeekCount` in Listing 72 nimmt Ihnen diese Arbeit ab. Diese Methode erfordert den Import der Namensräume `System` und `System.Globalization`.

```

public static int GetCalendarWeekCount(int year)
{
    // Kalenderwoche des 31.12. des Jahres ermitteln
    DateTime baseDate = new DateTime(year, 12, 31);
    CalendarWeek calendarWeek = GetCalendarWeek2(baseDate);

    // Wenn dieser Tag in die Woche 1 des neuen Jahres fällt, die Kalenderwoche
    // des um eine Woche reduzierten Datums ermitteln
    if (calendarWeek.Week == 1)
        return GetCalendarWeek2(baseDate.AddDays(-7)).Week;

    // Ergebnis zurückgeben
    return calendarWeek.Week;
}

```

Listing 72: Methode zur Ermittlung der Kalenderwochen-Anzahl eines Jahres

52 Systemdatum und -zeit setzen

Das Setzen des Systemdatums und der Systemzeit ist nicht mit .NET-Klassen möglich. Die entsprechenden Eigenschaften der `DateTime`-Struktur sind schreibgeschützt und `DateTime` besitzt keine Methoden zum Schreiben der Systemdatumswerte.

Das Systemdatum und die Systemzeit können Sie aber über die API-Funktion `SetSystemTime` setzen. Diese Funktion erwartet am einzigen Argument eine Instanz der API-Struktur `SYSTEMTIME`, die einzelne Felder für die Bestandteile eines Datums besitzt. Diese Struktur füllen Sie mit den gewünschten Werten und rufen damit `SetSystemTime` auf. Dabei müssen Sie beachten, dass diese Funktion die Zeitangabe im UTC-Format (Coordinated Universal Time) erwartet. Diese Zeitangabe, die früher als *Greenwich Mean Time (GMT)* bezeichnet wurde, bezieht sich auf die aktuelle Zeit in der englischen Ortschaft Greenwich und wird als internationale Basis für Zeitangaben verwendet. Die Zeit in Deutschland besitzt im Winter einen Offset von einer und im Sommer einen Offset von zwei Stunden zu UTC. Ein Datum können Sie ganz einfach über die Methode `ToUniversalTime` eines `DateTime`-Objekts in das UTC-Format umrechnen.

Die Methoden in Listing 76 helfen Ihnen dabei, das Systemdatum und die Systemzeit zu setzen. Zunächst habe ich die Struktur `SYSTEMTIME` und die API-Funktion `SetSystemTime` deklariert. `SYSTEMTIME` ist mit dem Attribut `StructLayout` mit dem Argument `LayoutKind.Sequential` versehen, damit die (.NET-)Struktur im Speicher als normale C-Struktur angelegt wird, bei der die einzelnen Elemente (anders als bei .NET) sequenziell hintereinander im Speicher positioniert werden. Die einzelnen Felder sind entsprechend der originalen Deklaration der Struktur (in *winbase.h*) als `short` deklariert. `SetSystemTime` bekommt einen Zeiger auf eine Instanz dieser Struktur übergeben, weswegen das entsprechende Argument bei der Deklaration dieser Funktion in C# mit `ref` deklariert werden muss.

Zum Kompilieren der im Folgenden beschriebenen Methoden müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```

/* Struktur für das Setzen der Systemzeit */
[StructLayout(LayoutKind.Sequential)]
private struct SYSTEMTIME
{
    public short uYear;
    public short uMonth;
    public short uDayOfWeek;
    public short uDay;
    public short uHour;
    public short uMinute;
    public short uSecond;
    public short uMilliseconds;
}

```

Listing 73: Deklaration der benötigten API-Struktur und -Funktion

Dann folgt die Deklaration der benötigten API-Funktionen. Neben `SetSystemTime` habe ich auch `FormatMessage` und die für das Lesen einer Systemmeldung verwendete Konstante deklariert um Fehlermeldungen auslesen zu können:

```

[DllImport("kernel32.dll", SetLastError=true)]
static extern int SetSystemTime(ref SYSTEMTIME lpSystemTime);

[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
    int dwMessageId, int dwLanguageId, System.Text.StringBuilder lpBuffer,
    int nSize, string [] Arguments);
private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;

```

Listing 74: Deklaration der benötigten API-Funktionen und -Konstanten

Die private Methode `setSystemTime` in Listing 75 übernimmt den Aufruf der API-Funktion. Tritt beim Aufruf ein Fehler auf, liest die Methode den Fehlercode aus, ermittelt dazu über `FormatMessage` eine passende Fehlerbeschreibung und wirft eine Ausnahme. Ich habe diese Methode implementiert um die aufwändige Fehlerbehandlung nicht in jede der öffentlichen Methoden integrieren zu müssen.

```

private static void setSystemTime(SYSTEMTIME sysTime)
{
    // Systemzeit setzen
    if (SetSystemTime(ref sysTime) == 0)
    {
        // Fehler bei der Ausführung: Den letzten Windows-Fehlercode
        // auslesen, in eine Beschreibung umwandeln und damit
        // eine Ausnahme werfen
        int apiError = Marshal.GetLastWin32Error();
        StringBuilder errorMessage = new StringBuilder(1024);
        FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, (IntPtr)0, apiError,

```

Listing 75: Methode zum Setzen der Systemzeit mit Fehlerbehandlung

```

        0, errorMessage, 1024, null);
    throw new Exception(errorMessage.ToString());
}
}

```

Listing 75: Methode zum Setzen der Systemzeit mit Fehlerbehandlung (Forts.)

Die Methode `SetSystemDateTime` in Listing 76 bekommt nun ein `DateTime`-Objekt übergeben und setzt das komplette Datum inklusive der Zeit. Dabei wird das übergebene Datum zunächst in UTC umgerechnet. Dann erzeugt diese Methode eine `SYSTEMTIME`-Struktur, liest die Bestandteile des Datums über die entsprechenden Eigenschaften des `DateTime`-Objekts und schreibt diese in die Struktur. Diese Struktur wird dann an die private Methode `setSystemTime` übergeben.

Die Methode `SetSystemDate` arbeitet ähnlich, setzt aber nur den Datumsteil des Systemdatums. Der Zeitanteil wird dazu einfach aus dem aktuellen Datum ausgelesen und in die Struktur geschrieben. `SetSystemTime` setzt auf eine ähnliche Weise nur den Zeitanteil. Die übergebene Zeit wird dazu natürlich wieder in eine UTC-Zeit umgerechnet.

```

/* Methode zum Setzen des Systemdatums inklusive Zeit */
public static void SetSystemDateTime(DateTime date)
{
    // Zeit in UTC umrechnen
    date = date.ToUniversalTime();

    // SYSTEMTIME-Struktur erzeugen und initialisieren
    SYSTEMTIME sysTime = new SYSTEMTIME();
    sysTime.uYear = (short)date.Year;
    sysTime.uMonth = (short)date.Month;
    sysTime.uDay = (short)date.Day;
    sysTime.uHour = (short)date.Hour;
    sysTime.uMinute = (short)date.Minute;
    sysTime.uSecond = (short)date.Second;
    sysTime.uMilliseconds = (short)date.Millisecond;

    // Systemzeit über die private Methode setzen
    setSystemTime(sysTime);
}

/* Methode zum Setzen des Systemdatums ohne Zeit */
public static void SetSystemDate(DateTime date)
{
    // SYSTEMTIME-Struktur erzeugen und initialisieren
    SYSTEMTIME sysTime = new SYSTEMTIME();
    sysTime.uYear = (short)date.Year;
    sysTime.uMonth = (short)date.Month;
    sysTime.uDay = (short)date.Day;
    sysTime.uHour = (short)DateTime.Now.Hour;
    sysTime.uMinute = (short)DateTime.Now.Minute;

```

Listing 76: Methoden zum Setzen des Systemdatums und der Systemzeit

```

    sysTime.uSecond = (short)DateTime.Now.Second;
    sysTime.uMilliseconds = (short)DateTime.Now.Millisecond;

    // Systemzeit über die private Methode setzen
    setSystemTime(sysTime);
}

/* Methode zum Setzen der Systemzeit (ohne Datum) */
public static void SetSystemTime(DateTime date)
{
    // Zeit in UTC umrechnen
    date = date.ToUniversalTime();

    // SYSTEMTIME-Struktur erzeugen und initialisieren
    SYSTEMTIME sysTime = new SYSTEMTIME();
    sysTime.uYear = (short)DateTime.Now.Year;
    sysTime.uMonth = (short)DateTime.Now.Month;
    sysTime.uDay = (short)DateTime.Now.Day;
    sysTime.uHour = (short)date.Hour;
    sysTime.uMinute = (short)date.Minute;
    sysTime.uSecond = (short)date.Second;
    sysTime.uMilliseconds = (short)date.Millisecond;

    // Systemzeit über die private Methode setzen
    setSystemTime(sysTime);
}

```

Listing 76: Methoden zum Setzen des Systemdatums und der Systemzeit (Forts.)

Voraussetzung für die Ausführung der Methoden zum Setzen der Systemzeit ist, laut der Dokumentation der `SetSystemTime`-Funktion, dass das Benutzerkonto, unter dem die Methoden ausgeführt werden, unter Windows NT, 2000 und XP das Privileg `SE_SYSTEMTIME_NAME` besitzt. Dieses Privileg können Sie über die lokalen Sicherheitsrichtlinien in der Systemsteuerung vergeben. Dort wird das Privileg als *Ändern der Systemzeit* bezeichnet. Für den Fall, dass Sie dieses Privileg nicht jedem Benutzer der Anwendung zuordnen können, können Sie die Anwendung auch vor der Ausführung der Methoden zum Setzen der Systemzeit mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 211 beschreibe.

53 Atomuhr-Datum inklusive Uhrzeit von einem NIST-Server abfragen

Das amerikanische Institut NIST (National Institute of Standards and Technologie), das hauptsächlich Messverfahren und Standards für die amerikanische Industrie entwickelt (siehe www.nist.gov), betreibt einige hochgenaue Atomuhren, deren Datum und Uhrzeit in den USA als Standard gelten. Einige von NIST betriebene TCP/UDP-Server liefern das Datum dieser Uhren als UTC²-Zeit. Eine Liste dieser Server finden Sie an der Adresse www.boulder.nist.gov/timefreq/service/time-servers.html.

-
2. UTC = Coordinated Universal Time. Eine UTC-Zeitangabe, die früher als *Greenwich Mean Time (GMT)* bezeichnet wurde, bezieht sich auf die aktuelle Zeit in der englischen Ortschaft Greenwich und wird als internationale Basis für Zeitangaben verwendet

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Diese Server implementieren u. a. das »Daytime Protocol« des W3C (RFC 867) über den Port 13 (siehe www.boulder.nist.gov/timefreq/service/its.htm). Dieses Protokoll sieht lediglich vor, dass ein Server auf dem Port 13 auf eine Verbindung wartet und das aktuelle Datum inklusive Uhrzeit in einer beliebigen Textform zurücksendet. Die NIST-Server verwenden dazu ein festgelegtes Format in der folgenden Form:

```
JJJJJ YR-MO-DA HH:MM:SS TT L H msADV UTC(NIST) OTM
```

- ▶ *JJJJ* ist ein (für uns uninteressantes) modifiziertes Julianisches Datum,
- ▶ *YR-MO-DA* ist das normale Datum in der Form *Jahr-Monat-Tag*,
- ▶ *HH:MM:SS* ist die Zeit,
- ▶ *TT* sagt aus, ob die USA gerade die Standard- oder die Sommerzeit verwenden,
- ▶ *L* sagt mit 0 oder 1 aus, ob im aktuellen Monat am letzten Tag eine extra Sekunde hinzugefügt wird (wegen der Schaltjahres-Berechnung),
- ▶ *H (Health)* gibt die »Gesundheit« des Servers an. Bei einer 0 ist der Server in Ordnung und liefert die genaue Zeit, bei 1 kann es sein, dass die Zeit mit einer Ungenauigkeit von fünf Sekunden zurückgegeben wird, weil der Server stark belastet ist, bei 2 ist die Differenz schon zehn Sekunden, bei 4 ist ein schwerer Fehler aufgetreten,
- ▶ *msADV* gibt einen Offset in Millisekunden an, den der NIST-Server auf die ermittelte Zeit wegen erwarteter Netzwerk-Zeitverzögerungen aufaddiert (zurzeit 50),
- ▶ *UTC(NIST)* gibt an, dass es sich um eine UTC-Zeit von NIST handelt,
- ▶ *OTM* ist immer ein Asterisk (*), der beim Empfang verwendet werden kann, um den genauen Zeitpunkt zu bestimmen. Die Zeitangabe im String bezieht sich (mit dem Offset) immer auf den Empfang des OTM-Zeichens.

Über eine Instanz der Klasse `TcpClient` ist es nun relativ einfach, einen NIST-Server abzufragen. Listing 77 setzt dies um. Die Methode `GetNISTTime` fragt dazu gleich mehrere NIST-Server ab, um das genaue Datum auch dann zu erhalten, wenn ein Server nicht erreichbar ist oder einen Gesundheitsstatus größer 0 zurückliefert. Die Abfrage erfolgt in einer Schleife, die alle im anfänglich initialisierten Array `servers` verwalteten Server-Adressen durchgeht. Die NIST-Server, die ich in diesem Array aufgenommen habe, haben sich in meinen Tests als zuverlässig erwiesen. Die Abfrage der Server erfolgt über eine `TcpClient`-Instanz, die natürlich vor der Schleife erzeugt wird. Der Empfangs-Timeout dieser Instanz wird dabei auf eine Sekunde (1000 Millisekunden) gesetzt, damit die `Read`-Methode nicht unnötig lange auf die Daten wartet. Der standardmäßige Timeout von 0 würde ansonsten bewirken, dass `Read` ganze 20 Sekunden auf Daten wartet (das ist jedenfalls die empirisch ermittelte Standard-Einstellung auf meinem Rechner, das Defaultverhalten ist nicht dokumentiert). Leider besteht nach meinen Recherchen keine direkte Möglichkeit einen Timeout für den Verbindungsaufbau (über die `Connect`-Methode) zu definieren. `Connect` wartet bei einem unerreichbaren Host standardmäßig (nach meinen Versuchen und Informationen aus Newsgroups) zehn Sekunden.

In der Schleife versucht das Programm zunächst über die `Connect`-Methode der `TcpClient`-Instanz zum Port 13 des aktuellen Servers eine Verbindung aufzubauen. Schlägt diese fehl (was über die Ausnahmebehandlung erkannt wird), speichert `GetNISTTime` den aufgetretenen Fehler in der Stringvariable `errors`. Die Schleife macht dann beim nächsten Server weiter.

Konnte die Verbindung aufgebaut werden, referenziert `GetNISTTime` den `NetworkStream` der `TcpClient`-Instanz und überprüft (zur Sicherheit ...), ob dieser gelesen und geschrieben werden kann (was aber immer der Fall sein sollte). Da ein NIST-Server beim Daytime-Protokoll direkt nach dem Verbindungsaufbau Daten sendet, liest das Programm den Stream nach dem Verbindungsaufbau sofort aus und konvertiert die empfangenen Bytes in einen ASCII-String. Dieser String wird dann in einzelne Token (Bestandteile) zerlegt, wobei das Leerzeichen als Trennzeichen verwendet wird. Vor dem Auswerten der Token wird überprüft, ob die Anzahl in Ordnung ist: Beim Testen ist es vorgekommen, dass ein Server statt dem erwarteten Zeit-String eine Folge von 0-Zeichen gesendet hat.

Das Token mit dem Index 5 ist dann der Health-Status. Dieser wird zunächst daraufhin überprüft, ob er den Wert »0« speichert (was ja auf die Rückgabe der genauen Zeit hinweist). Ist dies nicht der Fall, speichert `GetNISTTime` dies als Fehler in der Variablen `errors` und führt die Schleife weiter aus. Ist der Health-Status in Ordnung, werden das Datums- und das Zeit-Token in ihre Bestandteile zerlegt. Mit diesen Datums- bzw. Zeiteilen erzeugt `GetNISTTime` dann eine neue `DateTime`-Instanz und gibt diese in die lokale Zeit konvertiert zurück (womit die Schleife beendet ist). Tritt beim Lesen der Daten eine Ausnahme ein (z.B. weil der Timeout abgelaufen ist), wird die Fehlermeldung in der Variablen `errors` abgelegt. Für den Fall, dass alle Server unerreichbar sind, einen Health-Status ungleich »0« oder ungültige Daten zurückliefern oder nicht rechtzeitig Daten senden, erzeugt das Programm am Ende der Methode eine Ausnahme mit den ermittelten Fehlern als Meldung.

Die `TcpClient`-Instanz wird übrigens in der Schleife immer wieder neu erzeugt weil die `Close`-Methode, die die Verbindung schließt, leider auch die internen Ressourcen freigibt, sodass ein zweites Öffnen einer Verbindung nicht mehr möglich ist.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.Text` und `System.Net.Sockets` importieren.

```
public static System.DateTime GetNISTTime()
{
    // Variable für Fehlermeldungen
    string errors = null;

    // Array für die abzufragenden Server
    string[] servers = {"time-a.nist.gov", "time-b.nist.gov",
        "time.nist.gov", "utcnist.colorado.edu", "nist1.datum.com"};

    // Schleife, in der die Server abgefragt werden, bis das Ergebnis
    // in Ordnung ist
    for (int i = 0; i < servers.Length; i++)
    {
```

Listing 77: Methode zur Ermittlung des offiziellen Datums von NIST-Servern

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

TcpClient tcpClient = null;
try
{
    // TcpClient erzeugen und den Empfangs-Timeout auf eine Sekunde
    // setzen
    tcpClient = new TcpClient();
    tcpClient.ReceiveTimeout = 1000;

    // Versuch, zum aktuellen Server eine Verbindung aufzubauen
    tcpClient.Connect(servers[i], 13);

    // Den NetworkStream referenzieren
    NetworkStream networkStream = tcpClient.GetStream();

    string result = null;
    if (networkStream.CanWrite && networkStream.CanRead)
    {
        // Das Ergebnis empfangen und in ASCII konvertieren
        byte[] bytes = new byte[tcpClient.ReceiveBufferSize];
        try
        {
            networkStream.Read(bytes, 0,
                (int) tcpClient.ReceiveBufferSize);
            result = Encoding.ASCII.GetString(bytes);
        }
        catch (Exception ex)
        {
            // Fehler dokumentieren
            if (errors != null) errors += "\r\n";
            errors += "Fehler bei der Abfrage von " + servers[i] +
                ": " + ex.Message;
        }
    }

    if (result != null)
    {
        // Das Ergebnis, das die Form JJJJJ YR-MO-DA HH:MM:SS TT L H msADV
        // UTC(NIST) OTM besitzt, in einzelne Token aufsplitten
        string[] token = result.Split(' ');

        // Anzahl der Token überprüfen
        if (token.Length >= 6)
        {
            // Den Health-Status auslesen und überprüfen
            string health = token[5];
            if (health == "0")
            {
                // Alles OK: Datums- und Zeitangaben auslesen
                string[] dates = token[1].Split('-');
                string[] times = token[2].Split(':');
            }
        }
    }
}

```

Listing 77: Methode zur Ermittlung des offiziellen Datums von NIST-Servern (Forts.)

```

        // DateTime-Instanz mit diesen Daten erzeugen
        System.DateTime utcDate =
            new System.DateTime(Int32.Parse(dates[0]) + 2000,
                Int32.Parse(dates[1]), Int32.Parse(dates[2]),
                Int32.Parse(times[0]), Int32.Parse(times[1]),
                Int32.Parse(times[2]));

        // Lokale Zeit berechnen und zurückgeben
        return TimeZone.CurrentTimeZone.ToLocalTime(utcDate);
    }
    else
    {
        // Fehler dokumentieren
        if (errors != null) errors += "\r\n";
        errors += "Fehler bei der Abfrage von '" + servers[i] +
            "': Der Health-Status ist " + health;
    }
}
else
{
    // Fehler dokumentieren
    if (errors != null) errors += "\r\n";
    errors += "Fehler bei der Abfrage von '" + servers[i] +
        "': Die Anzahl der Token ist kleiner als 6";
}
}
}
catch (Exception ex)
{
    // Fehler dokumentieren
    if (errors != null) errors += "\r\n";
    errors += "Fehler bei der Abfrage von '" +
        servers[i] + "': " + ex.Message;
}
finally
{
    try
    {
        // TcpClient schließen
        tcpClient.Close();
    }
    catch {}
}
}

// Wenn die Methode hier ankommt, sind bei allen Abfragen
// Fehler aufgetreten, also eine Ausnahme werfen
throw new Exception(errors);
}

```

Listing 77: Methode zur Ermittlung des offiziellen Datums von NIST-Servern (Forts.)

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die Methode erfordert, dass eine Internetverbindung besteht und dass eine eventuelle Firewall die Anforderung des Programms zulässt.

Das Beispiel in Listing 78 liest das offizielle Datum ein und setzt damit das Systemdatum. Zum Setzen des Systemdatums wird die Methode `SetSystemDateTime` aus dem Rezept 52 eingesetzt.

```
Console.WriteLine("Hole das offizielle Datum von einem NIST-Server ...");

try
{
    // NIST-Server abfragen
    DateTime officialDate = GetNISTTime();

    // Systemdatum setzen
    SetSystemDateTime(officialDate);

    Console.WriteLine("Systemdatum auf den " + officialDate.ToString() +
        " gesetzt");
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Lesen oder Setzen des Systemdatums: {0}",
        ex.Message);
}
```

Listing 78: Setzen des Systemdatums auf das von einem NIST-Server abgefragte offizielle Datum



Abbildung 33: Das Programm hat (auf wundersame Weise ...) das Systemdatum auf das offizielle Datum gesetzt (und ich habe endlich einmal die genaue Zeit ...)

54 Datum auf Gültigkeit überprüfen

Datumswerte, die vom Anwender eingegeben werden, müssen eigentlich immer auf Gültigkeit überprüft werden. Das .NET Framework bietet dazu keine direkte Möglichkeit (außer in der Visual Basic .NET Assembly mit der Methode `IsDate`). Sie können einen String sehr einfach auf Gültigkeit überprüfen, indem Sie in einer Ausnahmebehandlung versuchen, den String über die `Parse`-Methode der `DateTime`-Struktur in ein Datum zu konvertieren. Listing 79 setzt dies in zwei Methoden um. Die erste Variante der Methode `IsDate` überprüft einen String daraufhin, ob er mit der aktuellen Systemeinstellung in ein Datum konvertiert werden kann. Dazu wird die `Parse`-Methode einfach in einer Ausnahmebehandlung aufgerufen.

Die zweite Variante dieser Methode erwartet am zweiten Argument ein `CultureInfo`-Objekt, das die Kultur repräsentiert, die für die Datums-Konvertierung verwendet werden soll. Diese Methode gibt das `CultureInfo`-Objekt einfach am zweiten Argument der `Parse`-Methode weiter, damit der Datums-String unter Berücksichtigung dieser Kultur geparkt wird.

Zum Kompilieren der `IsDate`-Methoden müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```
public static bool IsDate(string dateString)
{
    // Versuch, den String in ein Datum zu konvertieren
    try
    {
        DateTime.Parse(dateString);
        return true;
    }
    catch
    {
        return false;
    }
}

public static bool IsDate(string dateString, CultureInfo culture)
{
    // Versuch, den String in ein Datum zu konvertieren
    try
    {
        DateTime.Parse(dateString, culture);
        return true;
    }
    catch
    {
        return false;
    }
}
```

Listing 79: Methoden zur Überprüfung der Gültigkeit eines Datums-Strings

Listing 80 zeigt eine Beispiel-Anwendung der `IsDate`-Methoden, bei der der Anwender ein Datum im aktuellen Systemformat und im englischen Format eingeben soll. Das Programm erfordert den Import des Namensraums `System.Globalization`.

```
// Den Anwender ein Datum im aktuellen Format eingeben lassen
Console.WriteLine("Geben Sie ein Datum im aktuellen " +
    "System-Format ein: ");
string dateString = Console.ReadLine();

// Überprüfen, ob dieses Datum gültig ist
```

Listing 80: Beispielprogramm zur Ermittlung der Gültigkeit eingegebener Datumswerte

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

if (IsDate(dateString))
{
    Console.WriteLine("Das Datum ist gültig: {0}", DateTime.Parse(dateString));
}
else
    Console.WriteLine("Das Datum ist ungültig");

// Den Anwender ein Datum im englischen Format eingeben lassen
Console.Write("Geben Sie ein Datum im englischen Format ein: ");
dateString = Console.ReadLine();

// Überprüfen, ob dieses Datum gültig ist
if (IsDate(dateString, CultureInfo.CreateSpecificCulture("en")))
{
    Console.WriteLine("Das Datum ist gültig: {0}", DateTime.Parse(dateString,
        CultureInfo.CreateSpecificCulture("en")));
}
else
    Console.WriteLine("Das Datum ist ungültig");

```

Listing 80: Beispielprogramm zur Ermittlung der Gültigkeit eingegebener Datumswerte (Forts.)

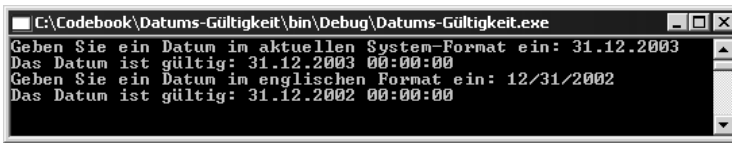


Abbildung 34: Das Beispielprogramm hat die eingegebenen Datumswerte überprüft

55 Datum normalisieren

Viele Anwender wollen die Möglichkeit haben, ein Datum in der Form *ddmmyy*, *ddmmyyy* oder auch nur als *ddmm* einzugeben (z.B. *31122003* für den 31.12.2003 oder *3112* für den 31.12. des aktuellen Jahres). Die `Parse`-Methode der `DateTime`-Struktur erwartet ein Datum aber im aktuellen Systemformat oder im Format des am zweiten Argument übergebenen Objekts, das die `IFormatProvider`-Schnittstelle implementiert (normalerweise ist das ein `CultureInfo`-Objekt).

Die Methode `NormalizeDate` in Listing 81 ermöglicht die Eingabe von Datumswerten in typischen (Anwender-)Formaten und wandelt den übergebenen Datums-String in ein `DateTime`-Objekt um. Die folgenden Formate werden unterstützt:

- das normale Systemformat,
- *mm.dd[.]*,
- *mm[.]*,
- *ddmmyy[y][y]*,
- *ddmm* und
- *d[d]*.

Unvollständige Eingaben werden automatisch um das aktuelle Jahr bzw. das Jahr und den aktuellen Monat ergänzt. Aus 3112 wird im Jahr 2003 z. B. das Datum 31.12.2003.

`NormalizeDate` entfernt zunächst eventuelle (störende) Whitespace-Zeichen am Anfang und am Ende des Datumsstrings und versucht dann, das Datum direkt zu konvertieren (falls es im Systemformat eingegeben wurde). Der `Parse`-Methode der `DateTime`-Struktur wird am zweiten Argument kein Format-Provider übergeben, damit die aktuell eingestellte Kultur verwendet wird.

Schlägt diese Konvertierung fehl, versucht `NormalizeDate` aus dem eingegebenen Datums-String einen gültigen Datumsstring zu erzeugen. Bei der Abfrage der Datumsteile wird zwar das aktuelle System-Datumstrennzeichen berücksichtigt (damit eine Eingabe prinzipiell z. B. auch auf einem englischen Rechner ausgewertet werden kann), der neue Datumsstring wird aber explizit im deutschen Format zusammengesetzt. Bei der Konvertierung am Ende der Methode übergibt das Programm dann auch am zweiten Argument der `Parse`-Methode ein neues `CultureInfo`-Objekt für die deutsche Kultur. So ist sichergestellt, dass der Datums-String auch auf Rechnern oder in Programmen konvertiert werden kann, die einer anderen Kultur zugeordnet sind.

`NormalizeDate` erreicht damit eine gewisse Globalisierung. Wenn das Datum allerdings nicht im Systemformat eingegeben wird, erwartet `NormalizeDate` die oben beschriebenen eher deutschen Eingabeformate (bei denen der Tag am Anfang und der Monat an zweiter Stelle steht). Ich habe versucht, eine Variante dieser Methode zu entwickeln, die die unterschiedlichen Kulturen berücksichtigt, sodass auf einem englischen Rechner das Datum z. B. auch im Format `mmddyyyy` eingegeben werden kann. Diese Variante wurde aber noch vor der Fertigstellung aufgrund der vielen verschiedenen Möglichkeiten viel zu komplex, sodass ich sie verworfen habe (damit noch Zeit fürs Snowboarden übrig blieb 😊).

`NormalizeDate` erfordert den Import des Namensraums `System`.

```
public static DateTime NormalizeDate(string dateString)
{
    // Whitespaces entfernen
    dateString = dateString.Trim();

    // Versuch, das Datum direkt zu konvertieren
    try
    {
        return DateTime.Parse(dateString);
    }
    catch {}

    // Das aktuelle Datumstrennzeichen ermitteln
    string dateSeparator =
        CultureInfo.CurrentCulture.DateTimeFormat.DateSeparator;

    if (dateString.IndexOf(dateSeparator) == -1)
    {
```

Listing 81: Methode zur Normalisierung eines Datums

```

// Wenn kein Punkt vorkommt: Versuch ein Datum im Format
// ddmmyy[yy], ddmm oder d[d] zu erkennen
if (dateString.Length >= 6 && dateString.Length <= 8)
{
    // ddmmyy, ddmmyy oder ddmmyyy
    dateString = dateString.Substring(0, 2) + "." +
        dateString.Substring(2, 2) + "." +
        dateString.Substring(4, dateString.Length - 4);
}
else if (dateString.Length == 4)
{
    // ddmm
    dateString = dateString.Substring(0, 2) + "." +
        dateString.Substring(2, 2) + "." + DateTime.Now.Year;
}
else if (dateString.Length <= 2)
{
    // d oder dd
    if (dateString.Length == 1)
        dateString = "0" + dateString;
    dateString = dateString.Substring(0, 2) + "." +
        DateTime.Now.Month + "." + DateTime.Now.Year;
}
}
else
{
    // Eingabe des Datums mit Punkten in der Form dd.mm oder
    // dd. Zunächst einen eventuellen rechten Punkt entfernen
    while (dateString.EndsWith(dateSeparator))
        dateString = dateString.Substring(0, dateString.Length - 1);

    // Ein im Format dd oder dd.mm angegebenes Datum um den
    // aktuellen Monat und das aktuelle Jahr ergänzen
    if (dateString.Length <= 2)
    {
        // dd.
        dateString = dateString + "." + DateTime.Now.Month + "." +
            DateTime.Now.Year;
    }
    else
    {
        // dd.mm[.]
        dateString = dateString + "." + DateTime.Now.Year;
    }
}

// Versuch, das ermittelte deutsche Datum zu konvertieren
return DateTime.Parse(dateString, CultureInfo.CreateSpecificCulture("de"));
}

```

Listing 81: Methode zur Normalisierung eines Datums (Forts.)

Listing 82 zeigt ein kleines Testprogramm für die `NormalizeDate`-Methode, bei dem der Anwender in einer Schleife ein Datum mehrfach eingeben kann.

```
string dateString = null;
do
{
    // Den Anwender einen Datums-String eingeben lassen
    Console.Write("Geben Sie ein Datum ein (Beenden mit leerer Eingabe): ");
    dateString = Console.ReadLine();

    if (dateString != "")
    {
        try
        {
            // Das Datum normalisieren
            DateTime date = NormalizeDate(dateString);

            // Das normalisierte Datum ausgeben
            Console.WriteLine(date.ToString());
            Console.WriteLine();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
} while (dateString != "");
```

Listing 82: Testprogramm für die `NormalizeDate`-Methode

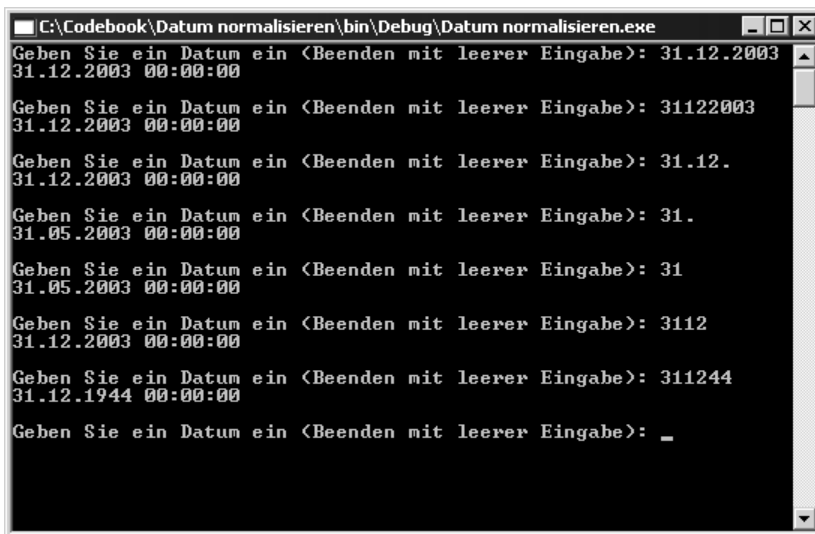


Abbildung 35: Das Testprogramm für die `NormalizeDate`-Methode

56 Deutsche Feiertage und andere besondere Tage berechnen

Das .NET Framework bietet keine Möglichkeiten, die deutschen Feiertage und die besonderen Tage (wie z. B. Rosenmontag) zu berechnen. Wenn Sie diese (z. B. für ein eigenes Kalender-Steuerelement) ermitteln wollen, müssen Sie selbst programmieren.

Einige Feiertage sind fest definiert:

- ▶ 1. Januar: Neujahr,
- ▶ 6. Januar: Heilige drei Könige (nur Baden-Württemberg, Bayern und Sachsen-Anhalt),
- ▶ 1. Mai: Maifeiertag (Tag der Arbeit),
- ▶ 15. August: Maria Himmelfahrt (nur Saarland, Bayern und Gemeinden mit überwiegend katholischer Bevölkerung),
- ▶ 3. Oktober: Tag der deutschen Einheit,
- ▶ 31. Oktober: Reformationstag (nur Brandenburg, Mecklenburg-Vorpommern, Sachsen, Sachsen-Anhalt und Thüringen),
- ▶ 1. November: Allerheiligen (nur Baden-Württemberg, Bayern, Nordrhein-Westfalen, Rheinland-Pfalz und Saarland),
- ▶ 25. Dezember: Erster Weihnachtsfeiertag,
- ▶ 26. Dezember: Zweiter Weihnachtsfeiertag.

Hinzu kommen einige besondere Tage, die keine Feiertage sind:

- ▶ Heiliger Abend: 24. Dezember,
- ▶ Valentinstag: 14. Februar.

Einige der beweglichen Feiertage beziehen sich auf den Ostersonntag:

- ▶ Karfreitag: – 2 Tage,
- ▶ Ostersonntag: +/- 0 Tage,
- ▶ Ostermontag: + 1 Tag,
- ▶ Christi Himmelfahrt: + 39 Tage,
- ▶ Pfingstsonntag: + 49 Tage,
- ▶ Pfingstmontag: + 50 Tage,
- ▶ Fronleichnam: + 60 Tage (Baden-Württemberg, Bayern, Hessen, Nordrhein-Westfalen, Rheinland-Pfalz, Saarland, Sachsen, Thüringen, außerdem Gemeinden mit überwiegend katholischer Bevölkerung).

Daneben existieren noch bewegliche, auf Ostern bezogene besondere Tage, die keine Feiertage sind

- ▶ Aschermittwoch: – 46 Tage,
- ▶ Rosenmontag: – 48 Tage,
- ▶ Gründonnerstag: – 3 Tage.

Andere bewegliche Feiertage beziehen sich auf Weihnachten:

- ▶ 4. Advent: der Sonntag vor dem 25. Dezember,
- ▶ 1. bis 3. Advent: die jeweils vorigen Sonntage,
- ▶ Totensonntag: der Sonntag vor dem 1. Advent,
- ▶ Buß- & Bettag: der Mittwoch vor dem Totensonntag (nur Sachsen).

Das einzige Problem ist also die Berechnung des Datums für den Ostersonntag. Per Definition ist Ostern auf der nördlichen Halbkugel der Sonntag nach dem ersten Vollmond nach der Tag- und Nachtgleiche. Der Mathematiker Carl Friedrich Gauß hat dazu als Erster recht komplexe Formeln entwickelt. Leider arbeiten diese nicht korrekt, sodass für einige Jahre nachkorrigiert werden muss. Im Internet finden Sie eine Vielzahl weiterer Algorithmen zur Berechnung des Ostersonntags.

Ich habe den Algorithmus zur Oster-Berechnung natürlich nicht selbst entwickelt (ich bin kein Mathematiker), sondern im Internet recherchiert. Der Algorithmus von Ronald W. Mallen soll nach Informationen aus dem Internet der definitive Algorithmus sein, der die Ostersonntage für den gregorianischen Kalender und die Jahre 1583 bis 4099 korrekt berechnet (vgl. www.assa.org.au/bulletin/bennecma.pdf). Sie finden diesen Algorithmus an der Adresse www.assa.org.au/edm.html. Ich habe diesen Algorithmus unverändert in der Methode `GetEasterSundayDate` nach C# übersetzt. Ronald W. Mallen hat die Veröffentlichung in dieser Übersetzung netterweise erlaubt. Für den Fall, dass ein ungültiges Jahr übergeben wird, generiert `GetEasterSundayDate` eine Ausnahme.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Collections` importieren.

```

/* Methode zur Berechnung des Datums des Ostersonntags.
 * Nach dem Original von Ronald W. Mallen (www.assa.org.au/edm.html) */
public static DateTime GetEasterSundayDate(int year)
{
    // Überprüfen, ob das Jahr für die Osterberechnung gültig ist
    if (year < 1583 || year > 4099)
        throw new Exception("Das Jahr muss zwischen 1583 " +
            "und 4099 liegen");

    int firstDigits, remainder19, temp; // Zwischenergebnisse
    int tA, tB, tC, tD, tE; // Tabellenergebnisse A bis E

```

Listing 83: Methode zur Berechnung des Datums des Ostersonntags

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
firstDigits = year / 100; // die ersten zwei Ziffern
remaining19 = year % 19; // Rest von year / 19

// PFM-Datum berechnen
temp = (firstDigits - 15) / 2 + 202 - 11 * remaining19;

switch (firstDigits)
{
    case 21:
    case 24:
    case 25:
    case 27:
    case 28:
    case 29:
    case 30:
    case 31:
    case 32:
    case 34:
    case 35:
    case 38:
        temp -= 1;
        break;
    case 33:
    case 36:
    case 37:
    case 39:
    case 40:
        temp -= 2;
        break;
}

temp = temp % 30;

tA = temp + 21;
if (temp == 29)
    tA = tA - 1;
if (temp == 28 && remaining19 > 10)
    tA = tA - 1;

// Nächsten Sonntag ermitteln
tB = (tA - 19) % 7;

tC = (40 - firstDigits) % 4;
if (tC == 3)
    tC = tC + 1;
if (tC > 1)
    tC = tC + 1;

temp = year % 100;
tD = (temp + temp / 4) % 7;
```

Listing 83: Methode zur Berechnung des Datums des Ostersonntags (Forts.)

```

tE = ((20 - tB - tC - tD) % 7) + 1;

// Das Datum ermitteln und zurückgeben
int day = tA + tE;
int month = 0;

if (day > 31)
{
    day -= 31;
    month = 4;
}
else
    month = 3;

return new DateTime(year, month, day);
}

```

Listing 83: Methode zur Berechnung des Datums des Ostersonntags (Forts.)

Hinweis

Zur Absicherung meiner Übersetzung habe ich die Ostersonntage von 1700 bis 2299 berechnet und mit den auf der Seite www.assa.org.au/edm.html angegebenen Datumswerten verglichen. Dazu habe ich diese Datumswerte in eine Textdatei geschrieben und so umformatiert, dass die einzelnen Werte in der Form *dd.mm.yyyy* mit einem Semikolon getrennt gespeichert werden. Das Vergleichsprogramm liest diese Textdatei in ein String-Array ein, das die einzelnen Datumswerte enthält, und vergleicht dann die berechneten Oster-Datumswerte mit denen der Textdatei. Das Programm, das keinen Fehler im Algorithmus ermittelte, finden Sie in den Beispielen zu diesem Kapitel auf der Buch-CD.

Die deutschen speziellen Tage lassen sich nun auf der Basis des Osterdatums recht einfach berechnen. Die Methode `GetGermanSpecialDays` in Listing 87 macht genau das. Ich habe diese Methode so entwickelt, dass sie möglichst flexibel einsetzbar ist. `GetGermanSpecialDays` gibt deswegen eine spezielle Hashtable (Listing 86) zurück, die Instanzen der Klasse `GermanSpecialDay` (Listing 84) verwaltet. Über diese Hashtable ist ein Durchgehen der besonderen Tage, aber auch ein gezieltes Zugreifen möglich. Zum Zugriff über den Indizierer habe ich eine Aufzählung erzeugt, die die deutschen Spezialtage als Konstante verwaltet (Listing 85). Das Ganze ist leider (bzw. zwangsläufig) ein wenig komplex ☺.

Zunächst folgt also die Klasse `GermanSpecialDay`, deren Instanzen einen besonderen Tag verwalten. Die Eigenschaften machen eine Aussage über den Namen des Tags, das Datum, darüber, ob der Tag deutschlandweit gilt und ob es sich um einen Feiertag handelt. Im Konstruktor werden zur späteren Vereinfachung alle Eigenschaftswerte übergeben. `GermanSpecialDay` implementiert zusätzlich die `CompareTo`-Methode der `IComparable`-Schnittstelle. Diese Methode vergleicht die Datumsanteile der Objekte und ermöglicht ein Sortieren einer Liste dieser Objekte, z.B. über die `Sort`-Methode der `Array`-Klasse, wenn die Instanzen über ein Array verwaltet werden (was ich im Beispielprogramm in Listing 88 nutze):

```
public class GermanSpecialDay: IComparable
{
    public string Name;
    public DateTime Date;
    public bool Nationwide;
    public bool Holiday;

    /* Konstruktor */
    public GermanSpecialDay(string name, DateTime date,
        bool nationwide, bool holiday)
    {
        this.Name = name;
        this.Date = date;
        this.Nationwide = nationwide;
        this.Holiday = holiday;
    }

    /* Compare-Methode für das Sortieren */
    public int CompareTo(object o)
    {
        GermanSpecialDay gsd = (GermanSpecialDay)o;
        return this.Date.CompareTo(gsd.Date);
    }
}
```

Listing 84: Klasse für einen speziellen Tag

Die Aufzählung `GermanSpecialDaysEnum` beinhaltet Konstanten für die deutschen Spezialtage. Die Werte dieser Aufzählung werden später als Schlüssel der Hashtable verwendet:

```
public enum GermanSpecialDaysEnum
{
    Neujahr,
    HeiligeDreiKönige,
    Valentinstag,
    Maifeiertag,
    MariaHimmelfahrt,
    TagDerDeutschenEinheit,
    Reformationstag,
    Allerheiligen,
    HeiligerAbend,
    ErsterWeihnachtstag,
    ZweiterWeihnachtstag,
    Rosenmontag,
    Aschermittwoch,
    Gründonnerstag,
    Karfreitag,
    Ostersonntag,
    Ostermontag,
}
```

Listing 85: Aufzählung für die besonderen Tage

```

    ChristiHimmelfahrt,
    Pfingstsonntag,
    Pfingstmontag,
    Fronleichnam,
    ErsterAdvent,
    ZweiterAdvent,
    DritterAdvent,
    VierterAdvent,
    Totensonntag,
    BußUndBetttag
}

```

Listing 85: Aufzählung für die besonderen Tage (Forts.)

Die Klasse `GermanSpecialDays` implementiert eine auf `GermanSpecialDay`-Objekte spezialisierte `Hashtable` und wird deswegen von `DictionaryBase` abgeleitet. `GermanSpecialDays` implementiert eine spezialisierte `Add`-Methode, der alle Daten eines besonderen Tages übergeben werden. `Add` hängt ein mit diesen Daten neu erzeugtes `GermanSpecialDay`-Objekt an die von `DictionaryBase` geerbte `Hashtable`-Auflistung an. Außerdem werden die `Values`- und die `Keys`-Eigenschaft implementiert, sodass ein Durchgehen der Werte und Schlüssel möglich ist.

```

public class GermanSpecialDays: DictionaryBase
{
    /* Implementierung der Add-Methode */
    internal void Add(GermanSpecialDaysEnum key, string name,
        DateTime date, bool nationWide, bool holiday)
    {
        base.Dictionary.Add(key, new GermanSpecialDay(name, date,
            nationWide, holiday));
    }

    /* Implementierung der Values- und der Keys-Eigenschaft */
    public ICollection Values
    {
        get {return base.Dictionary.Values;}
    }

    /* Implementierung der Values- und der Keys-Eigenschaft */
    public ICollection Keys
    {
        get {return base.Dictionary.Values;}
    }

    /* Implementierung des Indizierers */
    public GermanSpecialDay this[GermanSpecialDaysEnum key]
    {
        get {return (GermanSpecialDay)base.Dictionary[key];}
        set {base.Dictionary[key] = value;}
    }
}

```

Listing 86: Hashtable für die besonderen Tage

Die Methode `GetGermanSpecialDays` ermittelt schließlich die deutschen Spezialtage in einer `GermanSpecialDays`-Instanz. Die Berechnung erfolgt nach dem anfangs dieses Rezepts beschriebenen Schema, weswegen ich auf eine weitere Beschreibung verzichte. Die Kommentare im Quellcode sagen alles Wichtige aus:

```
public static GermanSpecialDays GetGermanSpecialDays(int year)
{
    // GermanSpecialDays-Instanz erzeugen
    GermanSpecialDays gsd = new GermanSpecialDays();

    // Die festen besonderen Tage eintragen
    gsd.Add(GermanSpecialDaysEnum.Neujahr, "Neujahr",
        new DateTime(year, 1, 1), true, true);
    gsd.Add(GermanSpecialDaysEnum.HeiligeDreiKönige, "Heilige Drei Könige",
        new DateTime(year, 1, 6), false, true);
    gsd.Add(GermanSpecialDaysEnum.Valentinstag, "Valentinstag",
        new DateTime(year, 2, 14), true, false);
    gsd.Add(GermanSpecialDaysEnum.Maifeiertag, "Maifeiertag",
        new DateTime(year, 5, 1), true, true);
    gsd.Add(GermanSpecialDaysEnum.MariaHimmelfahrt, "Maria Himmelfahrt",
        new DateTime(year, 8, 15), false, true);
    gsd.Add(GermanSpecialDaysEnum.TagDerDeutschenEinheit,
        "Tag der Deutschen Einheit", new DateTime(year, 10, 3), true, true);
    gsd.Add(GermanSpecialDaysEnum.Reformationstag, "Reformationstag",
        new DateTime(year, 10, 31), false, true);
    gsd.Add(GermanSpecialDaysEnum.Allerheiligen, "Allerheiligen",
        new DateTime(year, 11, 1), false, true);
    gsd.Add(GermanSpecialDaysEnum.HeiligerAbend, "Heiliger Abend",
        new DateTime(year, 12, 24), true, true);
    gsd.Add(GermanSpecialDaysEnum.ErsterWeihnachtstag, "Erster Weihnachtstag",
        new DateTime(year, 12, 25), true, true);
    gsd.Add(GermanSpecialDaysEnum.ZweiterWeihnachtstag,
        "Zweiter Weihnachtstag", new DateTime(year, 12, 26), true, true);

    // Datum des Ostersonntag berechnen
    DateTime easterSunday = GetEasterSundayDate(year);

    // Die beweglichen besonderen Tage ermitteln, die sich auf Ostern beziehen
    gsd.Add(GermanSpecialDaysEnum.Rosenmontag, "Rosenmontag",
        easterSunday.AddDays(-48), true, false);
    gsd.Add(GermanSpecialDaysEnum.Aschermittwoch, "Aschermittwoch",
        easterSunday.AddDays(-46), true, false);
    gsd.Add(GermanSpecialDaysEnum.Gründonnerstag, "Gründonnerstag",
        easterSunday.AddDays(-3), true, false);
    gsd.Add(GermanSpecialDaysEnum.Karfreitag, "Karfreitag",
        easterSunday.AddDays(-2), true, true);
    gsd.Add(GermanSpecialDaysEnum.Ostermontag, "Ostermontag",
        easterSunday.AddDays(1), true, true);
    gsd.Add(GermanSpecialDaysEnum.Ostersonntag, "Ostersonntag",
        easterSunday, true, true);
}
```

Listing 87: Methode zur Berechnung der besonderen Tage in Deutschland


```

gsd.Add(GermanSpecialDaysEnum.ChristiHimmelfahrt, "Christi Himmelfahrt",
    easterSunday.AddDays(39), true, true);
gsd.Add(GermanSpecialDaysEnum.Pfingstsonntag, "Pfingstsonntag",
    easterSunday.AddDays(49), true, true);
gsd.Add(GermanSpecialDaysEnum.Pfingstmontag, "Pfingstmontag",
    easterSunday.AddDays(50), true, true);
gsd.Add(GermanSpecialDaysEnum.Fronleichnam, "Fronleichnam",
    easterSunday.AddDays(60), false, true);

// Die beweglichen besonderen Tage ermitteln, die sich auf Weihnachten
// beziehen
// Sonntag vor dem 25. Dezember (4. Advent) ermitteln
DateTime firstXMasDay = new DateTime(year, 12, 25);
DateTime fourthAdvent;
int weekday = (int)firstXMasDay.DayOfWeek;
if (weekday == 0)
    // Sonntag
    fourthAdvent = firstXMasDay.AddDays(-7);
else
    fourthAdvent = firstXMasDay.AddDays(-weekday);
gsd.Add(GermanSpecialDaysEnum.VierterAdvent, "Vierter Advent",
    fourthAdvent, true, false);
gsd.Add(GermanSpecialDaysEnum.DritterAdvent, "Dritter Advent",
    fourthAdvent.AddDays(-7), true, false);
gsd.Add(GermanSpecialDaysEnum.ZweiterAdvent, "Zweiter Advent",
    fourthAdvent.AddDays(-14), true, false);
gsd.Add(GermanSpecialDaysEnum.ErsterAdvent, "Erster Advent",
    fourthAdvent.AddDays(-21), true, false);

// Totensonntag ermitteln
DateTime deadSunday = fourthAdvent.AddDays(-28);
gsd.Add(GermanSpecialDaysEnum.Totensonntag, "Totensonntag",
    deadSunday, true, false);

// Den Mittwoch vor dem Totensonntag ermitteln
weekday = (int)deadSunday.DayOfWeek;
if (weekday < 4)
    // Sonntag bis Mittwoch
    gsd.Add(GermanSpecialDaysEnum.BuBUndBetttag, "Buß- und Betttag",
        deadSunday.AddDays(-(weekday + 4)), false, true);
else
    // Donnerstag bis Samstag
    gsd.Add(GermanSpecialDaysEnum.BuBUndBetttag, "Buß- und Betttag",
        deadSunday.AddDays(-(weekday - 3)), false, true);

// Das Ergebnis zurückgeben
return gsd;
}

```

Listing 87: Methode zur Berechnung der besonderen Tage in Deutschland (Forts.)

Listing 88 zeigt eine beispielhafte Anwendung der `GetGermanSpecialDays`-Methode. Berechnet werden die besonderen Tage des aktuellen Jahres. Das Programm liest zuerst gezielt die

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Daten des Ostersonntags aus. Danach gibt das Programm die ermittelten Daten der Tage für die vier Bereiche aus (bundesweite Feiertage, lokale Feiertage etc.). Da die Ausgabe sortiert erfolgen soll (bei `Hashtable`-ähnlichen Auflistungen sind die Daten immer unsortiert und entsprechen auch nicht der Reihenfolge des Hinzufügens), erzeugt das Programm ein einfaches Array mit der Anzahl der ermittelten Feiertage, kopiert über die `CopyTo`-Methode der `Values`-Auflistung die Werte der ermittelten `GermanSpecialDays`-Instanz in dieses Array und sortiert das Array über `Array.Sort`, bevor dieses in einer Schleife durchlaufen wird. Das Sortieren erfolgt nach dem Datum, da die im Array verwalteten `GermanSpecialDay`-Objekte die `CompareTo`-Methode der `IComparable`-Schnittstelle implementieren und diese die Datumswerte vergleicht.

Das Programm erfordert den Import des Namensraums `System.Collections`.

```
// Feiertage des aktuellen Jahrs berechnen
GermanSpecialDays gsd = GetGermanSpecialDays(DateTime.Now.Year);

// Ostersonntag ermitteln
GermanSpecialDay easterSunday = gsd[GermanSpecialDaysEnum.Ostersonntag];
Console.WriteLine(easterSunday.Name);
Console.WriteLine(easterSunday.Date.ToShortDateString());
Console.WriteLine();

// Die speziellen Tage zur Sortierung in ein Array schreiben und sortieren
GermanSpecialDay[] outputArray = new GermanSpecialDay[gsd.Count];
gsd.Values.CopyTo(outputArray, 0);
Array.Sort(outputArray, 0, outputArray.Length);

// Die bundesweiten Feiertage ausgeben
Console.WriteLine("Bundesweite Feiertage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
    if (outputArray[i].Holiday && outputArray[i].Nationwide)
    {
        Console.WriteLine(outputArray[i].Name);
        Console.WriteLine(outputArray[i].Date.ToShortDateString());
        Console.WriteLine();
    }
}

// Die lokalen Feiertage ausgeben
Console.WriteLine("Lokale Feiertage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
    if (outputArray[i].Holiday && outputArray[i].Nationwide == false)
    {
        Console.WriteLine(outputArray[i].Name);
        Console.WriteLine(outputArray[i].Date.ToShortDateString());
        Console.WriteLine();
    }
}
```

Listing 88: Beispielanwendung der `GetGermanSpecialDays`-Methode

```

}

// Die bundesweiten Spezialtage ausgeben
Console.WriteLine("Bundesweite Spezialtage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
    if (outputArray[i].Holiday == false && outputArray[i].Nationwide)
    {
        Console.WriteLine(outputArray[i].Name);
        Console.WriteLine(outputArray[i].Date.ToShortDateString());
        Console.WriteLine();
    }
}

// Die lokalen Spezialtage ausgeben
Console.WriteLine("Lokale Spezialtage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
    if (outputArray[i].Holiday == false &&
        outputArray[i].Nationwide == false)
    {
        Console.WriteLine(outputArray[i].Name);
        Console.WriteLine(outputArray[i].Date.ToShortDateString());
        Console.WriteLine();
    }
}
}

```

Listing 88: Beispielanwendung der GetGermanSpecialDays-Methode (Forts.)

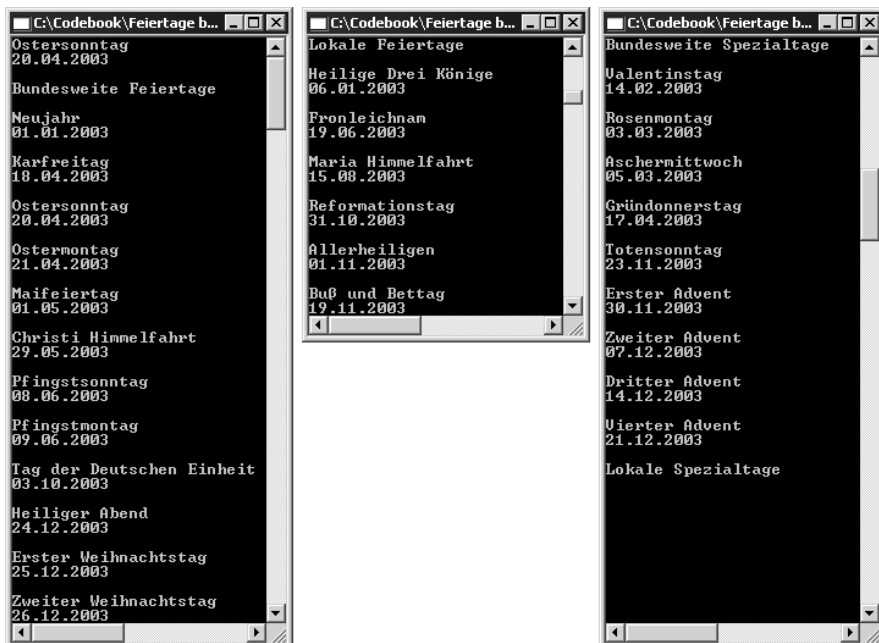


Abbildung 36: Berechnung der Feiertage und der besonderen Tage für das Jahr 2003

Zum Testen der `GetGermanSpecialDays`-Methode habe ich die besonderen Tage der Jahre 2002 bis 2010 berechnet und mit den offiziellen Datumswerten verglichen. Was mich erstaunte, war die Tatsache, dass bereits die erste Version fehlerfrei war 😊.

Abbildung 36 zeigt die Ausgabe dieses Programms für das Jahr 2003.

57 Ermitteln, ob ein bestimmter Tag ein Feiertag ist

Basierend auf der Methode `GetGermanSpecialDays` aus dem Rezept 56 können Sie recht einfach ermitteln, ob ein bestimmter Tag ein Feiertag ist. Die Methode `IsGermanHoliday` in Listing 89 erzeugt dazu eine Auflistung der besonderen Tage des Jahres des abgefragten Datums und geht diese durch. Handelt es sich um einen Feiertag, vergleicht die Methode den Tag und den Monat der `DateTime`-Instanzen (das Jahr ist ja auf jeden Fall identisch). Wird ein Feiertag ermittelt, gibt die Methode `true` zurück und schreibt den Namen des Feiertags und eine Information darüber, ob es ein bundesweiter Feiertag ist, in die am zweiten und dritten Argument By Reference übergebenen Variablen.

`IsGermanHoliday` erfordert den Import der Namensräume `System` und `System.Collections`.

```

/* Die Implementierung der Klassen, Aufzählung und der Methoden für die
 * Oster- und Feiertagsberechnung wird aus Übersichtsgründen hier nicht
 * dargestellt */

/* Methode zur Ermittlung, ob ein bestimmter Tag ein Feiertag ist */
public static bool IsGermanHoliday(DateTime date, out string name,
    out bool nationwide)
{
    // out-Argumente initialisieren
    name = null;
    nationwide = false;

    // Auflistung der besonderen Tage des angegebenen Jahres erzeugen,
    // durchgehen und das Datum der Feiertage mit dem angegebenen Datum
    // vergleichen
    foreach (GermanSpecialDay gsd in
        GetGermanSpecialDays(date.Year).Values)
    {
        if (date.Day == gsd.Date.Day &&
            date.Month == gsd.Date.Month)
        {
            // Datum gefunden
            if (gsd.Holiday)
            {
                // Es ist ein Feiertag: Infos definieren und true zurückgeben
                name = gsd.Name;
                nationwide = gsd.Nationwide;
                return true;
            }
        }
        else

```

Listing 89: Methode zur Ermittlung, ob ein bestimmter Tag ein Feiertag ist

```
        // Kein Feiertag
        return false;
    }

    // Tag wurde nicht gefunden
    return false;
}
```

Listing 89: Methode zur Ermittlung, ob ein bestimmter Tag ein Feiertag ist (Forts.)

Listing 90 zeigt eine beispielhafte Anwendung der `IsGermanHoliday`-Methode, die überprüft, ob der 1.11. des aktuellen Jahres ein Feiertag ist.

```
DateTime date = new DateTime(DateTime.Now.Year, 11, 1);

string name;
bool nationwide;
if (IsGermanHoliday(date, out name, out nationwide))
{
    if (nationwide)
        Console.WriteLine("Der {0} ist ein bundesweiter Feiertag: {1}",
            date.ToShortDateString(), name);
    else
        Console.WriteLine("Der {0} ist ein lokaler Feiertag: {1}",
            date.ToShortDateString(), name);
}
else
    Console.WriteLine("Der {0} ist kein Feiertag",
        date.ToShortDateString());
```

Listing 90: Ermitteln, ob der 1.11. des aktuellen Jahres ein Feiertag ist



Abbildung 37: Das Ergebnis der Ermittlung, ob der 1.11. ein Feiertag ist

58 Zeit genau messen

Das genaue Messen eines Zeitraums ist besonders dann interessant, wenn Sie Ihre Methoden oder Algorithmen daraufhin überprüfen wollen, wie schnell diese ausgeführt werden, oder wenn Sie die Performance verschiedener Algorithmen miteinander vergleichen wollen. Dabei kommt es häufig darauf an, dass die Zeit mit einer möglichst hohen Genauigkeit und Auflösung gemessen wird.

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Zur groben Zeitmessung können Sie die Ticks der aktuellen Zeit verwenden: Das Systemdatum wird in Einheiten von 100ns-Intervallen gespeichert, die seit dem 1.1.0001 00:00:00 vergangen sind. Nun könnten Sie denken, dass ein 100ns-Intervall, das ja immerhin 0,1 Mikrosekunden bzw. 0,0000001 Sekunden entspricht, für eine (recht) genaue Zeitmessung ausreicht. Und das würde es auch, wäre da nicht die Tatsache, dass der System-Zeitgeber je nach System lediglich alle 15 bis 55 Millisekunden aktualisiert wird. Auf einem High-Performance-System mit Windows 2000 oder XP können Sie die Zeit also mit einer Genauigkeit von 15 Millisekunden messen, auf einem älteren System leider nur mit einer höheren Ungenauigkeit. Als Beweis können Sie den folgenden Code verwenden:

```
// Die aktuellen Ticks zwischenspeichern
long startTicks = DateTime.Now.Ticks;

// Schleife, die nur wenig Zeit benötigt
int i = 0;
while (i < 10000000)
    i++;

// Die benötigten Ticks und Sekunden berechnen und ausgeben
long ticks = (DateTime.Now.Ticks - startTicks);
double seconds = ticks / 10000000F;

Console.WriteLine("Benötigte Zeit: {0} Ticks, {1} Sekunden", ticks, seconds);
```

Listing 91: Ungenaue Zeitmessung über die Ticks der Systemzeit

Wenn Sie die Schleifenanzahl auf Ihrem System etwas variieren, erhalten Sie bei einer niedrigeren Anzahl an Durchläufen als Ergebnis 0 und bei einer etwas höheren als Ergebnis einen Wert, der der Aktualisierung des Zeitgebers Ihres Systems entspricht (auf meinem System war das der Wert 0,015625 Sekunden). Bei einem Rechner mit weniger als 2 GHz Prozessorfrequenz müssen Sie die Anzahl der Durchläufe u.U. verkleinern um an dieses Ergebnis zu kommen. Da die gemessenen Werte immer entweder 0 oder ein Vielfaches des Aktualisierungsintervalls sind, ist bewiesen, dass der Systemtimer tatsächlich ein so hohes Intervall verwendet.

Die Lösung des Zeitmess-Problems liegt in der Verwendung der API-Funktionen `QueryPerformanceCounter` und `QueryPerformanceFrequency`. Jedes moderne Motherboard sollte heutzutage einen speziellen so genannten Performance-Counter³ besitzen, dessen Wert in einem sehr kleinen Intervall hochgezählt wird. Das Intervall basiert auf der Motherboard-Grundfrequenz. Den aktuellen Wert dieses Counters können Sie über `QueryPerformanceFrequency` auslesen. Dieser Wert allein sagt noch nicht viel aus. Über `QueryPerformanceFrequency` erhalten Sie aber die Anzahl der Zählungen, die pro Sekunde ausgeführt werden. So können Sie recht einfach errechnen, wie viele Sekunden zwischen zwei Aufrufen von `QueryPerformanceCounter` liegen.

3. Ein Counter ist für Programmierer ein Wert, der in regelmäßigen oder unregelmäßigen Abständen hochgezählt wird

Die Klasse `HighResStopClock` in Listing 92 setzt den Aufruf dieser Funktionen um. Im Konstruktor wird dazu zunächst die aktuelle Performance-Counter-Frequenz ermittelt. `QueryPerformanceFrequency` gibt 0 zurück, wenn beim Aufruf ein Fehler auftritt. Der einzig denkbare Fehler ist, dass das Motherboard keinen Performance-Counter besitzt. Der Konstruktor fragt deswegen den Fehler ab und erzeugt damit eine Ausnahme. So ist eine Zeitmessung nur bei funktionierendem Performance-Counter möglich. Ich habe bei der Auswertung des Fehlers bewusst darauf verzichtet, die dazu passende Systemmeldung auszulesen, da eigentlich nur die eine Fehlermöglichkeit besteht. Die API-Fehlernummer habe ich aber dennoch mit in die Ausnahmebeschreibung aufgenommen (für den Fall, dass `QueryPerformanceFrequency` aus irgendeinem anderen Grund fehlschlägt). Tritt kein Fehler auf, hat `QueryPerformanceFrequency` die aktuelle Frequenz in die By Reference übergebene private Eigenschaft `pcFrequency` geschrieben.

Um bei der Anwendung der Klasse die aktuelle Frequenz und die Auflösung der Zeitmessung auslesen zu können, habe ich die Eigenschaften `PCFrequency` und `PCResolution` implementiert. `PCFrequency` gibt lediglich den Wert der Frequenz zurück, `PCResolution` teilt 1 durch diesen Wert und ergibt somit die Differenz zwischen zwei Zählungen in Sekunden.

Die Methode `Start` startet die Zeitmessung über einen Aufruf der Funktion `QueryPerformanceFrequency` unter der (ref-)Übergabe der privaten Eigenschaft `startCount`. `QueryPerformanceFrequency` schreibt den aktuellen Wert des Performance Counters in die übergebene Eigenschaft, wenn der Aufruf erfolgreich war. Für den unwahrscheinlichen Fall eines Fehlers wertet `Start` die Rückgabe der Funktion aus und generiert im Fehlerfall eine Ausnahme. Da ein Fehler eigentlich nur dann auftreten kann, wenn das System keinen Performance-Counter unterstützt, generiert `Start` nur eine einfache Fehlermeldung.

Über die Methode `Stop` kann eine (Zwischen-)Zeitmessung ausgeführt werden. `Stop` ruft ähnlich `Start` wieder `QueryPerformanceFrequency` auf, übergibt aber eine lokale `long`-Variable. Im Erfolgsfall rechnet `Stop` dann die Differenz zwischen dem Start- und dem Stoppwert aus und gibt diese geteilt durch die Performance-Counter-Frequenz zurück.

Zum Kompilieren der Klasse `HighResStopClock` müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
public class HighResStopClock
{
    /* Deklaration der benötigten API-Funktionen */
    [DllImport("kernel32.dll", SetLastError=true)]
    private extern static int QueryPerformanceCounter(
        ref long lpPerformanceCount);

    [DllImport("kernel32.dll", SetLastError=true)]
    private extern static int QueryPerformanceFrequency(
        ref long lpFrequency );

    /* Variablen für die Frequenz des Performance-Counters und für
     * den Start einer Zeitmessung */
    private long pcFrequency = 0;
```

Listing 92: Klasse zur hochgenauen Zeitmessung

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

private long startCount;

/* Konstruktor */
public HighResStopClock()
{
    // Ermitteln der Performance-Counter-Frequenz
    if (QueryPerformanceFrequency(ref this.pcFrequency) == 0)
        throw new Exception("Die Frequenz des Performance-Counters kann " +
            "nicht ermittelt werden. Windows-Fehler " +
            Marshal.GetLastWin32Error() + ". Wahrscheinlich besitzt das " +
            "Motherboard keinen Performance-Counter");
}

/* Eigenschaft zum Lesen der Frequenz */
public long PCFrequency
{
    get {return this.pcFrequency;}
}

/* Eigenschaft zum Lesen der Auflösung */
public double PCResolution
{
    get {return 1F / this.pcFrequency;}
}

/* Methode zum Starten der Zeitmessung */
public void Start()
{
    if (QueryPerformanceCounter(ref this.startCount) == 0)
        throw new Exception("Der Wert des Performance-Counters kann nicht " +
            "ermittelt werden. Windows-Fehler " +
            Marshal.GetLastWin32Error());
}

/* Methode zum Stoppen der Zeitmessung */
public double Stop()
{
    long count = 0;
    if (QueryPerformanceCounter(ref count) == 0)
        throw new Exception("Der Wert des Performance-Counters kann nicht" +
            "ermittelt werden. Windows-Fehler " +
            Marshal.GetLastWin32Error());

    return (count - this.startCount) / (double)this.pcFrequency;
}
}

```

Listing 92: Klasse zur hochgenauen Zeitmessung (Forts.)

Listing 93 implementiert ein kleines Programm zum Testen beider Varianten der Zeitmessung. Gemessen wird die Zeit, die eine `while`-Schleife für 1.000.000 Durchläufe benötigt, wenn in dieser eine `int`-Variable hochgezählt wird.


```
// Zeit ungenau (mit einem Intervall von 15 bis 55ms) über die Ticks
// der DateTime-Struktur messen
Console.WriteLine("Zeitmessung über DateTime.Now.Ticks:");
long startTicks = DateTime.Now.Ticks;

int i = 0;
while (i < 1000000)
    i++;

long ticks = (DateTime.Now.Ticks - startTicks);
double seconds = ticks / 10000000F;
Console.WriteLine("Benötigte Zeit: {0} Ticks, {1} Sekunden", ticks, seconds);

// Zeit genau über QueryPerformanceCounter messen
Console.WriteLine();
Console.WriteLine("Zeitmessung über QueryPerformanceCounter:");
HighResStopClock stopClock;
try
{
    stopClock = new HighResStopClock();
    Console.WriteLine("Aktuelle Frequenz: {0}", stopClock.PCFrequency);
    Console.WriteLine("Aktuelle Auflösung: {0:G} Sekunden",
        stopClock.PCResolution);

    stopClock.Start();
    i = 0;
    while (i < 1000000)
        i++;
    seconds = stopClock.Stop();
    Console.WriteLine("Benötigte Zeit: {0} Sekunden", seconds);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

Listing 93: Ungenaue und genaue Zeitmessung einer Schleife mit 1.000.000 Durchläufen

Abbildung 38 zeigt, dass die Zeitmessung über QueryPerformanceCounter wesentlich genauer ist als die über die Systemticks.

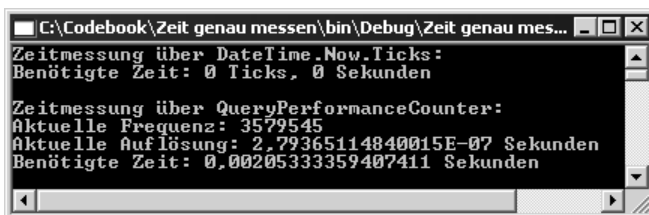


Abbildung 38: Zeitmessung einer Schleife mit 1.000.000 Durchläufen

Anwendungs-Konfiguration, Anwendungen, Prozesse und Dienste

59 Den Dateinamen der Anwendung ermitteln

Den vollen Dateinamen der ausführenden Anwendung (inklusive Pfad) erhalten Sie in einer Windows-Anwendung über die Eigenschaft `ExecutablePath` des `Application`-Objekts:

```
string filename = System.Windows.Forms.Application.ExecutablePath;
```

Um die `Application`-Klasse verfügbar zu haben, benötigt die Anwendung einen Verweis auf die Assembly `System.Windows.Forms.dll`, was ohne Probleme auch in einer Konsolenanwendung oder Klassenbibliothek möglich ist.

Ohne diesen Verweis können Sie aber auch stattdessen die Eigenschaft `Location` eines `Assembly`-Objekts abfragen, das die »Eintritts-Assembly« (das ist die Assembly, über die das Programm gestartet wurde) repräsentiert. Dieses Objekt erhalten Sie über die Methode `GetEntryAssembly` der `Assembly`-Klasse, die Bestandteil des `System.Reflection`-Namensraums ist:

```
string filename = System.Reflection.Assembly.GetEntryAssembly().Location;
```

Hinweis

Als *Reflektion* (Reflection) werden Features bezeichnet, über die Sie in der Laufzeit eines Programms Informationen über dessen Assemblies, die darin enthaltenen Klassen und deren Schnittstellen, Methoden, Eigenschaften und Ereignisse etc. auslesen können. Ein weiterer Teil von Reflektion ist das dynamische Erzeugen von Klassen und Assemblies, also das Erzeugen von Programmcode in der Laufzeit eines Programms.

Häufig wird auch vorgeschlagen, die »ausführende Assembly« zu verwenden, die Sie über die Methode `GetExecutingAssembly` erreichen:

```
string filename = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

Dabei müssen Sie allerdings aufpassen: Wird diese Methode in einer Klassenbibliothek aufgerufen, liefert sie ein `Assembly`-Objekt zurück, das die Assembly der *Klassenbibliothek* repräsentiert. Der ausgelesene Dateiname ist also in diesem Fall nicht der der Anwendung, sondern der der Klassenbibliothek.

Basics

Datum
Zeit

Anwen-
dunge

Datei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

In allen Fällen erhalten Sie den Dateinamen inklusive Pfad. Wenn Sie nur den Namen der Datei auslesen wollen, können Sie diesen über ein `FileInfo`-Objekt ermitteln. Ein solches Objekt dient dem Auslesen von Informationen zu einer Datei und liefert u.a. über die Eigenschaft `Name` den Dateinamen. Die folgende Funktion liest den Dateinamen der Anwendung aus und liefert diesen zurück:

```
public string GetApplicationFilename()
{
    /* FileInfo-Objekt für die Datei erzeugen, die die Eintritts-
     * Assembly speichert */
    System.IO.FileInfo fi = new System.IO.FileInfo(
        System.Reflection.Assembly.GetEntryAssembly().Location);

    /* Dateiname auslesen und zurückgeben */
    return fi.Name;
}
```

60 Das Verzeichnis der Anwendung ermitteln

In einer Anwendung mit Referenz auf die Assembly `System.Windows.Forms.dll` können Sie den Pfad der Anwendung über die Eigenschaft `StartupPath` des `Application`-Objekts ermitteln:

```
string applicationPath = System.Windows.Forms.Application.StartupPath;
```

Ohne Referenz auf die `System.Windows.Forms.dll` können Sie stattdessen auch den vollen Dateinamen der »Eintritts-Assembly« ermitteln und über ein `FileInfo`-Objekt den Pfad auslesen:

```
public string GetApplicationPath()
{
    // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-
    // Assembly speichert
    System.IO.FileInfo fi = new System.IO.FileInfo(
        System.Reflection.Assembly.GetEntryAssembly().Location);

    // Den Pfad des Verzeichnisses der Datei zurückgeben
    return fi.DirectoryName;
}
```

Beide Varianten funktionieren auch ohne Probleme in einer Klassenbibliothek. Sie erhalten dann allerdings das Verzeichnis der Anwendung zurück, nicht das der Klassenbibliothek.

61 Konfigurationsdaten aus der .config-Datei lesen

Eine Anwendung kann aus einer XML-Datei, deren Name dem Dateinamen der Anwendung mit angehängter Endung *.config* entspricht und die im Ordner der Anwendung gespeichert ist, Konfigurationsdaten auslesen. Diese Datei ist beispielsweise folgendermaßen aufgebaut:

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

  <system.windows.forms jitDebugging="false" />

  <appSettings>
    <add key="server" value="Zaphod" />
    <add key="trustedConnection" value="false" />
    <add key="userId" value="Trillian" />
    <add key="password" value="42" />
  </appSettings>

</configuration>
```

Listing 94: Eine Beispiel-Konfigurationsdatei

Die `add`-Elemente im `appSettings`-Element speichern anwendungsspezifische Daten. Diese Daten können Sie sehr einfach über die `AppSettings`-Eigenschaft der Klasse `ConfigurationSettings` aus dem Namensraum `System.Configuration` auslesen:

```
string server = ConfigurationSettings.AppSettings["Server"];
string userId = ConfigurationSettings.AppSettings["UserId"];
string password = ConfigurationSettings.AppSettings["password"];
```

Listing 95: Auslesen von Konfigurationsdaten aus der .config-Datei

Etwas problematisch an dieser .NET-mäßigen Speicherung von Konfigurationsdaten ist, dass die Daten automatisch und nur beim Start der Anwendung ausgelesen werden. Werden die Daten in der Konfigurationsdatei geändert, während die Anwendung läuft, sind die Änderungen nicht sichtbar, auch wenn die Anwendung die Daten erneut abfragt. Dies kann bei Anwendungen, die permanent (z.B. als Dienst) ausgeführt werden, zum echten Problem werden; leider habe ich noch keine Lösung dafür gefunden. Sie können aber stattdessen einfach eine eigene XML-Datei für die Konfiguration verwenden, wie ich es im Rezept 64 zeige.

Ein anderes Problem ist, dass die `ConfigurationSettings`-Klasse leider keine Methode zum Speichern von neuen oder veränderten Daten in der *.config*-Datei besitzt. Dieses Problem löse ich im Rezept 63.

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

62 Konfigurationsdaten in eigenen Abschnitten speichern

Wenn Sie die Konfigurationsdaten einer Anwendung zwar in der *.config*-Datei, aber nicht im dafür vorgesehenen Abschnitt *appSettings* speichern wollen (siehe vorhergehendes Rezept), können Sie diese in einer eigenen Sektion speichern und dafür einen Handler entwickeln. Ein solcher implementiert die Schnittstelle *IConfigurationSectionHandler*, deren Methode *Create* für alle registrierten, benutzerdefinierten Konfigurationsabschnitte aufgerufen wird. Über das Argument *section*, das ein *XmlNode*-Objekt referenziert, erhalten Sie Zugriff auf das XML-Element, das die Daten verwaltet.

Zunächst müssen die neuen Abschnitte in der Konfigurationsdatei registriert werden. Das folgende Listing registriert die Abschnitte *system* und *database* im Element *codebook*:

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

  <!-- Bekanntmachung der eigenen Konfigurations-Sektionen -->
  <configSections>
    <sectionGroup name="codebook">
      <section
        name="database"
        type="Addison_Wesley.Codebook.Configuration.
          ConfigSectionHandler, Config-Handler"/>
      <section
        name="system"
        type="Addison_Wesley.Codebook.Configuration.
          ConfigSectionHandler, Config-Handler"/>
    </sectionGroup>
  </configSections>
```

Listing 96: Registration eigener Konfigurations-Abschnitte

Das *sectionGroup*-Element legt über das Attribut *name* den Namen des XML-Elements fest, das die Konfigurationsdaten verwaltet. Im Element *section* werden die einzelnen Abschnitte definiert. Das Attribut *name* bestimmt dabei den Namen des XML-Elements. Das Attribut *type* legt fest, welche Klasse für das Handling dieser Elemente zuständig ist. Dabei geben Sie zuerst den vollen Namen der Klasse (inklusive Namensraum) und durch ein Komma getrennt die Assembly an, in der diese Klasse gespeichert ist (im Beispiel *Config-Handler*). Das *sectionGroup*-Element kann (natürlich) auch komplett in der Haupt-Konfigurationsdatei *machine.config* gespeichert werden, was u.U. Sinn macht, wenn es sich um Konfigurationsdaten handelt, die Sie öfter verwenden. Dann müssen Sie aber auch dafür sorgen, dass diese Eintragung in der Maschinenkonfiguration auf den Computern eingetragen wird, auf dem Ihr Programm ausgeführt werden soll.

In der Konfigurationsdatei folgen dann Ihre eigenen Konfigurations-Elemente:

```

<codebook>
  <database>
    <server>Zaphod</server>
    <userId>Trillian</userId>
    <password>42</password>
  </database>

  <system>
    <lastAccess>01.01.2003 10:00:00</lastAccess>
  </system>
</codebook>

</configuration>

```

Listing 97: Eigene Konfigurations-Abschnitte

Zum Speichern der später eingelesenen Daten benötigen Sie zunächst für jeden benutzerdefinierten Konfigurationsabschnitt eine Klasse.

```

using System;
using System.Xml;
using System.Configuration;

...

/* Klasse zur Speicherung der Datenbank-Konfiguration */
public class DatabaseConfig
{
    public string Server;
    public string UserId;
    public string Password;

    internal DatabaseConfig(string server, string userId, string password)
    {
        this.Server = server;
        this.UserId = userId;
        this.Password = password;
    }
}

/* Klasse zur Speicherung der System-Konfiguration */
public class SystemConfig
{
    public string LastAccess;

    internal SystemConfig(string lastAccess)
    {
        this.LastAccess = lastAccess;
    }
}

```

Listing 98: Klassen zur Speicherung der eingelesenen Daten

 }

Listing 98: Klassen zur Speicherung der eingelesenen Daten (Forts.)

Zum Lesen der eigenen Konfigurationsabschnitte müssen Sie eine Klasse entwickeln, die die Schnittstelle `IConfigurationSectionHandler` implementiert. Der `Create`-Methode dieser Klasse werden später Informationen über das übergeordnete Element, ein Konfigurationskontext (nur für ASP.NET-Anwendungen, wo dieser den virtuellen Pfad der Konfigurationsdatei enthält) und ein `XmlNode`-Objekt übergeben, das das XML-Element mit den Konfigurationsdaten enthält. Über dieses Objekt können Sie die Daten lesen.

Die Methode `Create` in Listing 99 fragt zunächst über die `Name`-Eigenschaft ab, um welchen Abschnitt es sich handelt. Dann werden die entsprechenden Daten eingelesen und damit eine Instanz der Klasse erzeugt, die für diese Daten vorgesehen ist. Diese Instanz gibt `Create` dann zurück.

Etwas problematisch ist, dass Konfigurationsdaten auch fehlen können. `Create` fragt deshalb vor dem Lesen ab, ob die einzelnen Unterelemente vorhanden sind. Für den Fall, dass ein unbekannter Konfigurationsabschnitt registriert wurde, generiert die Methode eine Ausnahme:

```

/* Klasse, die den Konfigurations-Sektions-Handler implementiert */
public class ConfigSectionHandler: IConfigurationSectionHandler
{
    /* Implementierung der Create-Methode */
    public object Create(object parent, object configContext, XmlNode section)
    {
        if (section.Name == "database")
        {
            // Einlesen der Unterelemente der Sektion
            XmlNode subNode;
            string server = null;
            subNode = section.SelectSingleNode("server");
            if (subNode != null)
                server = subNode.InnerText;

            string userId = null;
            subNode = section.SelectSingleNode("userId");
            if (subNode != null)
                userId = subNode.InnerText;

            string password = null;
            subNode = section.SelectSingleNode("password");
            if (subNode != null)
                password = subNode.InnerText;

            // Neue DatabaseConfig-Instanz zurückgeben
            return new DatabaseConfig(server, userId, password);
        }
    }
}

```

Listing 99: Handler für eigene Konfigurationsabschnitte

```

    }
    else if (section.Name == "system")
    {
        // Einlesen der Unterelemente der Sektion
        XmlNode subNode;
        string lastAccess = null;
        subNode = section.SelectSingleNode("lastAccess");
        if (subNode != null)
            lastAccess = subNode.InnerText;

        // Neue SystemConfig-Instanz zurückgeben
        return new SystemConfig(lastAccess);
    }
    else
    {
        // Unbekannte Sektion: Ausnahme werfen
        throw new ConfigurationException("Unbekannte Konfigurations-" +
            "Sektion '" + section.Name + "'");
    }
}
}
}

```

Listing 99: Handler für eigene Konfigurationsabschnitte (Forts.)

Nun müssen Sie die Konfigurationsdaten in der Anwendung lediglich noch über die Methode `ConfigurationSettings.GetConfig` einlesen, indem Sie den relativen Pfad zum Konfigurationsabschnitt angeben. `GetConfig` gibt das in `Create` erzeugte Objekt zurück, wenn der Abschnitt gefunden wird. Im anderen Fall wird `null` zurückgegeben. So können Sie beim Einlesen überprüfen, ob der Abschnitt vorhanden ist. Das Programm in Listing 100 implementiert dies in Form einer einfachen Konsolenanwendung. Sie müssen die Namensräume `System` und `System.Configuration` importieren, um dieses Programm kompilieren zu können.

```

DatabaseConfig databaseConfig = null;
SystemConfig systemConfig = null;

try
{
    // Einlesen der Konfiguration
    systemConfig = (SystemConfig)ConfigurationSettings.GetConfig(
        "codebook/system");
    databaseConfig = (DatabaseConfig)ConfigurationSettings.GetConfig(
        "codebook/database");

    // Überprüfen, ob der System-Konfigurationsabschnitt eingelesen wurde
    if (systemConfig != null)
    {
        // Ausgeben der Konfigurationsdaten
        Console.WriteLine("LastAccess: {0}", systemConfig.LastAccess);
    }
}

```

Listing 100: Einlesen eigener Konfigurationsabschnitte in einer Konsolenanwendung

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    }
    else
        Console.WriteLine("Die Systemkonfiguration konnte nicht " +
            "eingelesen werden");

    // Überprüfen, ob der Datenbank-Konfigurationsabschnitt eingelesen wurde
    if (databaseConfig != null)
    {
        // Ausgeben der Konfigurationsdaten
        Console.WriteLine("Server: {0}", databaseConfig.Server);
        Console.WriteLine("UserId: {0}", databaseConfig.UserId);
        Console.WriteLine("Passwort: {0}", databaseConfig.Password);
    }
    else
        Console.WriteLine("Die Datenbankkonfiguration konnte nicht " +
            "eingelesen werden");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 100: Einlesen eigener Konfigurationsabschnitte in einer Konsolenanwendung (Forts.)



Abbildung 39: Das Beispielprogramm zum Einlesen eigener Konfigurationsabschnitte

63 Konfigurationsdaten in der .config-Datei speichern

Das .NET Framework bietet leider keine direkte Möglichkeit, Konfigurationsdaten programmatisch in der Konfigurationsdatei einer Anwendung zu speichern. Das Lesen dieser Daten ist hingegen möglich, wie ich es in den Rezepten 61 und 62 beschrieben habe.

Das Ändern der Anwendungskonfiguration ist aber (natürlich) dennoch möglich. Dazu können Sie ein `XmlDocument`-Objekt verwenden, über das Sie die XML-Datei direkt bearbeiten. Das Ändern von XML-Dateien über ein solches Objekt beschreibe ich grundlegend im Rezept 140.

Das Prinzip ist einfach (wenn man weiß, wie es geht): Sie lesen die Konfigurationsdatei ein und suchen über einen XPath-Ausdruck (siehe Rezept 137) das `appSettings`-Element. Um eine Einstellung zu schreiben, überprüfen Sie zunächst, ob diese vorhanden ist, und aktualisieren in diesem Fall deren Wert. Ist die Einstellung nicht vorhanden, legen Sie diese einfach neu an.

Die Methode `SaveSettingToDefaultConfigFile` in Listing 101 setzt dies um. Diese Methode setzt neben dem `XmlDocument`-Objekt auch XPath-Ausdrücke zur Suche von Elementen ein. Zum

Kompilieren müssen Sie die Namensräume `System`, `System.Xml`, `System.IO` und `System.Windows.Forms` einbinden.

```
public static void SaveSettingToDefaultConfigFile(string settingName,
    object settingValue)
{
    // Dateiname zusammenstellen
    string configFileName = System.Windows.Forms.Application.ExecutablePath +
        ".config";

    // XmlDocument für diese Datei erzeugen
    XmlDocument xmlDoc = new XmlDocument();
    try
    {
        xmlDoc.Load(configFileName);
    }
    catch (XmlException ex)
    {
        throw new XmlException("Fehler beim Laden der Konfigurationsdatei: " +
            ex.Message, ex);
    }

    // appSettings-Element suchen und falls nicht vorhanden einfach neu
    // anlegen
    XmlNode appSettingsNode = xmlDoc.SelectSingleNode(
        "//configuration/appSettings");
    if (appSettingsNode == null)
    {
        // appSettings-Element erzeugen
        appSettingsNode = xmlDoc.CreateElement("appSettings");

        // configuration-Element suchen
        XmlNode configurationNode = xmlDoc.SelectSingleNode(
            "//configuration");
        if (configurationNode == null)
        {
            // configuration-Element nicht gefunden: Ausnahme werfen
            throw new XmlException("configuration-Element nicht gefunden",
                null);
        }
        // appSettings-Element anhängen
        configurationNode.AppendChild(appSettingsNode);
    }

    // Einstellungs-Element suchen
    XmlNode settingNode = xmlDoc.SelectSingleNode("//configuration/" +
        "appSettings/add[@attribute::key='" + settingName + "']");
    if (settingNode != null)
    {
```

Listing 101: Methode zum Speichern von Konfigurationsdaten in der Konfigurationsdatei einer Anwendung

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// Element gefunden: Wert des Attributs value aktualisieren
settingNode.Attributes["value"].InnerText = settingValue.ToString();
}
else
{
    // Element nicht gefunden, also neu anlegen
    XmlElement settingElement = xmlDoc.CreateElement("add");

    // Attribute definieren
    settingElement.SetAttribute("key", settingName.ToString());
    settingElement.SetAttribute("value", settingValue.ToString());

    // Einstellungs-Element an das appsettings-Element anfügen
    appSettingsNode.AppendChild(settingElement);
}

// Datei speichern
try
{
    xmlDoc.Save(configFileName);
}
catch (XmlException ex)
{
    throw new XmlException("Fehler beim Speichern der " +
        "Konfigurationsdatei: " + ex.Message, ex);
}
}
```

Listing 101: Methode zum Speichern von Konfigurationsdaten in der Konfigurationsdatei einer Anwendung (Forts.)

Die Anwendung dieser Methode zeigt Listing 102:

```
SaveSettingToDefaultConfigFile("Server", "Zaphod");
SaveSettingToDefaultConfigFile("UserId", "Trillian");
SaveSettingToDefaultConfigFile("Password", "42");
```

Listing 102: Anwendung der Methode zum Speichern von Konfigurationsdaten

64 Anwendungskonfiguration über eine eigene XML-Datei

Die Anwendungskonfiguration über die Standard-Konfigurationsdatei einer Anwendung (siehe die vorhergehenden Rezepte) ist zwar recht einfach, aber leider auch etwas unflexibel. So sind Änderungen, die während der Laufzeit eines Programms in der Datei vorgenommen werden, leider nicht im Programm sichtbar. Daneben ist die Verwaltung der Daten in `add`-Elementen des `appSettings`-Elements für einen unbedarften Anwender u. U. etwas undurchsichtig.

Die im Folgenden beschriebene Klasse `Config` ermöglicht hingegen die Verwaltung von Konfigurationsdaten in einer einfachen XML-Datei, die beispielsweise folgendermaßen aussieht:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<config>
  <Database>
    <Server>Zaphod</Server>
    <UserId>Trillian</UserId>
    <Password>42</Password>
  </Database>
  <System>
    <StartTime>10:00</StartTime>
    <LastAccess>12.04.2003 14:03:36</LastAccess>
  </System>
</config>
```

Listing 103: Eine eigene Konfigurationsdatei

Die einzelnen Einstellungen sind in normalen XML-Elementen gespeichert und werden über übergeordnete Elemente in Sektionen eingeteilt. Eine Instanz der Klasse `Config` ermöglicht über eine Auflistung den Zugriff auf die einzelnen Sektionen. Innerhalb jeder Sektion steht eine weitere Auflistung für die Einstellungen zur Verfügung. Für jede zu speichernde Einstellung verwaltet die Klasse den Namen, den Wert, einen Defaultwert und eine Information darüber, ob die Einstellung beim Lesen der Konfigurationsdatei in dieser gefunden wurde.

Zur Realisation dieser Idee ist zunächst eine Klasse `Setting` zur Speicherung der einzelnen Einstellungen notwendig.

```
using System;
using System.IO;
using System.Xml;
using System.Collections;

...

public class Setting
{
    public string Name;           // der Name der Einstellung
    public string Value;          // der Wert der Einstellung
    public string DefaultValue;   // der Defaultwert für das Lesen
    public bool WasInFile;        // Info darüber, ob die Einstellung in der
                                // Config-Datei gefunden wurde

    /* Konstruktor */
    public Setting(string name, string defaultValue)
    {
        this.Name = name;
        this.DefaultValue = defaultValue;
    }
}
```

Listing 104: Klasse zur Speicherung einer Einstellung

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die Klasse `Settings` implementiert eine Auflistung vom Typ `Hashtable` und ist spezialisiert auf Instanzen von `Setting`. `Settings` wird von der Klasse `DictionaryBase` abgeleitet, die als Grundlage für die Erzeugung von typsicheren Auflistungen dient. Um mit der Auflistung arbeiten zu können werden der Indizierer, die `Add`- und die `Remove`-Methode sowie die `Values`- und die `Keys`-Eigenschaft implementiert. Die Eigenschaften `Values` und `Keys` erlauben ein späteres Durchgehen der Werte bzw. der Schlüssel der Auflistung über `foreach`.

```
public class Settings: DictionaryBase
{
    /* Implementierung des Indizierers */
    public Setting this[string name]
    {
        set {this.Dictionary[name] = value;}
        get {return (Setting)this.Dictionary[name];}
    }

    /* Implementierung der Add-Methode */
    public void Add(string settingName, string defaultValue)
    {
        this.Dictionary.Add(settingName,
            new Setting(settingName, defaultValue));
    }

    /* Implementierung der Remove-Methode */
    public void Remove(string name)
    {
        this.Dictionary.Remove(name);
    }

    /* Implementierung der Values-Eigenschaft zum Durchgehen mit foreach */
    public ICollection Values
    {
        get {return Dictionary.Values;}
    }

    /* Implementierung der Keys-Eigenschaft zum Durchgehen mit foreach */
    public ICollection Keys
    {
        get {return Dictionary.Keys;}
    }
}
```

Listing 105: Klasse zur Verwaltung eine Auflistung von Einstellungen

Die Klasse `Section` verwaltet die Daten einer Sektion. Neben dem Namen derselben enthält diese Klasse eine Eigenschaft `Settings` vom Typ der Einstellungs-Auflistung. In dieser Auflistung sollen die Einstellungen verwaltet werden, die der Sektion zugeordnet sind. Damit die Auflistung automatisch zur Verfügung steht, wird die Instanz der Klasse `Settings` im Konstruktor erzeugt:

```

public class Section
{
    public string Name;           // Der Name der Sektion
    public Settings Settings;     // Die Einstellungen der Sektionen
    /* Konstruktor */
    public Section(string sectionName)
    {
        this.Name = sectionName;
        this.Settings = new Settings();
    }
}

```

Listing 106: Klasse zur Speicherung einer Einstellungs-Sektion

Ähnlich der Auflistung zur Verwaltung von Einstellungen implementiert die Klasse `Sections` eine `HashTable`-Auflistung zur Verwaltung mehrerer `Section`-Instanzen:

```

public class Sections: DictionaryBase
{
    /* Implementierung des Indizierers */
    public Section this[string name]
    {
        set {this.Dictionary[name] = value;}
        get {return (Section)this.Dictionary[name];}
    }

    /* Implementierung der Add-Methode */
    public void Add(string name)
    {
        this.Dictionary.Add(name, new Section(name));
    }

    /* Implementierung der Remove-Methode */
    public void Remove(string name)
    {
        this.Dictionary.Remove(name);
    }

    /* Implementierung der Values-Eigenschaft zum Durchgehen mit foreach */
    public ICollection Values
    {
        get {return Dictionary.Values;}
    }

    /* Implementierung der Keys-Eigenschaft zum Durchgehen mit foreach */
    public ICollection Keys
    {
        get {return Dictionary.Keys;}
    }
}

```

Listing 107: Auflistungs-Klasse zur Speicherung von Einstellungs-Sektionen

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die eigentliche Haupt-Klasse `Config` verwaltet in einer privaten Eigenschaft den Dateinamen der Konfigurationsdatei und in einer öffentlichen Eigenschaft eine Instanz der Klasse `Sections`. Im Konstruktor wird der Dateinamen übergeben und die `Sections`-Instanz erzeugt:

```
public class Config
{
    /* Eigenschaften */
    private string fileName;
    public Sections Sections;

    /* Konstruktor */
    public Config(string fileName)
    {
        this.fileName = fileName;
        this.Sections = new Sections();
    }
}
```

Listing 108: Eigenschaften und Konstruktor der Klasse Config

Die Methode `Load` ermöglicht das Lesen der Einstellungen aus der Konfigurationsdatei. Diese Methode erzeugt dazu zunächst eine Instanz der Klasse `XmlDocument` und lädt die über die Eigenschaft `fileName` angegebene XML-Datei. Für den Fall, dass beim Laden ein Fehler auftritt, fängt die Methode die möglichen Ausnahmen ab und erzeugt daraus direkt eine neue Ausnahme, allerdings mit einer genaueren Fehlermeldung. Danach geht `Load` alle Sektionen und in diesen alle Einstellungen durch und ermittelt über einen XPath-Ausdruck das XML-Element, das diese Einstellung speichert. Wird das Element gefunden, liest die Methode dessen Wert aus und speichert ihn im aktuellen `Setting`-Objekt. Um später auswerten zu können, ob Einstellungen erfolgreich gelesen wurden, wird die Eigenschaft `WasInFile` des `Setting`-Objekts entsprechend mit `true` oder `false` beschrieben. `Load` gibt `true` zurück, wenn alle Einstellungen gelesen werden konnten. Konnte auch noch eine Einstellung nicht gelesen werden, gibt diese Methode `false` zurück.

```
public bool Load()
{
    // Variable für den Rückgabewert
    bool returnValue = true;

    // XmlDocument-Objekt für die Einstellungs-Datei erzeugen
    XmlDocument xmlDoc = new XmlDocument();

    // Datei laden
    try
    {
        xmlDoc.Load(this.fileName);
    }
    catch (IOException ex1)
    {
    }
}
```

Listing 109: Methode zum Lesen der Konfigurationsdatei

```

        throw new IOException("Fehler beim Laden der Konfigurationsdatei '" +
            this.fileName + "': " + ex1.Message);
    }
    catch (XmlException ex2)
    {
        throw new XmlException("Fehler beim Laden der Konfigurationsdatei '" +
            this.fileName + "': " + ex2.Message, ex2);
    }

    // Alle Sektionen durchgehen und die Einstellungen einlesen
    foreach (Section section in this.Sections.Values)
    {
        // Alle Einstellungen der Sektion durchlaufen
        foreach (Setting setting in section.Settings.Values)
        {
            // Einstellung im XML-Dokument lokalisieren
            XmlNode settingNode = xmlDoc.SelectSingleNode(
                "/config/" + section.Name + "/" + setting.Name);
            if (settingNode != null)
            {
                // Einstellung gefunden
                setting.Value = settingNode.InnerText;
                setting.WasInFile = true;
            }
            else
            {
                // Einstellung nicht gefunden
                setting.Value = setting.DefaultValue;
                setting.WasInFile = false;
                returnValue = false;
            }
        }
    }

    // Ergebnis zurückmelden
    return returnValue;
}

```

Listing 109: Methode zum Lesen der Konfigurationsdatei (Forts.)

Die Methode `Save` speichert die Konfigurationsdaten in die XML-Datei. Sie erzeugt dazu eine neue Instanz der Klasse `XmlDocument` und initialisiert diese mit dem Skelett des XML-Dokuments (wobei als kleiner Trick die `LoadXml`-Methode mit einem String aufgerufen wird, der genau dieses Skelett beinhaltet).

Dann geht `Save` die einzelnen Sektionen der Einstellungen durch, um innerhalb des XML-Dokuments für jede Sektion und für jede Einstellung ein entsprechendes Element zu erzeugen. Schließlich wird die Datei noch gespeichert, wobei natürlich eventuelle Ausnahmen abgefangen und mit einer genaueren Fehlermeldung weitergegeben werden.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
public void Save()
{
    // XmlDocument-Objekt für die Einstellungs-Datei erzeugen
    XmlDocument xmlDoc = new XmlDocument();

    // Skelett der XML-Datei erzeugen
    xmlDoc.LoadXml("<?xml version=\"1.0\" encoding=\"utf-8\" \" +
        \"standalone=\"yes\"?><config></config>");

    // Alle Sektionen durchgehen und die Einstellungen schreiben
    foreach (Section section in this.Sections.Values)
    {
        // Element für die Sektion erzeugen und anfügen
        XmlElement sectionElement = xmlDoc.CreateElement(section.Name);
        xmlDoc.DocumentElement.AppendChild(sectionElement);

        // Alle Einstellungen der Sektion durchlaufen
        foreach (Setting setting in section.Settings.Values)
        {
            // Einstellungs-Element erzeugen und anfügen
            XmlElement settingElement =
                xmlDoc.CreateElement(setting.Name);
            settingElement.InnerText = setting.Value;
            sectionElement.AppendChild(settingElement);
        }
    }

    // Datei speichern
    try
    {
        xmlDoc.Save(this.fileName);
    }
    catch (IOException ex1)
    {
        throw new IOException("Fehler beim Speichern der " +
            "Konfigurationsdatei '" + this.fileName + "': " + ex1.Message);
    }
    catch (XmlException ex2)
    {
        throw new XmlException("Fehler beim Speichern der " +
            "Konfigurationsdatei '" + this.fileName + "': " + ex2.Message, ex2);
    }
}
```

Listing 110: Methode zum Speichern der Konfigurationsdatei

Die beispielhafte Anwendung dieser Klasse zeigt das folgende Listing:

```
// Config-Instanz erzeugen
Config config = new Config(Path.Combine(Application.StartupPath,
    "Config.xml"));

// Konfigurations-Sektionen und Einstellungen definieren
config.Sections.Add("System");
config.Sections.Add("Database");
config.Sections["System"].Settings.Add("LastAccess", "00:00");
config.Sections["Database"].Settings.Add("Server", "(local)");

// Datei einlesen
config.Load();

// Ein Datum lesen und ausgeben
Setting setting = config.Sections["System"].Settings["LastAccess"];
if (setting.WasInFile == false)
    Console.WriteLine("Einstellung System/LastAccess nicht gefunden");
Console.WriteLine("LastAccess: {0}", setting.Value);

// Den Wert Datum ändern
setting.Value = DateTime.Now.ToString();

// Datei speichern
config.Save();
```

Listing 111: Anwendung der Klasse Config zur Verwaltung von Anwendungs-Konfigurationsdaten

Hinweis

Die in diesem Rezept vorgestellten Klassen könnten noch wesentlich erweitert werden. So könnte die `Setting`-Klasse zusätzlich den Datentyp der Einstellung und bei numerischen Daten einen Minimal- und Maximalwert verwalten. Beim Einlesen bzw. Schreiben der Daten könnten diese dann auf den korrekten Datentyp und die Einhaltung der Wertgrenzen überprüft werden. Eine weitere sinnvolle Idee ist die automatische Erzeugung eines Formulars über eine weitere Methode, in dem der Anwender die Einstellungen bearbeiten kann. Das Formular könnte die einzelnen Sektionen und deren Einstellungen in einem `TreeView`-Steuerelement darstellen. Wählt der Anwender eine Einstellung aus, wird diese in einem separaten Bereich des Formulars zur Bearbeitung in einer `TextBox` dargestellt.

65 Daten aus der Registry lesen und in die Registry schreiben

Über die Klasse `Registry` aus dem Namensraum `Microsoft.Win32` können Sie Daten aus der Windows-Registry lesen, Daten manipulieren, neue Daten in vorhandenen Registry-Schlüsseln erzeugen und neue Schlüssel erzeugen.

Basis für die Arbeit mit der Registry ist die Klasse `RegistryKey`, die einen (Unter-) Schlüssel der Registry repräsentiert. Über verschiedene Methoden können Sie den Schlüssel bearbeiten. Die Methode `GetValue` liest z.B. den Wert eines Eintrags in einem Schlüssel, über die Methode `SetValue` können Sie einen Wert schreiben. Die Methode `CreateSubKey` ermöglicht das Erzeugen eines Unterschlüssels, über `DeleteSubKey` können Sie einen Unterschlüssel löschen. Die `OpenSubKey`-Methode öffnet einen Schlüssel, der einem anderen Schlüssel untergeordnet ist.

Die Wurzel-Schlüssel der Registry erreichen Sie über die statischen Eigenschaften `ClassesRoot` (`HKEY_CLASSES_ROOT`), `CurrentConfig` (`HKEY_CURRENT_CONFIG`), `CurrentUser` (`HKEY_CURRENT_USER`), `DynData` (`HKEY_DYN_DATA`), `LocalMachine` (`HKEY_LOCAL_MACHINE`), `PerformanceData` (`HKEY_PERFORMANCE_DATA`) und `Users` (`HKEY_USERS`). Da diese Eigenschaften ein `RegistryKey`-Objekt referenzieren, können Sie über die `OpenSubKey`-Methode zu einem Unterschlüssel navigieren. Die einzelnen Schlüsselnamen des (relativen) Pfades geben Sie durch Backslashes getrennt an.

Das folgende Beispiel liest den Wert `DOC-PATH` des Schlüssels `HKEY_CURRENT_USER\Software\Microsoft\Office\10.0\Word\Options`. Das Programm benötigt den Import der Namensräume `System` und `Microsoft.Win32`.

```
RegistryKey regKey = Registry.CurrentUser.OpenSubKey(
    @"Software\Microsoft\Office\10.0\Word\Options");

string wordDocPath = null;
if (regKey != null)
{
    object regValue = regKey.GetValue("DOC-PATH");
    if (regValue != null)
    {
        wordDocPath = regValue.ToString();
        Console.WriteLine(wordDocPath);
    }
    else
    {
        Console.WriteLine("Wert nicht gefunden");
    }
}
else
{
    Console.WriteLine("Schlüssel nicht gefunden");
}
```

Listing 112: Lesen eines Registry-Eintrags

Beim Schreiben sollten Sie für jeden Unterschlüssel überprüfen, ob dieser existiert, und den Schlüssel gegebenenfalls anlegen, bis Sie am letzten Schlüssel angelangt sind und dort den Wert schreiben können. Da das Ganze für die Praxis ein wenig kompliziert ist, habe ich eine Klasse `RegUtils` entwickelt, die den Registry-Zugriff erleichtert.

Den Methoden dieser Klasse wird der Registry-Pfad in der unter Windows üblichen Form übergeben, wobei der Wurzel-Schlüssel allerdings separat übergeben wird. Ein solcher Pfad sieht dann z.B. so aus:

`Software\Microsoft\Office\10.0\Word\Options\DOC-PATH`

Um den Wurzel-Schlüssel angeben zu können, habe ich zunächst eine Aufzählung implementiert:

```
public enum RegistryRootKeys
{
    HKEY_CLASSES_ROOT,
    HKEY_CURRENT_CONFIG,
    HKEY_CURRENT_USER,
    HKEY_DYN_DATA,
    HKEY_LOCAL_MACHINE,
    HKEY_PERFORMANCE_DATA,
    HKEY_USERS
}
```

Listing 113: Aufzählung zur Angabe der Registry-Wurzelschlüssel

Die Methoden der Klasse `RegUtils` fangen mögliche Ausnahmen (z.B. die Ausnahme vom Typ `UnauthorizedAccessException` bei einem unautorisierten Zugriff) nicht ab, sodass diese an den Aufrufer weitergereicht werden.

Die erste (private) Methode `GetRegistryRootKey` gibt eine Referenz auf das zu einem Wert der `RegistryRootKeys`-Aufzählung passende `RegistryKey`-Objekt zurück. Diese Methode wird in den anderen Methoden der Klasse aufgerufen.

```
public class RegUtils
{
    private static RegistryKey GetRegistryRootKey(RegistryRootKeys rootKey)
    {
        // Ermitteln des Registry-Wurzel-Schlüssels
        RegistryKey regKey = null;
        switch (rootKey)
        {
            case RegistryRootKeys.HKEY_CLASSES_ROOT:
                regKey = Registry.ClassesRoot;
                break;
            case RegistryRootKeys.HKEY_CURRENT_CONFIG:
                regKey = Registry.CurrentConfig;
                break;
        }
    }
}
```

Listing 114: Methode zum Ermitteln einer Referenz auf ein Wurzel-RegistryKey-Objekt

```

        case RegistryRootKeys.HKEY_CURRENT_USER:
            regKey = Registry.CurrentUser;
            break;
        case RegistryRootKeys.HKEY_DYN_DATA:
            regKey = Registry.DynData;
            break;
        case RegistryRootKeys.HKEY_LOCAL_MACHINE:
            regKey = Registry.LocalMachine;
            break;
        case RegistryRootKeys.HKEY_PERFORMANCE_DATA:
            regKey = Registry.PerformanceData;
            break;
        case RegistryRootKeys.HKEY_USERS:
            regKey = Registry.Users;
            break;
    }

    return regKey;
}

```

Listing 114: Methode zum Ermitteln einer Referenz auf ein Wurzel-RegistryKey-Objekt (Forts.)

Die Methode zum Lesen eines Registry-Eintrags (`ReadValue`) ruft zunächst `GetRegistryRootKey` auf um das zum übergebenen Wurzel-Schlüssel-Wert passende `RegistryKey`-Objekt zu referenzieren. Über diese Referenz ruft `ReadValue` dann die `OpenSubKey`-Methode unter der Übergabe des Registry-Pfades auf. Wird der Schlüssel gefunden, liest die Methode den Wert des übergebenen Wertnamens. Im Erfolgsfall liefert `ReadValue` den gelesenen Wert zurück. Wird der Wert nicht gefunden, gibt diese Methode den übergebenen Defaultwert zurück.

```

public static object ReadValue(RegistryRootKeys rootKey, string keyPath,
    string valueName, object defaultValue)
{
    // Schlüssel ermitteln
    RegistryKey regKey = GetRegistryRootKey(rootKey).OpenSubKey(keyPath);

    // Wert auslesen, wenn der Unterschlüssel gefunden wurde
    object regValue = defaultValue;
    if (regKey != null)
    {
        regValue = regKey.GetValue(valueName);
    }

    return regValue;
}

```

Listing 115: Methode zum Lesen eines Registry-Eintrags

`ReadValue` gibt (entsprechend der `GetValue`-Methode des `RegistryKey`-Objekts) einen `String` zurück, wenn der Registry-Eintrag vom Typ `REG_SZ` oder `REG_EXPAND_SZ` ist. Bei einem mehrteiligen `String` (`REG_MULTI_SZ`) wird ein `String-Array` mit den einzelnen Teilen zurückgegeben.

Ein numerischer Wert (REG_DWORD) wird als `int` zurückgegeben, binäre Daten (REG_BINARY) als Byte-Array.

Ich habe bewusst bei dieser Methode auf die Erzeugung einer Ausnahme für den Fall, dass der Eintrag nicht gefunden wurde, verzichtet. Durch die Übergabe eines geeigneten Defaultwerts können Sie recht flexibel auf das Nichtvorhandensein eines Eintrags reagieren. Übergeben Sie beim Aufruf dieser Methode als Defaultwert `null`, können Sie problemlos entscheiden, ob der Eintrag gefunden wurde. Gibt `ReadValue` in diesem Fall `null` zurück, wurde der Eintrag nicht gefunden (für leere, aber vorhandene Einträge gibt `ReadValue` je nach Datentyp des Eintrags einen Leerstring, 0 oder ein leeres Array zurück).

Die Methode zum Schreiben von Einträgen (`WriteValue`) arbeitet ähnlich. Übergeben wird wie bei `ReadValue` ein Wert der Aufzählung `RegistryRootKeys`, der den Wurzel-Schlüssel bezeichnet, der Pfad des Schlüssels und der Name des Eintrags. Dann folgen der Wert des Eintrags und eine Information darüber, ob `WriteValue` Unterschlüssel anlegen soll, falls diese nicht gefunden wurden.

`WriteValue` ermittelt zuerst eine Referenz auf das Wurzel-RegistryKey-Element und zerlegt den Pfad dann in seine Teile. Beim Durchgehen des Pfads wird ein nicht existierender Unterschlüssel über die `CreateSubKey`-Methode angelegt, falls das Argument `createIfNotExist` `true` ist. Der jeweilige Schlüssel muss dazu durch die Übergabe von `true` am zweiten Argument der `OpenSubKey`-Methode so geöffnet werden, dass auch ein Schreiben möglich ist. Schließlich wird noch der Wert geschrieben, wenn der letzte Unterschlüssel ermittelt werden konnte. Für den Fall, dass dieser nicht gefunden wird, erzeugt `WriteValue` eine Ausnahme.

```
public static void WriteValue(RegistryRootKeys rootKey, string keyPath,
    string valueName, object value, bool createIfNotExist)
{
    // Wurzel-Schlüssel ermitteln
    RegistryKey regKey = GetRegistryRootKey(rootKey);

    // Den Pfad in seine Einzelteile zerlegen
    string[] pathToken = keyPath.Split('\\');

    // Pfad durchgehen und den Schlüssel referenzieren
    RegistryKey subKey = null;
    for (int i = 0; i < pathToken.Length; i++)
    {
        if (regKey != null)
        {
            // Unterschlüssel zum Lesen und Schreiben öffnen
            subKey = regKey.OpenSubKey(pathToken[i], true);

            if (subKey == null && createIfNotExist)
            {
                // Unterschlüssel nicht gefunden: Schlüssel erzeugen
                // falls dies gewünscht ist
                subKey = regKey.CreateSubKey(pathToken[i]);
            }
        }
    }
}
```

Listing 116: Methode zum Schreiben eines Registry-Eintrags

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        }
        regKey = subKey;
    }
}

// Wert schreiben, wenn der Unterschlüssel gefunden wurde
if (regKey != null)
{
    regKey.SetValue(valueName, value);
}
else
{
    // Ausnahme werfen
    throw new Exception("Schlüssel " + rootKey.ToString() + "\\\" +
        keyPath + " nicht gefunden");
}
}

```

Listing 116: Methode zum Schreiben eines Registry-Eintrags (Forts.)

Nun fehlt noch eine Methode zum Löschen eines Eintrags. `DeleteValue` referenziert dazu zunächst den Schlüssel, der dem übergebenen Pfad entspricht. Wird der Schlüssel gefunden, ruft `DeleteValue` dessen `Delete`-Methode. Im negativen Fall erzeugt die Methode eine Ausnahme:

```

public static void DeleteValue(RegistryRootKeys rootKey, string keyPath,
    string valueName)
{
    // Unterschlüssel zum Schreiben öffnen
    RegistryKey regKey = GetRegistryRootKey(rootKey).OpenSubKey(
        keyPath, true);

    // Wert über den übergeordneten Schlüssel löschen
    // wenn dieser gefunden wurde
    if (regKey != null)
        regKey.DeleteValue(valueName, false);
    else
        // Schlüssel nicht gefunden: Ausnahme werfen
        throw new Exception("Schlüssel " + rootKey.ToString() + "\\\" +
            keyPath + " nicht gefunden");
}

```

Listing 117: Methode zum Löschen eines Werts in der Registry

Die Methode `DeleteKey` löscht schließlich einen ganzen Schlüssel. Da ein Schlüssel über die `DeleteSubKey`-Methode des übergeordneten Schlüssels gelöscht werden muss, zerlegt `DeleteKey` den übergebenen Pfad zunächst in den Pfad zum übergeordneten Schlüssel und den Namen des zu löschenden Schlüssels. Mit dem so ermittelten Pfad referenziert die Methode den übergeordneten Schlüssel und ruft dessen `DeleteSubKey`-Methode mit dem Namen des

Unterschlüssels auf. Für den Fall, dass der übergeordnete Schlüssel nicht gefunden wurde, erzeugt `DeleteKey` eine Ausnahme:

```
public static void DeleteKey(RegistryRootKeys rootKey, string keyPath)
{
    // Den Pfad zum übergeordneten Schlüssel und den Namen des zu löschenden
    // Schlüssels ermitteln
    int i = keyPath.LastIndexOf("\\");
    string parentKeyPath = keyPath.Substring(0, i);
    string keyName = keyPath.Substring(i + 1, keyPath.Length - i - 1);

    // Den dem zu löschenden Schlüssel übergeordneten Schlüssel zum
    // Schreiben öffnen
    RegistryKey regKey = GetRegistryRootKey(rootKey).OpenSubKey(
        parentKeyPath, true);

    if (regKey != null)
    {
        // Schlüssel über den übergeordneten Schlüssel löschen
        regKey.DeleteSubKey(keyName);
    }
    else
    {
        // Schlüssel nicht gefunden: Ausnahme werfen
        throw new Exception("Schlüssel " + rootKey.ToString() + "\\\" +
            keyPath + " nicht gefunden");
    }
}
```

Listing 118: Methode zum Löschen eines Schlüssels in der Registry

Damit ist die Klasse fertig. Listing 119 zeigt eine beispielhafte Anwendung dieser Klasse.

```
// Wert des Eintrags HKEY_CURRENT_USER\Software\Microsoft\Office\
// 10.0\Word\Options\DOC-PATH lesen
object wordDocPath = RegUtils.ReadValue(RegistryRootKeys.HKEY_CURRENT_USER,
    @"Software\Microsoft\Office\10.0\Word\Options", "DOC-PATH", null);
if (wordDocPath != null)
    Console.WriteLine(wordDocPath);
else
    Console.WriteLine("Eintrag nicht gefunden");

// Eintrag HKEY_CURRENT_USER\Software\Addison-Wesley\Codebook\Version so
// schreiben, dass der Eintrag inklusive allen Schlüsseln automatisch angelegt
// wird, falls er noch nicht existiert
RegUtils.WriteValue(RegistryRootKeys.HKEY_CURRENT_USER,
    @"Software\Addison-Wesley\Codebook", "Version", "1.0", true);

// Eintrag HKEY_CURRENT_USER\Software\Addison-Wesley\Codebook\Samples
// schreiben
```

Listing 119: Anwendung der Klasse `RegUtils`

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

RegUtils.SetValue(RegistryRootKeys.HKEY_CURRENT_USER,
    @"Software\Addison-Wesley\Codebook\Samples", "Registry",
    "c:\Samples\Registry", true);

// Schlüssel HKEY_CURRENT_USER\Software\Addison-Wesley\Codebook\Samples
// löschen
RegUtils.DeleteKey(RegistryRootKeys.HKEY_CURRENT_USER,
    @"Software\Addison-Wesley\Codebook\Samples");

```

Listing 119: Anwendung der Klasse RegUtils (Forts.)

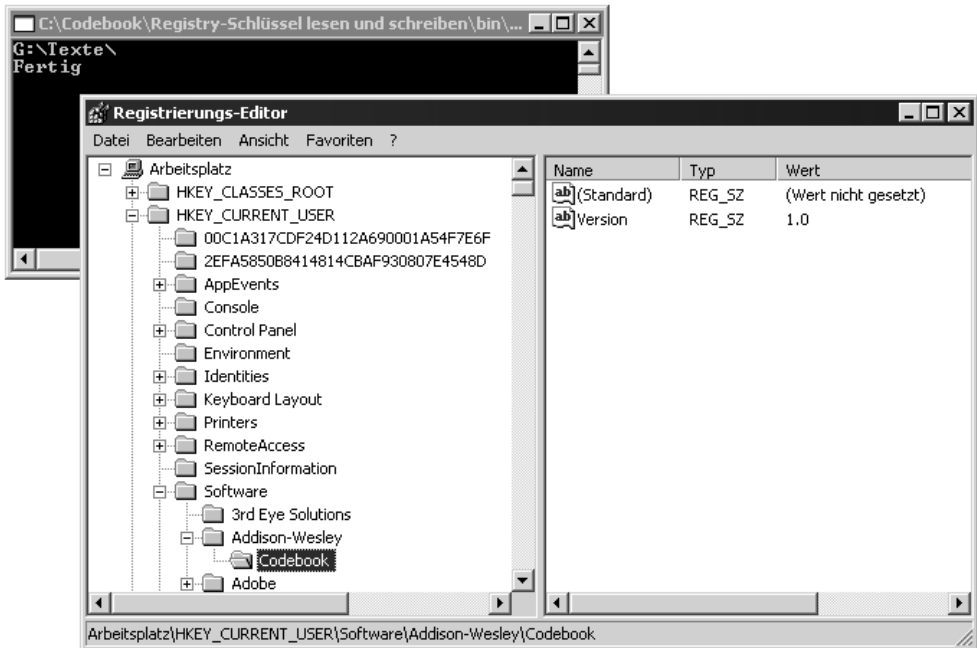


Abbildung 40: Das Beispielprogramm zum Lesen und Schreiben der Registry und der Registrierungs-Editor mit dem erzeugten Registry-Schlüssel

66 Verhindern, dass eine Anwendung mehrfach gestartet werden kann

In einigen Fällen ist es sinnvoll zu verhindern, dass eine Anwendung mehrfach gestartet werden kann. Das ist z. B. dann der Fall, wenn die Anwendung sehr langsam startet (weil beim Start umfangreiche Ressourcen geöffnet werden) oder wenn die Anwendung beim Start Ressourcen wie z. B. Dateien exklusiv öffnet (die dann beim zweiten Start nicht mehr geöffnet werden können bzw. dürfen).

Abfrage der gleichnamigen laufenden Prozesse mit einem Vergleich des Fenstertitels

Sie können den mehrfachen Start verhindern, indem Sie (idealerweise in der `Main`-Funktion) abfragen, ob bereits ein Prozess gleichen Namens wie die Anwendung ausgeführt wird. Dazu

können Sie die `GetProcessesByName`-Methode der Klasse `System.Diagnostics.Process` einsetzen. Dieser Methode übergeben Sie den Namen des aktuellen Prozesses, den Sie über `System.Diagnostics.Process.GetCurrentProcess().ProcessName` ermitteln. `GetProcessesByName` gibt ein Array von `Process`-Objekten zurück, das alle aktuell laufenden Prozesse repräsentiert, die den angegebenen Namen tragen. Da der aktuelle Prozess mit aufgeführt wird, könnten Sie die Länge des Arrays mit `> 1` vergleichen um herauszufinden, ob mehr als ein Prozess mit dem Namen des aktuellen Prozesses ausgeführt wird.

Leider ist dieses Vorgehen nicht sicher. Der Prozessname entspricht nämlich dem Namen der Assembly (also dem Namen der ausführbaren Datei). Wird gerade eine *andere* Anwendung mit *demselben* Dateinamen ausgeführt, wird dieser Prozess auch aufgeführt.

Als Lösung können Sie die gefundenen Prozesse durchgehen und den Fenstertitel vergleichen. Um den aktuellen Prozess auszuschließen, vergleichen Sie die Prozess-Id der Prozesse. Die Methode `RunningInstance` in Listing 120 geht auf diese Weise vor. Sie gibt eine Referenz auf den gefundenen Prozess zurück, falls ein Prozess mit demselben Namen und demselben Fenstertitel gefunden wird wie der aktuelle. Über diese Referenz können Sie die andere Anwendung dann z.B. in den Vordergrund holen (siehe Rezept 67). Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Diagnostics` importieren.

```
public static Process RunningInstance()
{
    // Alle Prozesse mit dem Namen des aktuellen Prozesses holen
    System.Diagnostics.Process[] processes = Process.GetProcessesByName(
        Process.GetCurrentProcess().ProcessName);

    // Alle Prozesse durchgehen und den Fenstertitel vergleichen
    string mainWindowTitle = Process.GetCurrentProcess().MainWindowTitle;
    for (int i = 0; i < processes.Length; i++)
    {
        // Den aktuellen Prozess ausschließen
        if (processes[i].Id != Process.GetCurrentProcess().Id)
        {
            if (processes[i].MainWindowTitle == mainWindowTitle)
            {
                // Prozess gefunden, diesen zurückgeben
                return processes[i];
            }
        }
    }

    // Kein Prozess gefunden, der dem aktuellen entspricht, also null
    // zurückgeben
    return null;
}
```

Listing 120: Verhindern, dass eine Anwendung mehrfach gestartet werden kann in einer einfachen Variante

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Listing 121 zeigt eine Anwendung der `RunningInstance`-Methode in einer Windows-Anwendung mit Formularen. Da der Fenstertitel eines Formulars erst definiert ist, wenn das Formular auf dem Bildschirm erscheint, kann `RunningInstance` nicht in der `Main`-Methode oder im `Load`-Ereignis, sondern erst im `Activated`-Ereignis verwendet werden.

```
private void StartForm_Activated(object sender, System.EventArgs e)
{
    // Überprüfen, ob die Anwendung bereits ausgeführt wird.
    if (RunningInstance() != null)
    {
        // Die Anwendung wird bereits ausgeführt, also eine Meldung ausgeben
        // und die Anwendung beenden
        MessageBox.Show("Dieses Programm kann nicht mehrfach ausgeführt " +
            "werden.", Application.ProductName, MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation);
        this.Close();
    }
}
```

Listing 121: Anwendung der `RunningInstance`-Methode in einem Formular

In einer Konsolenanwendung funktioniert die Methode `ApplicationCurrentlyRunning` übrigens auch relativ sicher, da der Fenstertitel einer solchen normalerweise der Dateiname der Anwendung¹ oder der volle Dateiname von `cmd.exe` mit der Angabe des Dateinamens der Exe-Datei (ohne Endung) ist².

ACHTUNG

Das unter Windows XP kompilierte Programm legte bei der Ausführung unter Windows 2000 ein eigenartiges Verhalten an den Tag: Es funktionierte nicht. Die Anwendung konnte problemlos mehrfach gestartet werden. Das Eigenartige daran war, dass das Programm auf einmal einwandfrei arbeitete, nachdem ich es unter Windows 2000 mit dem Kommandozeilencompiler neu kompiliert hatte (natürlich ohne eine Veränderung). Unter Windows 98 trat ebenfalls eine nicht erklärbares Verhalten auf: Das Programm merkte zwar beim einem erneuten Start, dass es mehrfach gestartet werden sollte, die `MessageBox` erschien aber immer wieder erneut, nachdem diese geschlossen wurde. Dasselbe gilt übrigens für das Programm im nächsten Abschnitt, nicht aber für die Lösung mit einem `Mutex` am Schluss dieses Abschnitts.

Abfrage der gleichnamigen laufenden Prozesse mit einem Vergleich des Dateinamens

Die Methode `RunningInstance` in Listing 120 ist leider nicht fehlerfrei. Die Abfrage des Fenstertitels kann nämlich problematisch werden, wenn Ihre Anwendung den Titel dynamisch anpasst. Als Lösung dieses Problems könnten Sie statt des Fenstertitels den Dateinamen der Exe-Datei des Prozesses abfragen, wie es Listing 122 zeigt. Diese Variante setzt allerdings

-
1. beim Start über die Windows-Oberfläche
 2. beim Start über die Konsole

voraus, dass das Programm nur ein einziges Mal auf dem Rechner gespeichert ist und folglich nicht aus verschiedenen Ordnern heraus gestartet werden kann. Zum Kompilieren benötigen Sie den Import der Namensräume `System` und `System.Diagnostics`.

```
public static Process RunningInstance()
{
    // Alle Prozesse mit dem Namen des aktuellen Prozesses holen
    System.Diagnostics.Process[] processes =
        Process.GetProcessesByName(
            Process.GetCurrentProcess().ProcessName);

    // Alle Prozesse durchgehen und den Dateinamen vergleichen
    for (int i = 0; i < processes.Length; i++)
    {
        // Den aktuellen Prozess ausschließen
        if (processes[i].Id != Process.GetCurrentProcess().Id)
        {
            // Den Namen der Exe-Datei vergleichen
            if (processes[i].MainModule.FileName ==
                Process.GetCurrentProcess().MainModule.FileName)
                return processes[i];
        }
    }

    // Kein Prozess gefunden, der dem aktuellen entspricht, also null
    // zurückgeben
    return null;
}
```

Listing 122: Mehrfachstart über die Abfrage des Dateinamens der Exe-Datei verhindern

Da bei dieser Variante nicht der Fenstertitel des Hauptfensters abgefragt wird, können Sie die Abfrage auch in der `Main`-Methode bzw. im `Load`-Ereignis des Startformulars unterbringen.

Verhinderung des Mehrfachstarts über einen Mutex

Alternativ zur Abfrage der laufenden Prozesse können Sie auch einen Mutex verwenden, um den mehrfachen Start einer Anwendung zu verhindern. Ein Mutex ist ein Systemobjekt, das eigentlich für das (prozessübergreifende) Synchronisieren von Threads verwendet wird. Bei der Erzeugung eines Mutex können Sie diesem einen systemweiten Namen zuweisen.

Ein Thread kann über die `WaitOne`-Methode eines Mutex-Objekts dieses in einen »signalisierten« Zustand versetzen. Ist das Objekt gerade in diesem Zustand, wartet `WaitOne` so lange, bis der andere Thread den Mutex über die `ReleaseMutex`-Methode freigegeben hat (oder bis zu einem definierten Timeout). `WaitOne` gibt `true` zurück, wenn der Mutex in den Besitz des Thread gelangt ist. Nach dem erfolgreichen Aufruf von `WaitOne` kann der Thread dann auf externe Ressourcen zugreifen, ohne die Gefahr einzugehen, dass andere Threads, die denselben Mutex verwenden, gleichzeitig auch darauf zugreifen. Der Mutex wird schließlich über `ReleaseMutex` wieder freigegeben, damit andere Threads diesen über `WaitOne` sperren können.

Um den Mehrfachstart einer Anwendung zu verhindern, können Sie einen Mutex zweckentfremden. Dazu erzeugen Sie ein `Mutex`-Objekt, dem Sie im Konstruktor am zweiten Argument einen systemweit eindeutigen Namen übergeben. Die Methode `OtherInstanceRunning` in Listing 133 verwendet als Name den Anwendungsnamen mit dem Zusatz »_MultiStartPrevent«. Am ersten Argument können Sie festlegen, ob der Mutex direkt in den Besitz des Thread gelangen soll (falls dies möglich ist). Da dies in der Praxis wenig Sinn macht (weil Sie dazu auch `WaitOne` aufrufen können), übergibt `OtherInstanceRunning` an diesem Argument `false`.

Dann können Sie über `WaitOne` den Besitz des Mutex anfordern. `OtherInstanceRunning` übergibt am ersten Argument als Timeout 0, damit `WaitOne` direkt zurückkehrt. Am zweiten Argument wird `true` übergeben, was hier keine weitere Bedeutung besitzt. Über dieses Argument können Sie bei Anwendungen, die vor dem Warten auf den Mutex den Zugriff auf Objekte oder Klassen über `lock` (bzw. über die `Monitor`-Klasse) sperren, festlegen, dass die Sperrung vor dem Warten aufgehoben und nach dem Warten wieder gesetzt wird. Damit verhindern Sie Deadlocks, wenn Sie gleichzeitig über `lock` sperren und Mutexe verwenden.

`OtherInstanceRunning` fragt dann einfach die Rückgabe von `WaitOne` ab um zu entscheiden, ob der Mutex in den Besitz des aktuellen Thread gelangt ist. Der Mutex wird dann auch nicht explizit freigegeben, damit dieser so lange signalisiert bleibt, wie die Anwendung geöffnet ist. Beim Beenden der Anwendung wird der Mutex automatisch freigegeben.

Zum Kompilieren der Methode müssen Sie die Namensräume `System` und `System.Threading` einbinden.

```

/* Variable für den Mutex, der von der Anwendung erzeugt wird */
private static Mutex mutex;

/* Methode zur Ermittlung, ob bereits eine Instanz der Anwendung ausgeführt
 * wird */
public static bool OtherInstanceRunning()
{
    // Mutex mit einem systemweit eindeutigen Namen erzeugen
    string mutexName = System.Windows.Forms.Application.ProductName +
        "_MultiStartPrevent";
    mutex = new Mutex(false, mutexName);

    // Signalisieren des Mutex und gleichzeitig abfragen, ob bereits ein
    // gleichnamiger Mutex existiert
    if (mutex.WaitOne(0, true))
        return false;
    else
        return true;
}

```

Listing 123: Mehrfachstart einer Anwendung über einen Mutex verhindern

Die `OtherInstanceRunning`-Methode können Sie nun auch in der `Main`-Methode einsetzen:

```
static void Main()
{
    // Überprüfen, ob die Anwendung bereits ausgeführt wird
    if (OtherInstanceRunning())
    {
        // Die Anwendung wird bereits ausgeführt, also eine Meldung ausgeben
        // und die Anwendung beenden
        MessageBox.Show("Dieses Programm kann nicht mehrfach ausgeführt " +
            "werden.", Application.ProductName, MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation);
    }
    else
    {
        /* Anwendung mit dem Startformular starten */
        Application.Run(new StartForm());
    }
}
```

Listing 124: Verhindern des Mehrfachstarts über `OtherInstanceRunning`

67 Aktivieren einer laufenden Anwendung

Aktivieren einer laufenden Anwendung über `AppActivate`

Wenn Sie eine laufende Anwendung aktivieren wollen, was besonders nach dem Verhindern des mehrfachen Starts einer Anwendung sinnvoll sein kann, können Sie eigentlich die zu Visual Basic gehörende Methode `AppActivate` des `Interaction`-Moduls aus dem Namensraum `Microsoft.VisualBasic` verwenden. Dazu müssen Sie lediglich die Assembly *Microsoft.VisualBasic.dll* referenzieren. In Visual Studio finden Sie diese unter dem Komponentennamen *Microsoft Visual Basic .NET Runtime*.

`AppActivate` können Sie den Fenstertitel der zu aktivierenden Anwendung oder eine Win32-Prozess-Id übergeben. Wird keine Anwendung mit dem übergebenen Titel bzw. der übergebenen Prozess-Id gefunden, erzeugt `AppActivate` eine Ausnahme.

Das folgende Beispiel aktiviert die Anwendung mit dem Fenstertitel »Codebook.doc – Microsoft Word«. Das Beispiel erfordert die Referenzierung der Assembly *Microsoft.VisualBasic.dll* und den Import der Namensräume `System` und `Microsoft.VisualBasic`.

```
try
{
    Interaction.AppActivate("Codebook.doc - Microsoft Word");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

Listing 125: Aktivieren einer Anwendung über `AppActivate`

Leider ist `AppActivate` nicht in der Lage, eine minimierte Anwendung nach dem Aktivieren in den normalen Modus wiederherzustellen. Die Anwendung wird zwar aktiviert, bleibt aber minimiert. Im folgenden Listing finden Sie die Lösung dieses Problems über die Verwendung von Windows-API-Funktionen.

Aktivieren einer laufenden Anwendung über API-Funktionen

Um die Aktivierung auch von Anwendungen zu ermöglichen, deren Hauptfenster minimiert ist, habe ich eine Methode `ActivateApplication` entwickelt, die mit Windows-API-Funktionen arbeitet.

Zum Kompilieren des Quellcodes müssen Sie zunächst die Namensräume `System`, `System.Text` und `System.Runtime.InteropServices` importieren und die benötigten API-Funktionen deklarieren:

```
[DllImport("user32", SetLastError=true)]
private static extern int FindWindow(string lpClassName, string lpWindowName);

[DllImport("user32", SetLastError=true)]
public static extern int SetForegroundWindow(IntPtr hWnd);

[DllImport("user32", SetLastError=true)]
public static extern int IsIconic(IntPtr hWnd);

[DllImport("user32", SetLastError=true)]
public static extern int OpenIcon(IntPtr hWnd);

[DllImport("user32", SetLastError=true)]
private static extern int GetClassName(IntPtr hWnd, StringBuilder lpClassName,
    int nMaxCount);
```

Listing 126: Deklaration der benötigten API-Funktionen

`ActivateApplication` sucht zunächst über die Funktion `FindWindow` ein Fenster, das den übergebenen Fenstertitel und/oder Klassennamen trägt. Der Klassenname ist der Windows-interne Name der Klasse des Fensters. Wenn Sie für den Klassennamen `null` übergeben, sucht `FindWindow` ein beliebiges Fenster mit dem übergebenen Titel. Ich habe den Klassennamen mit in `ActivateApplication` aufgenommen, da es möglich ist, dass zwei verschiedene Fenster denselben Titel tragen. Über den Klassennamen können Sie das Fenster bei Bedarf genauer spezifizieren. Um diesen herauszufinden, können Sie ein Tool wie das Microsoft-Werkzeug *Spy* verwenden (das ein Teil des Plattform-SDK ist). Um das Ganze zu vereinfachen, habe ich aber auch eine Hilfsmethode `GetWindowClassName` entwickelt, der ein Fenstertitel übergeben wird und die den Klassennamen zurückgibt.

`FindWindow` gibt den Handle des Fensters zurück, wenn eines gefunden wurde. In diesem Fall überprüft `ActivateApplication` über die API-Funktion `IsIconic`, ob das Fenster minimiert ist. Ist dies der Fall, wird das Fenster über `OpenIcon` wiederhergestellt. Danach ruft `ActivateApplication` die API-Funktion `SetForegroundWindow` auf um das Fenster in den Vordergrund zu holen.

```

public static void ActivateApplication(string mainWindowClassName,
    string mainWindowTitle)
{
    // Fenster mit dem übergebenen Klassennamen und Titel suchen
    int windowHandle = FindWindow(mainWindowClassName, mainWindowTitle);

    if (windowHandle != 0)
    {
        // Fenster gefunden: Überprüfen, ob das Fenster verkleinert ist
        if (IsIconic((IntPtr)windowHandle) != 0)
        {
            // Fenster wiederherstellen
            if (OpenIcon((IntPtr)windowHandle) == 0)
            {
                throw new Exception("Windows-Fehler " +
                    Marshal.GetLastWin32Error() + " beim Wiederherstellen " +
                    "des Fensters mit dem Titel '" + mainWindowTitle +
                    "' aus dem minimierten Zustand.");
            }
        }

        // Fenster in den Vordergrund holen
        if (SetForegroundWindow((IntPtr)windowHandle) == 0)
        {
            throw new Exception("Windows-Fehler " +
                Marshal.GetLastWin32Error() + " beim Nach-vorne-Holen " +
                "des Fensters mit dem Titel '" + mainWindowTitle + "'");
        }
    }
    else
    {
        throw new Exception("Fenster mit Titel '" + mainWindowTitle +
            "' nicht gefunden");
    }
}

/* Hilfs-Methode zum Auslesen des Klassennamens eines Fensters */
public static string GetWindowClassName(string windowTitle)
{
    int windowHandle = FindWindow(null, windowTitle);
    if (windowHandle != 0)
    {
        StringBuilder className = new StringBuilder(255);
        if (GetClassName((IntPtr)windowHandle, className, 255) > 0)
        {
            return className.ToString();
        }
        else
        {
            throw new Exception("Windows-Fehler " +

```

Listing 127: Methoden zum Aktivieren einer Anwendung und zum Auslesen des Klassennamens eines Fensters

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        Marshal.GetLastWin32Error() + "beim Ermitteln des " +
        " Klassennamens des Fensters mit dem Titel '" +
        windowTitle + "'");
    }
}
else
{
    throw new Exception("Fenster mit dem Titel '" + windowTitle +
        "' nicht gefunden");
}
}

```

Listing 127: Methoden zum Aktivieren einer Anwendung und zum Auslesen des Klassennamens eines Fensters (Forts.)

Alle verwendeten API-Funktionen liefern im Fehlerfall 0 zurück. In diesen Fällen generieren `ActivateApplication` und `GetWindowClassName` eine Ausnahme. Um den zugrunde liegenden Windows-Fehler zu ermitteln, rufen beide Methoden `Marshal.GetLastWin32Error` auf, die den zuletzt aufgetretenen Windows-Fehler ausliest. Ich habe bewusst darauf verzichtet, den Windows-Fehlercode über die API-Funktion `FormatMessage` in eine lesbare Fehlermeldung umzuwandeln um die Methoden übersichtlich zu gestalten und weil Windows-Fehler bei der Anwendung der eingesetzten Funktionen wohl eher selten auftreten.

Das folgende Listing zeigt eine Anwendung der `ActivateApplication`-Methode, bei der das Fenster des Mozilla-Browsers aktiviert wird:

```

try
{
    ActivateApplication("MozillaWindowClass", "Mozilla");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 128: Beispiel-Anwendung der `ActivateApplication`-Methode

Aktivieren einer laufenden Anwendung über ein `Process`-Objekt

Wenn Sie ein `Process`-Objekt für eine Anwendung besitzen (z.B. weil Sie die Methode `RunningInstance` aus dem Rezept 66 aufgerufen haben), können Sie diese ebenfalls in den Vordergrund holen. Dazu lesen Sie den `Handle` des Hauptfensters über die Eigenschaft `MainWindowHandle` aus und setzen die API-Funktionen des vorhergehenden Rezepts ein. Listing 129 zeigt dies am Beispiel der Methode `ActivateApplication`. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Diagnostics`, `System.Text` und `System.Runtime.InteropServices` einbinden.

```

[DllImport("user32", SetLastError=true)]
public static extern int SetForegroundWindow(IntPtr hwnd);

[DllImport("user32", SetLastError=true)]
public static extern int IsIconic(IntPtr hwnd);

[DllImport("user32", SetLastError=true)]
public static extern int OpenIcon(IntPtr hwnd);

public static void ActivateApplication(Process process)
{
    if ((int)process.MainWindowHandle != 0)
    {
        if (IsIconic((IntPtr)process.MainWindowHandle) != 0)
        {
            // Hauptfenster nach vorne holen
            if (OpenIcon(process.MainWindowHandle) == 0)
            {
                throw new Exception("Windows-Fehler " +
                    Marshal.GetLastWin32Error() + " beim Wiederherstellen " +
                    "des Fensters für den Prozess " + process.ProcessName +
                    "' aus dem minimierten Zustand.");
            }
        }

        // Fenster in den Vordergrund holen
        if (SetForegroundWindow(process.MainWindowHandle) == 0)
        {
            throw new Exception("Windows-Fehler " +
                Marshal.GetLastWin32Error() + " beim Nach-vorne-Holen " +
                "des Hauptfensters des Prozesses '" + process.ProcessName +
                "'");
        }
    }
    else
    {
        throw new Exception("Der Prozess '" + process.ProcessName +
            " besitzt kein Hauptfenster");
    }
}

```

Listing 129: Aktivieren einer Anwendung über ein Process-Objekt

68 Auflisten aller laufenden Prozesse eines Computers

Über die `GetProcesses`-Methode der `Process`-Klasse erhalten Sie ein Array von `Process`-Objekten für alle aktuell ausgeführten Prozesse eines Computers. Wenn Sie dieser Methode nichts übergeben, verwaltet das zurückgegebene Array Objekte für alle lokalen Prozesse. Alternativ können Sie dieser Methode den Namen eines Computers im Netzwerk übergeben, um dessen Prozesse aufzulisten. Die Übergabe einer IP-Adresse oder des Namens *localhost* ist (eigenartigerweise) leider nicht möglich.

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

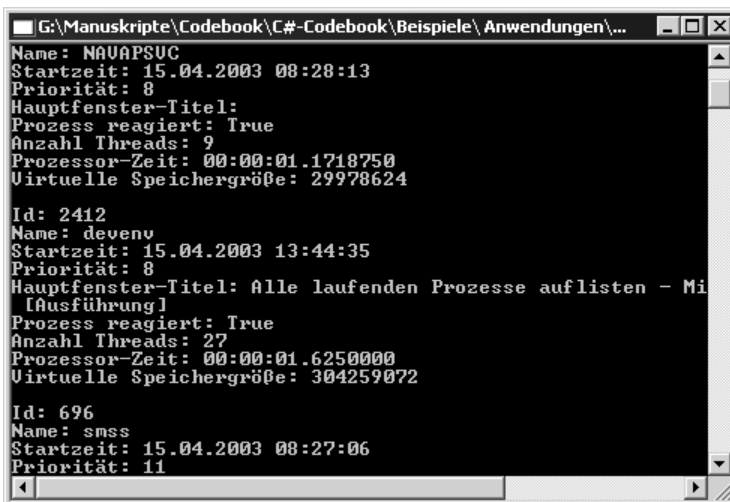
Multi-
mediaAutom-
tionSons-
tiges

Die im Array verwalteten `Process`-Objekte besitzen eine Vielfalt an Eigenschaften und Methoden. Die interessantesten setzt der Beispiel-Programmcode in Listing 130 ein. Zum Kompilieren des Programms müssen Sie die Namensräume `System` und `System.Diagnostics` importieren.

```
// Alle auf dem lokalen Computer laufenden Prozesse ermitteln
Process[] processes = Process.GetProcesses();

// Die Prozesse durchgehen und Informationen dazu anzeigen
for (int i = 0; i < processes.Length; i++)
{
    Console.WriteLine("Id: {0}", processes[i].Id);
    Console.WriteLine("Name: {0}", processes[i].ProcessName);
    Console.WriteLine("Startzeit: {0}", processes[i].StartTime.ToString());
    Console.WriteLine("Priorität: {0}", processes[i].BasePriority);
    Console.WriteLine("Hauptfenster-Titel: {0}", processes[i].MainWindowTitle);
    Console.WriteLine("Prozess reagiert: {0}", processes[i].Responding);
    Console.WriteLine("Anzahl Threads: {0}", processes[i].Threads.Count);
    Console.WriteLine("Prozessor-Zeit: {0}", processes[i].TotalProcessorTime);
    Console.WriteLine("Virtuelle Speichergröße: {0}",
        processes[i].VirtualMemorySize);
    Console.WriteLine();
}
}
```

Listing 130: Auflisten der wichtigsten Eigenschaften aller Prozesse des lokalen Computers



```
G:\Manuskripte\Codebook\C#-Codebook\Beispiele\Anwendungen\...
Name: NAUAPSUC
Startzeit: 15.04.2003 08:28:13
Priorität: 8
Hauptfenster-Titel:
Prozess reagiert: True
Anzahl Threads: 9
Prozessor-Zeit: 00:00:01.1718750
Virtuelle Speichergröße: 29978624

Id: 2412
Name: devenv
Startzeit: 15.04.2003 13:44:35
Priorität: 8
Hauptfenster-Titel: Alle laufenden Prozesse auflisten - Mi
[Ausführung]
Prozess reagiert: True
Anzahl Threads: 27
Prozessor-Zeit: 00:00:01.6250000
Virtuelle Speichergröße: 304259072

Id: 696
Name: smss
Startzeit: 15.04.2003 08:27:06
Priorität: 11
```

Abbildung 41: Das Programm zum Auflisten aller Prozesse eines Computers in Aktion

Wenn Sie die Prozesse eines Computers im Netzwerk auflisten wollen, geben Sie einfach dessen Namen an:

```
Process[] processes = Process.GetProcesses("Trillian");  
...
```

69 Auflisten aller laufenden Anwendungen eines Computers

Das Auflisten aller laufenden Anwendungen ist nicht auf eine einfache Weise möglich. Problematisch daran ist, dass Sie zwar die laufenden Prozesse ermitteln können, aber nicht ohne weiteres erkennen, welcher Prozess eine echte Anwendung und welcher ein Hintergrundprozess ist.

Zum Ermitteln der laufenden Prozesse können Sie wie im vorhergehenden Rezept die `GetProcesses`-Methode der `Process`-Klasse (aus dem Namensraum `System.Diagnostics`) aufrufen, die ein Array von `Process`-Objekten zurückgibt. Die Lösung, Anwendungen von Hintergrundprozessen zu unterscheiden, basiert nun auf der Idee, dass Anwendungen ein Haupt-Fenster besitzen. Da Sie den Handle dieses Fensters aus der Eigenschaft `MainWindowHandle` auslesen können, ist es ein Leichtes, Anwendungen (mit einer relativ hohen Sicherheit) von Hintergrundprozessen zu unterscheiden.

Das Beispiel erfordert den Import der Namensräume `System` und `System.Diagnostics`.

```
// Alle laufenden Prozesse ermitteln  
Process[] processes = Process.GetProcesses();  
  
// Die Prozesse durchgehen und die Prozesse mit einem Haupt-Fenster ermitteln  
for (int i = 0; i < processes.Length; i++)  
{  
    if ((int)processes[i].MainWindowHandle != 0)  
    {  
        // Hauptfenster ist vorhanden, der Prozess ist also eine Anwendung:  
        // Prozessname und Titel des Hauptfensters auslesen  
        Console.WriteLine("Prozessname: {0}\r\nFenstertitel: {1}\r\n",  
            processes[i].ProcessName, processes[i].MainWindowTitle);  
    }  
}
```

Listing 131: Ermitteln aller laufenden Anwendungen

Das Beispiel liest bei den gefundenen Anwendungen den Prozessnamen über die Eigenschaft `ProcessName` und den Titel des Hauptfensters über die Eigenschaft `MainWindowTitle` aus.



Abbildung 42: Das Programm zum Auflisten aller Anwendungen in Aktion

Hinweis

Der Programmcode ist leider nicht absolut sicher, da einige (wenige) Hintergrundprozesse ein (unsichtbares) Hauptfenster besitzen und diese dann auch mit aufgelistet werden.

70 Informationen über die Dienste eines Computers auslesen

Über die Methode `GetServices` der Klasse `System.ServiceProcess.ServiceController` erhalten Sie ein Array von Referenzen auf `ServiceController`-Instanzen, die die Dienste eines Computers repräsentieren. Rufen Sie `GetServices` ohne Argument oder mit einem Punkt als Computernamen auf, erhalten Sie die Dienste des lokalen Computers. Sie können aber auch den Namen eines entfernten Rechners angeben, um dessen Dienste auszulesen.

Die `ServiceController`-Objekte der einzelnen Dienste liefern eine Vielzahl an Informationen und bieten darüber hinaus sogar Möglichkeiten, Dienste zu starten, anzuhalten oder zu stoppen (siehe nächstes Rezept). Der folgende Quellcode listet die wichtigsten Daten der ermittelten Dienste auf.

Das Programm benötigt eine Referenz auf die Assemblies `System.ServiceProcess.dll` und `System.Windows.Forms.dll`, und die Einbindung der Namensräume `System`, `System.ServiceProcess` und `System.Windows.Forms`. Unter Windows 98 und Me kann das Programm nicht ausgeführt werden, da diese Systeme keine Dienste unterstützen.

```
string machineName = "."; // . = lokaler Rechner

// Dienste ermitteln
ServiceController[] services = null;
try
```

Listing 132: Auflisten von Informationen über die Dienste eines Computers

```

{
    services = ServiceController.GetServices(machineName);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Auslesen der Dienste " +
        "auf dem Computer " + machineName + ": " +
        ex.Message, Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

// Alle Dienste durchgehen und die wichtigsten Informationen ausgeben
for (int i = 0; i < services.Length; i++)
{
    Console.WriteLine();

    // Dienstname
    Console.WriteLine("Dienstname: {0}", services[i].ServiceName);

    // Anzeigename
    Console.WriteLine("Anzeigename: {0}", services[i].DisplayName);

    // Status
    Console.Write("Status: ");
    switch (services[i].Status)
    {
        case ServiceControllerStatus.Running:
            Console.WriteLine("Wird ausgeführt");
            break;
        case ServiceControllerStatus.Stopped:
            Console.WriteLine("Gestoppt");
            break;
        case ServiceControllerStatus.Paused:
            Console.WriteLine("Angehalten");
            break;
        case ServiceControllerStatus.StartPending:
            Console.WriteLine("Wird gestartet");
            break;
        case ServiceControllerStatus.ContinuePending:
            Console.WriteLine("Wird wieder gestartet");
            break;
        case ServiceControllerStatus.PausePending:
            Console.WriteLine("Wird angehalten");
            break;
        case ServiceControllerStatus.StopPending:
            Console.WriteLine("Wird beendet");
            break;
    }
}

```

Listing 132: Auflisten von Informationen über die Dienste eines Computers (Forts.)

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// Dienst-Typ
Console.WriteLine("Diensttyp: ");
switch (services[i].ServiceType)
{
    case ServiceType.Adapter:
        Console.WriteLine("Adapter");
        break;
    case ServiceType.FileSystemDriver:
        Console.WriteLine("Dateisystem-Treiber");
        break;
    case ServiceType.InteractiveProcess:
        Console.WriteLine("Interaktiver Prozess");
        break;
    case ServiceType.KernelDriver:
        Console.WriteLine("Kernel-Treiber");
        break;
    case ServiceType.RecognizerDriver:
        Console.WriteLine("Dateisystem-Ermittlungs-Treiber");
        break;
    case ServiceType.Win32OwnProcess:
        Console.WriteLine("Win32-Anwendung in eigenem Prozess");
        break;
    case ServiceType.Win32ShareProcess:
        Console.WriteLine("Win32-Anwendung in " +
            "gemeinsamen Prozess");
        break;
}
```

Listing 132: Auflisten von Informationen über die Dienste eines Computers (Forts.)

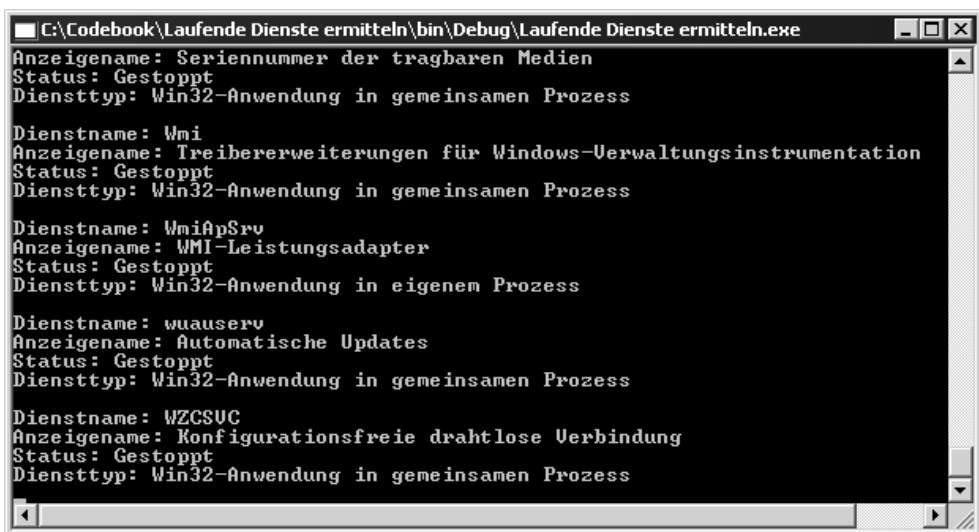


Abbildung 43: Auflisten von Informationen über die Dienste eines Computers

Neben den im Beispielprogramm dargestellten Informationen können Sie noch weitere Informationen abrufen, wie z. B., ob ein Dienst beendet oder angehalten werden kann (`CanPauseAndContinue`, `CanShutdown`, `CanStop`) oder von welchen anderen Diensten ein Dienst abhängig ist (`ServicesDependedOn`).

71 Dienste starten, anhalten, fortfahren und stoppen

Eine Instanz der Klasse `ServiceController` ermöglicht neben dem Auslesen von Informationen zu Diensten (siehe Rezept 70) auch, diese zu starten, anzuhalten, fortzufahren oder zu stoppen. Dazu verwenden Sie die Methoden `Start`, `Pause`, `Continue` und `Stop`.

Beim Erzeugen des `ServiceController`-Objekts geben Sie den Dienstnamen an. Im optionalen zweiten Argument können Sie den Namen des Computers angeben, auf dem der Dienst bearbeitet werden soll.

Beim Starten, Pausieren, Fortfahren und Stoppen müssen Sie einiges beachten. So kann ein Dienst nur erfolgreich gestartet werden, wenn er den Status `Stopped` besitzt. Sogar bei einem Dienst, der bereits ausgeführt wird, erzeugt die `Start`-Methode eine Ausnahme, was eigentlich unsinnig ist. Eine tolerantere Methode sollte den Status abfragen und in dem Fall, dass der Dienst bereits läuft oder gerade gestartet wird, einfach nichts machen. Ähnlich verhält es sich beim Stoppen, Pausieren oder Fortfahren eines Dienstes.

Daneben kann es einige Zeit dauern, bis ein Dienst gestartet, gestoppt wurde, pausiert oder fortgefahren ist. Die entsprechenden Methoden kehren aber sofort zurück, ohne auf die erfolgreiche Ausführung zu warten. Wenn Sie sich nach dem Aufruf der Methode darauf verlassen müssen, dass der Dienst seinen Status wie gewünscht geändert hat, müssen Sie über die Methode `WaitForStatus`, der Sie einen Dienst-Status und optional einen Timeout (in Form eines `TimeSpan`-Objekts) übergeben, darauf warten, dass der Dienst den gewünschten Status annimmt.

Die folgende Klasse `ServiceUtil` setzt dies um und stellt einige Methoden zur Arbeit mit Diensten zur Verfügung. Zum Kompilieren müssen Sie eine Referenz auf die Assembly `System.ServiceProcess.dll` anlegen und die Namensräume `System` und `System.ServiceProcess` einbinden. Ein Programm, das diese Klasse einsetzt, können Sie nicht unter Windows 98 und Me ausführen, da Dienste erst ab Windows NT unterstützt werden.

Die Methode `StartService` startet einen Dienst auf einem anzugebenden Computer (`null` oder ».« für den lokalen Rechner). Am dritten Argument kann ein String-Array übergeben werden, dessen Strings als Startargumente für den Dienst dienen. Die Methode akzeptiert aber auch `null`, wenn keine Argumente übergeben werden sollen. Das letzte Argument gibt einen Timeout in Millisekunden für den Start an. Nach dem Starten des Dienstes wartet `StartService` bis maximal zu diesem Timeout darauf, dass der Dienst den Status `Running` annimmt. `StartService` liefert den Status des Dienstes zurück, damit dieser nach dem Aufruf überprüft werden kann. Den Rest erläutern die Kommentare:

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
public class ServiceUtil
{
    /* Methode zum Starten eines Dienstes */
    public static ServiceControllerStatus StartService(string serviceName,
        string machineName, string[] arguments, int timeout)
    {
        // Dienst ermitteln
        if (machineName == null) machineName = ".";
        ServiceController service = new ServiceController(serviceName,
            machineName);

        // Dienst starten, falls dieser nicht bereits gestartet ist oder wird
        if (service.Status != ServiceControllerStatus.Running &&
            service.Status != ServiceControllerStatus.StartPending &&
            service.Status != ServiceControllerStatus.ContinuePending)
        {
            if (arguments != null)
                service.Start(arguments);
            else
                service.Start();
        }

        // Den übergebenen Timeout (in Millisekunden) warten,
        // bis der Dienst den Status Running besitzt
        service.WaitForStatus(ServiceControllerStatus.Running,
            new TimeSpan(0, 0, 0, timeout));

        // Status des Dienstes zurückliefern
        return service.Status;
    }
}
```

Listing 133: Methode zum toleranten Starten eines Dienstes

Die Methode `PauseService` besitzt ähnliche Argumente wie `StartService` (lediglich ohne die Dienst-Start-Argumente) und hält einen Dienst an. `PauseService` fragt zusätzlich zu dem aktuellen Status des Dienstes über die Eigenschaft `CanPauseAndContinue` ab, ob dieser angehalten werden kann, und erzeugt im negativen Fall eine Ausnahme. Nach dem Aufruf der `Pause`-Methode wartet `PauseService` maximal bis zum Ablauf des Timeout darauf, dass der Dienst den Status `Paused` annimmt. Zur Überprüfung liefert diese Methode wieder den Status des Dienstes zurück:

```
public static ServiceControllerStatus PauseService(string serviceName,
    string machineName, int timeout)
{
    // Dienst ermitteln
    if (machineName == null) machineName = ".";
    ServiceController service = new ServiceController(serviceName,
        machineName);
}
```

Listing 134: Methode zum toleranten Anhalten eines Dienstes

```

// Überprüfen, ob der Dienst angehalten werden kann
if (service.CanPauseAndContinue == false)
    throw new Exception("Der Dienst '" + serviceName +
        "' kann nicht angehalten werden");

// Dienst anhalten, falls dieser nicht bereits angehalten ist
if (service.Status != ServiceControllerStatus.Paused &&
    service.Status != ServiceControllerStatus.PausePending &&
    service.Status != ServiceControllerStatus.StopPending)
    service.Pause();

// Den übergebenen Timeout (in Millisekunden) warten,
// bis der Dienst den Status Paused besitzt
service.WaitForStatus(ServiceControllerStatus.Paused,
    new TimeSpan(0, 0, 0, timeout));

// Status des Dienstes zurückliefern
return service.Status;
}

```

Listing 134: Methode zum toleranten Anhalten eines Dienstes (Forts.)

Die Methode `ContinueService`, die der Methode `StartService` ähnlich ist, fährt einen angehaltenen Dienst fort:

```

public static ServiceControllerStatus ContinueService(string serviceName,
    string machineName, int timeout)
{
    // Dienst ermitteln
    if (machineName == null) machineName = ".";
    ServiceController service = new ServiceController(
        serviceName, machineName);

    // Überprüfen, ob der Dienst fortgefahren werden kann
    if (service.CanPauseAndContinue == false)
        throw new Exception("Der Dienst '" + serviceName +
            "' kann nicht fortgefahren werden");

    // Dienst fortfahren, falls dieser nicht bereits ausgeführt wird
    // oder gerade startet
    if (service.Status != ServiceControllerStatus.Running &&
        service.Status != ServiceControllerStatus.StartPending &&
        service.Status != ServiceControllerStatus.ContinuePending)
        service.Continue();

    // Den übergebenen Timeout (in Millisekunden) warten, bis der Dienst den
    // Status Running besitzt
    service.WaitForStatus(ServiceControllerStatus.Running,
        new TimeSpan(0, 0, 0, timeout));
}

```

Listing 135: Methode zum Fortfahren eines Dienstes

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
    // Status des Dienstes zurückliefern
    return service.Status;
}
```

Listing 135: Methode zum Fortfahren eines Dienstes (Forts.)

Nun fehlt nur noch eine Methode zum Stoppen von Diensten:

```
public static ServiceControllerStatus StopService(string serviceName,
    string machineName, int timeout)
{
    // Dienst ermitteln
    if (machineName == null) machineName = ".";
    ServiceController service = new ServiceController(serviceName,
        machineName);

    // Überprüfen, ob der Dienst gestoppt werden kann
    if (service.CanStop == false)
        throw new Exception("Der Dienst '" + serviceName +
            "' kann nicht gestoppt werden");

    // Dienst stoppen, falls dieser nicht gestoppt ist oder gerade stoppt
    if (service.Status != ServiceControllerStatus.Stopped &&
        service.Status != ServiceControllerStatus.StopPending)
        service.Stop();

    // Warten, bis der Dienst den Status Stopped besitzt
    service.WaitForStatus(ServiceControllerStatus.Stopped,
        new TimeSpan(0, 0, timeout));

    // Status des Dienstes zurückliefern
    return service.Status;
}
```

Listing 136: Methode zum Stoppen eines Dienstes

Alle Methoden überprüfen zwar über den Status und die Eigenschaften `CanPauseAndContinue` bzw. `CanStop`, ob der gewünschte Statuswechsel grundsätzlich möglich bzw. nötig ist, fragen aber beim Aufruf der Methode, die den Statuswechsel bewirkt, nicht mehr weiter nach, ob dieser möglich ist. Ausnahmen, die im Fehlerfall generiert werden, werden auf diese Weise einfach an den Aufrufer weitergegeben. Die `Stop`-Methode führt z.B. nur dann zum Erfolg, wenn der Dienst gerade ausgeführt wird (also den Status `Running` besitzt). Da die im Fehlerfall erzeugten Ausnahmen recht aussagekräftig sind, wollte ich den Programmcode nicht noch über weitere Überprüfungen bzw. Ausnahmebehandlungen unübersichtlicher machen.

Das folgende Listing zeigt eine beispielhafte Anwendung der `StartService`-Methode. Gestartet wird der Dienst `NetDDE` auf dem lokalen Computer ohne Argumente. Als Timeout wird 2000 ms übergeben:

```

ServiceControllerStatus serviceStatus;
try
{
    serviceStatus = StartService("NetDDE", null, null, 2000);
    if (serviceStatus != ServiceControllerStatus.Running)
    {
        Console.WriteLine("Der Dienst konnte nicht innerhalb " +
            "des Timeout gestartet werden");
        return;
    }
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Starten des Dienstes: {0}", ex.Message);
    return;
}

```

Listing 137: Beispiel-Anwendung der StartService-Methode

72 Ermitteln, ob eine Anwendung ausgeführt wird

Wenn Sie in einem Ihrer Programme überprüfen müssen, ob eine andere Anwendung bereits ausgeführt wird (um diese im negativen Fall gegebenenfalls zu starten, wie ich es im Rezept 73 beschreibe), können Sie dazu prinzipiell die `Process`-Klasse auf eine ähnliche Weise verwenden, wie ich es bereits im Rezept 66 beschrieben habe. Dazu erzeugen Sie über die `GetProcessesByName`-Methode ein Array der Prozesse, die den Namen der Anwendung tragen. Enthält das alle Elemente, können Sie davon ausgehen, dass die Anwendung bereits ausgeführt wird.

Problematisch ist dabei allerdings, dass die Prozessnamen verschiedener Anwendungen identisch sein können. Deshalb können Sie nicht mit Sicherheit sagen, dass genau die Anwendung, die Sie meinen, bereits ausgeführt wird.

Zur Lösung dieses Problems können Sie den Titel des Hauptfensters der Anwendung überprüfen, wie ich es in dem oben genannten Rezept programmiert habe. Diese Titel werden bei verschiedenen Anwendungen dynamisch zusammengesetzt (bei vielen Anwendungen wird z.B. der Name einer geladenen Datei mit im Titel dargestellt). Deshalb sollten Sie lediglich überprüfen, ob ein für die Anwendung typischer Begriff im Titel vorkommt.

Die Methode `ApplicationRunning` in Listing 138 nimmt Ihnen diese Arbeit ab. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Diagnostics` importieren.

```

public static bool ApplicationRunning(string name, string title)
{
    // Ermitteln der Prozesse, die den übergebenen Namen tragen
    Process[] processes = Process.GetProcessesByName(name);

```

Listing 138: Methode zur Überprüfung, ob eine bestimmte Anwendung ausgeführt wird

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// Überprüfen, ob einer der eventuell gefundenen Prozesse im Titel
// des Hauptfensters den übergebenen Titel-String besitzt
foreach (Process p in processes)
{
    if (p.MainWindowTitle.ToLower().IndexOf(title.ToLower()) > 0)
        return true;
}

// Wenn die Methode hier ankommt, wurde kein Prozess gefunden
return false;
}
```

Listing 138: Methode zur Überprüfung, ob eine bestimmte Anwendung ausgeführt wird (Forts.)

Im folgenden Listing finden Sie eine beispielhafte Anwendung dieser Methode. Überprüft wird, ob Microsoft Outlook bereits ausgeführt wird.

```
// Überprüfen, ob Outlook läuft
if (ApplicationRunning("Outlook", "Microsoft Outlook"))
    Console.WriteLine("Outlook läuft");
else
    Console.WriteLine("Outlook läuft nicht");
```

Listing 139: Beispielhafte Anwendung der Methode zur Überprüfung, ob eine Anwendung bereits ausgeführt wird

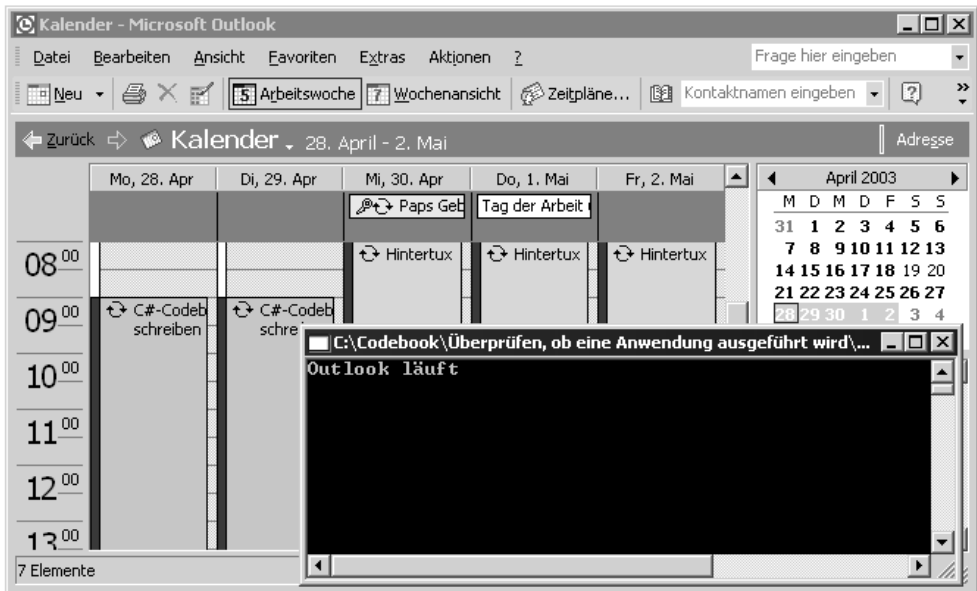


Abbildung 44: Das Beispielprogramm zur Überprüfung, ob Outlook ausgeführt wird

73 Anwendungen starten

Anwendungen können Sie über die `Start`-Methode der `Process`-Klasse aus dem Namensraum `System.Diagnostics` starten. Dazu können Sie am ersten Argument den Dateinamen und im optionalen zweiten Argument Befehlszeilenargumente für den Start übergeben. Flexibler ist allerdings die Übergabe eines `ProcessStartInfo`-Objekts, das Sie mit genaueren Angaben für den Start initialisieren können. So können Sie z.B. über die Eigenschaft `WindowStyle` festlegen, dass das Anwendungsfenster maximiert geöffnet wird.

Das Programm in Listing 140 startet den Windows-Editor maximiert und übergibt als Befehlszeilenargument die Datei `eula.txt`, die im Windows-Systemverzeichnis gespeichert ist. Ausnahmen, die beim Start der Anwendung erzeugt werden können (z.B. weil die Anwendung nicht vorhanden ist), werden abgefangen. Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.IO` und `System.Diagnostics` einbinden.

```
// Dateinamen zusammensetzen
string exeFileName = Path.Combine(Environment.SystemDirectory, "notepad.exe");
string arguments = Path.Combine(Environment.SystemDirectory, "eula.txt");

// Anwendungs-Prozess mit einem maximierten Fenster starten
ProcessStartInfo psi = new ProcessStartInfo(exeFileName, arguments);
psi.WindowStyle = ProcessWindowStyle.Maximized;

try
{
    Process.Start(psi);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Starten der Anwendung: " +
        ex.Message, Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Listing 140: Starten einer Anwendung

Ist das zu startende Programm in einem Ordner gespeichert, der in der Umgebungsvariable `Path` eingetragen ist, muss der Pfad zum Ordner nicht angegeben werden. In diesem Fall können Sie aber nicht sicher sein, dass das gewünschte Programm gestartet wird. Schließlich kann ein Programm mit demselben Dateinamen auch in einem anderen Ordner gespeichert sein.

74 Dateien mit der assoziierten Anwendung öffnen

In Windows sind viele Dateitypen mit einer Anwendung assoziiert. Wenn Sie eine Datei mit der verknüpften Anwendung öffnen wollen, können Sie dazu die `Start`-Methode der Klasse `System.Diagnostics.Process` verwenden. Am ersten Argument übergeben Sie wie im vorhergehenden Rezept ein `ProcessStartInfo`-Objekt mit Informationen zum Start. Neben dem

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Dateinamen, den Sie bei der Erzeugung dieses Objekts im Konstruktor übergeben können, legen Sie dann über die Eigenschaft `UseShellExecute` fest, dass die `Start`-Methode beim Starten des Prozesses die Shell verwendet (so, als würden Sie die Datei im Explorer öffnen). Über die Eigenschaft `WindowState` können Sie festlegen, wie das Fenster geöffnet werden soll (normal, minimiert, maximiert).

Die Methode `OpenFileWithAssociatedApp` in Listing 141 setzt dies um. Der am ersten Argument übergebene Dateiname wird dem Konstruktor des erzeugten `ProcessStartInfo`-Objekts übergeben. Wenn am zweiten Argument über `true` festgelegt wird, dass das Anwendungsfenster maximiert geöffnet werden soll, setzt `OpenFileWithAssociatedApp` die Eigenschaft `WindowState` auf `ProcessWindowState.Maximized`. Ausnahmen werden nicht abgefangen, sodass diese an den Aufrufer weitergegeben werden. Die Methode erfordert das Einbinden der Namensräume `System` und `System.Diagnostics`.

```
public static void OpenFileWithAssociatedApp(string fileName, bool maximized)
{
    // ProcessStartInfo-Instanz erzeugen und initialisieren
    ProcessStartInfo psi = new ProcessStartInfo(fileName);
    if (maximized)
        psi.WindowStyle = ProcessWindowState.Maximized;
    psi.UseShellExecute = true;

    // Prozess starten
    Process.Start(psi);
}
```

Listing 141: Methode zum Öffnen einer Datei mit der assoziierten Anwendung

75 Starten einer anderen Anwendung und warten, bis diese beendet ist

In einigen Fällen muss ein Programm eine andere Anwendung starten und mit der weiteren Ausführung warten, bis die andere Anwendung beendet wurde. Die Lösung dieses Problems zeigt die Methode `StartApplication`, die eine Anwendung über die `Start`-Methode der `Process`-Klasse (siehe Rezept 73) startet und abhängig vom Argument `waitForExit` so lange wartet, bis diese beendet ist. `StartApplication` ruft dazu die Methode `WaitForExit` auf, die so auf das Beenden des gestarteten Prozesses wartet, dass Windows dabei nicht ausgebremst wird. Zum Kompilieren müssen Sie die Namensräume `System` und `System.Diagnostics` importieren.

```
public static void StartApplication(string exeFileName, string arguments,
    bool maximized, bool waitForExit)
{
    // ProcessStartInfo-Instanz erzeugen und initialisieren
    ProcessStartInfo psi = new ProcessStartInfo(exeFileName);
    if (arguments != "")
```

Listing 142: Methode zum Starten einer Anwendung mit optionalem Warten auf deren Beendigung


```

        psi.Arguments = arguments;
    if (maximized)
        psi.WindowStyle = ProcessWindowStyle.Maximized;
    psi.UseShellExecute = true;

    // Prozess starten
    Process process = Process.Start(psi);

    // Auf die Beendigung des Prozesses warten, sofern dies gewünscht ist
    if (waitForExit)
        process.WaitForExit();
}

```

Listing 142: Methode zum Starten einer Anwendung mit optionalem Warten auf deren Beendigung (Forts.)

Das folgende Listing zeigt die Anwendung dieser Methode:

```

// Dateinamen zusammensetzen
string exeFileName = Path.Combine(Environment.SystemDirectory, "notepad.exe");
string arguments = Path.Combine(Environment.SystemDirectory, "eula.txt");

// Anwendung starten und warten, bis diese beendet ist
Console.WriteLine("Starte '{0}' und warte auf die Beendigung ... ",
    exeFileName);
StartApplication(exeFileName, arguments, false, true);
Console.WriteLine("Anwendung wurde beendet");

```

Listing 143: Anwendung der Methode zum Starten einer Anwendung mit Warten auf deren Beendigung

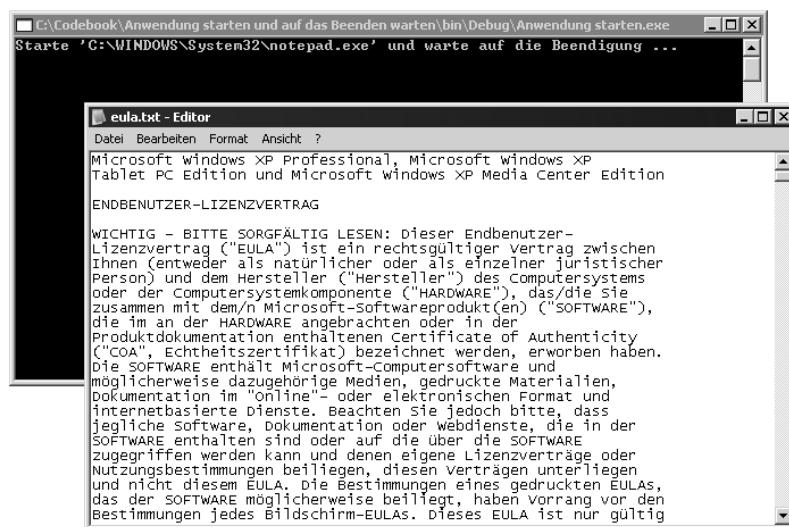


Abbildung 45: Das Beispielprogramm hat den Editor gestartet und wartet auf dessen Beendigung

76 Andere Anwendungen beenden

Wenn Sie eine andere Anwendung beenden wollen, können Sie dies über die `Process`-Klasse erreichen. Um die laufenden Prozesse einer Anwendung zu lokalisieren, rufen Sie die Methode `GetProcessesByName` auf. Diese Methode gibt ein Array von Referenzen auf `Process`-Objekte zurück. Über deren Methode `CloseMainWindow` können Sie dem Hauptfenster die Windows-Nachricht zum Schließen (und damit zum Beenden der Anwendung) senden. Das Schließen funktioniert natürlich (wie üblich) nur dann, wenn kein modaler Dialog aktiv ist und wenn die Anwendung nicht nachfragt, ob geänderte Daten gespeichert werden sollen (o.Ä.). Statt `CloseMainWindow` können Sie auch `Kill` aufrufen, um den Prozess abzuschießen (ähnlich wie mit dem Taskmanager).

Die Methode `CloseApplication` in Listing 144 setzt diese Technik ein. Um das Schließen ein wenig sicherer zu machen, erwartet diese Methode neben dem Prozessnamen den Titel des Hauptfensters (an dem Sie allerdings auch null übergeben können, wenn Sie alle Prozesse mit dem übergebenen Namen beenden wollen). Das dritte Argument (`waitForExitTimeout`) bestimmt einen Timeout zum Warten auf das Beenden der Anwendung in Millisekunden.

`CloseApplication` ermittelt zunächst alle Prozesse mit dem übergebenen Namen und geht diese durch. Für jeden Prozess überprüft die Methode, ob dessen Hauptfenster den eventuell übergebenen Titel trägt, und ruft dann die `CloseMainWindow`-Methode auf. Wenn am Argument `waitForExitTimeout` ein Wert größer 0 übergeben wurde, ruft `CloseApplication` danach die `WaitForExit`-Methode des `Process`-Objekts unter Übergabe des Timeout auf um auf das Beenden zu warten. Dieser Timeout ist sinnvoll für Anwendungen, die beim Beenden eventuell nachfragen, ob geänderte Daten gespeichert werden sollen.

Konnte der Prozess beendet werden, wird die lokale Variable `count` inkrementiert. Beim Beenden ohne Timeout wartet `CloseApplication` zur Sicherheit 100 Millisekunden um dem Prozess Zeit zu geben, sich zu beenden, und fragt dann die Eigenschaft `HasExited` ab. Ist diese Eigenschaft `true`, wird `count` hochgezählt. Der Wert von `count` wird schließlich zurückgegeben.

Zum Kompilieren der Methode `CloseApplication` müssen Sie die Namensräume `System`, `System.Threading` und `System.Diagnostics` importieren.

```
public static int CloseApplication(string processName,
    string mainWindowTitle, int waitForExitTimeout)
{
    int count = 0;

    // Liste der Prozesse holen, die den übergebenen Prozessnamen tragen
    Process[] processes = Process.GetProcessesByName(processName);
```

Listing 144: Methode zum Beenden einer Anwendung

```

foreach (Process process in processes)
{
    // Überprüfen des Hauptfenster-Titels, sofern dieser übergeben wurde
    if ((mainWindowTitle != null &&
        process.MainWindowTitle == mainWindowTitle) ||
        (mainWindowTitle == null))
    {
        // Eine Beenden-Nachricht an das Hauptfenster senden
        process.CloseMainWindow();

        // Auf das Beenden des Prozesses warten, wenn ein Timeout
        // übergeben wurde
        if (waitForExitTimeout > 0)
        {
            if (process.WaitForExit(waitForExitTimeout))
                count++;
        }
        else
        {
            // Etwas warten, um dem Prozess Zeit zu lassen sich zu beenden
            Thread.Sleep(100);
            if (process.HasExited)
                count++;
        }
    }
}

// Anzahl der beendeten Prozesse zurückgeben
return count;
}

```

Listing 144: Methode zum Beenden einer Anwendung (Forts.)

77 Java-Anwendungen starten

Auch im .NET-Zeitalter (oder vielleicht gerade deswegen) besitzt Java Bedeutung. Und möglicherweise wird es sogar vorkommen, dass eines Ihrer .NET-Programme eine Anwendung starten muss, die mit dem Klassenfeind programmiert wurde. Auch wenn es vielleicht wehtut – so geht’s:

Java-Anwendungen liegen meist in kompilierten *.class*-Dateien vor, können aber auch in Java-Archiven³ (*.jar*-Dateien) gespeichert sein. Beide Arten von Dateien können über den Java-Interpreter *java* ausgeführt werden, der unter Windows in der Datei *java.exe* gespeichert ist. Dieser Interpreter erzeugt automatisch ein Konsolenfenster. Java-Anwendungen, die mit Fenstern arbeiten, können Sie auch über den Interpreter *javaw* starten, der kein Konsolen-

-
3. Normale Java-Anwendungen bestehen üblicherweise aus mehreren Klassendateien. Zum Start einer solchen Anwendung müssen alle Klassendateien vorhanden sein. Ein Java-Archiv ist eine Datei, die mehrere Klassendateien (und eventuell auch andere Dateien) zu einer Datei zusammenfasst und damit die Weitergabe und das Handling von Java-Anwendungen erleichtert.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

fenster erzeugt. Handelt es sich um ein Java-Archiv, müssen Sie beim Starten das Argument `-jar` mit angeben.

Voraussetzung für die Ausführung einer Java-Anwendung ist, dass die Java-Laufzeitumgebung (JRE) auf dem Rechner installiert ist. Falls dies nicht der Fall ist, können Sie diese unter der Adresse java.sun.com/j2se/downloads.html herunterladen. Der Java-Interpreter ist Teil dieser Laufzeitumgebung.

Da normalerweise nicht mit Sicherheit bekannt ist, in welchem Ordner der Java-Interpreter gespeichert ist, wird der Pfad zu diesem häufig in die Umgebungsvariable `Path` aufgenommen. Sie können das Programm `java.exe` bzw. `javaw.exe` also normalerweise ohne Angabe des Pfads aufrufen. Zum Aufrufen verwenden Sie dann wie im Rezept 73 die `Start`-Methode der `Process`-Klasse.

Java-Interpreter arbeiten mit einem so genannten Klassenpfad (Classpath). In diesem Pfad werden Klassen gesucht, die vom Programm verwendet, aber nicht im Programmordner oder in einem der Standard-Java-Ordner gefunden werden. Häufig ist der Klassenpfad in der Umgebungsvariable `Classpath` eingestellt. Beim Aufruf des Java-Interpreters kann der Klassenpfad aber auch über das Befehlszeilenargument `-classpath` mit angegeben werden.

Die folgende Methode `StartJavaApplication` setzt dieses Wissen um und startet eine Java-Anwendung, deren Dateiname übergeben wird. Am zweiten Argument kann ein Klassenpfad übergeben werden.

`StartJavaApplication` überprüft zunächst, ob das Java-Programm überhaupt existiert, um undurchsichtige und beim Start über die Windows-Oberfläche bzw. von Fensteranwendungen an der Konsole nur kurz erscheinende Fehlermeldungen des Java-Interpreters zu vermeiden. Dann zerlegt die Methode den Dateinamen der Java-Datei in den Pfad und den eigentlichen Namen. Danach wird überprüft, ob es sich um eine `.jar`- oder eine `.class`-Datei handelt. Bei einer `.jar`-Datei wird das Befehlszeilenargument `-jar` in der Variablen `arguments` abgelegt, die später zum Aufruf des Java-Interpreters verwendet wird. Bei einer `.class`-Datei wird der Dateiname um die Endung gekürzt, da diese bei solchen Dateien nicht mit übergeben werden darf. Für den Fall einer unbekannten Dateiendung erzeugt `StartJavaApplication` eine Ausnahme (wieder um die undurchsichtigen und an der Konsole ausgegebenen Fehlermeldungen des Java-Interpreters zu vermeiden). Die Startargumente werden dann noch um den u.U. übergebenen Klassenpfad und den Dateinamen des Java-Programms erweitert.

Java-Interpreter sollten (bzw. müssen) aus dem Ordner heraus gestartet werden, in dem das Java-Programm gespeichert ist. In allen anderen Fällen resultiert der Java-Fehler, dass eine oder mehrere Klassen nicht gefunden wurden (in Java ausgedrückt ist das eine Ausnahme vom Typ `ClassDefNotFoundError`), wenn der Pfad zum Ordner der Anwendung nicht im Klassenpfad vorkommt. `StartJavaApplication` wechselt deswegen über `Environment.CurrentDirectory` in den Ordner des Java-Programms.

Danach wird der Java-Interpreter unter der Übergabe der ermittelten Befehlszeilenargumente gestartet. Dabei wird am Argument `isWindowApp` erkannt, ob `java` oder (für Anwendungen mit Fenstern) `javaw` gestartet werden soll. Tritt beim Start eine Ausnahme vom Typ `Win32Exception` auf, wird diese mit einer genaueren Fehlermeldung (als `IOException`) weiter-

gereicht. Damit ist es später wesentlich einfacher, zu erkennen, wo die Ursache für ein Fehlschlagen der Methode zu suchen ist.

StartJavaApplication erfordert das Einbinden der Namensräume System, System.Diagnostics, System.IO und System.ComponentModel.

```
public static void StartJavaApplication(string fileName, string classPath,
    bool isWindowApp)
{
    // Überprüfen, ob die Datei existiert
    if (File.Exists(fileName) == false)
    {
        throw new IOException("Das Java-Programm '" + fileName +
            "' existiert nicht");
    }

    // Variable für die Start-Argumente
    string arguments = null;

    // Pfad und Dateiname trennen
    FileInfo fi = new FileInfo(fileName);
    string path = fi.DirectoryName;
    fileName = fi.Name;

    // Argumente zusammensetzen und den Dateinamen bei einer .class-Datei
    // ohne Endung auslesen
    if (fi.Extension == ".jar")
        arguments += " -jar";
    else if (fi.Extension == ".class")
        fileName = fileName.Substring(0, fileName.Length - 6);
    else
    {
        throw new IOException("Unbekannte Dateiendung '" +
            fi.Extension + "'");
    }

    // Den Argumenten den Klassenpfad hinzufügen
    if (classPath != null)
        arguments += " -classpath " + classPath;

    // Den Dateinamen zu den Argumenten hinzufügen
    arguments += " " + fileName;

    // In den Ordner wechseln, in dem die Datei gespeichert ist
    Environment.CurrentDirectory = path;

    // Den Java-Interpreter starten
    string javaInterpreter = null;
    if (isWindowApp)
        javaInterpreter = "javaw.exe";
```

Listing 145: Methode zum Starten einer Java-Anwendung

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateien

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

else
    javaInterpreter = "java.exe";

try
{
    Process.Start(javaInterpreter, arguments);
}
catch (Win32Exception ex)
{
    throw new IOException("Fehler beim Starten des " +
        "Java-Interpreters " + javaInterpreter + ": " +
        ex.Message);
}
}

```

Listing 145: Methode zum Starten einer Java-Anwendung (Forts.)

Das folgende Listing zeigt eine Beispielanwendung der Methode `StartJavaApplication`. Gestartet wird einmal das Programm *hello.class*, das in dem dem Anwendungsordner untergeordneten Ordner *Java\class* erwartet wird, und zum anderen das in einem Java-Archiv vorliegende Programm *hello.jar* (im Unterordner *Java\jar*):

```

// Java-Programm in .class-Dateien ohne explizite Angabe des
// ClassPath starten
string javaFileName = Path.Combine(Application.StartupPath,
    @"Java\class\hello.class");
try
{
    Console.WriteLine("Starte {0} ...", javaFileName);
    StartJavaApplication(javaFileName, null, true);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

// Java-Programm in .jar-Datei ohne explizite Angabe des ClassPath starten
javaFileName = Path.Combine(Application.StartupPath,
    @"Java\jar\hello.jar");
try
{
    Console.WriteLine("Starte {0} ...", javaFileName);
    StartJavaApplication(javaFileName, null, true);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 146: Starten einer Java-Anwendung in einer class- und einer jar-Version

```
Console.WriteLine("Beenden mit Return");  
Console.ReadLine();
```

Listing 146: Starten einer Java-Anwendung in einer class- und einer jar-Version (Forts.)

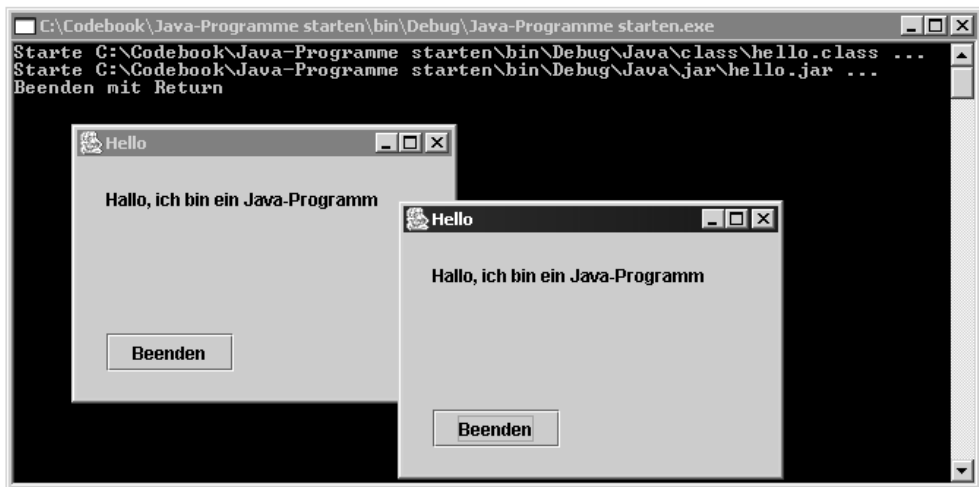


Abbildung 46: Das Beispielprogramm hat zwei (gleichartige) Java-Anwendungen gestartet

Tipp

Für den Fall, dass der Java-Interpreter nicht gefunden wird, können Sie dem Anwender über das Rezept 113 die Möglichkeit geben, diesen automatisch zu suchen. Idealerweise speichern Sie den gefundenen Pfad dann in der Konfigurationsdatei der Anwendung (Rezept 63 und 64), um diesen beim nächsten Start dort wieder auszulesen (Rezept 61 und 64).

78 Dateiname einer Windows-Anwendung ermitteln

Den vollen Dateinamen einer Windows-Anwendung (inklusive Pfad) erhalten Sie über die Eigenschaft `ExecutablePath` der `Application`:

```
string filename = System.Windows.Forms.Application.ExecutablePath;
```

Um die `Application`-Klasse verfügbar zu haben, benötigt die Anwendung einen Verweis auf die Assembly `System.Windows.Forms.dll`, was ohne Probleme auch in einer Konsolenanwendung oder Klassenbibliothek möglich ist.

Ohne diesen Verweis können Sie aber auch stattdessen die Eigenschaft `Location` eines `Assembly`-Objekts abfragen, das die »Eintritts-Assembly« (das ist die Assembly, über die das Programm gestartet wurde) repräsentiert. Dieses Objekt erhalten Sie über die Methode `GetEntryAssembly` der `Assembly`-Klasse, die Bestandteil des `System.Reflection`-Namensraums ist:

```
string filename = System.Reflection.Assembly.GetEntryAssembly().Location;
```

Hinweis

Als *Reflektion* (Reflection) werden Features bezeichnet, über die Sie in der Laufzeit eines Programms Informationen über dessen Assemblies, die darin enthaltenen Klassen und deren Schnittstellen, Methoden, Eigenschaften und Ereignisse etc. auslesen können. Ein weiterer Teil von Reflektion ist das dynamische Erzeugen von Klassen und Assemblies, also das Erzeugen von Programmcode in der Laufzeit eines Programms.

Häufig wird auch vorgeschlagen, die »ausführende Assembly« zu verwenden, die Sie über die Methode `GetExecutingAssembly` erreichen:

```
string filename = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

Dabei müssen Sie allerdings aufpassen: Wird diese Methode in einer Klassenbibliothek aufgerufen, liefert sie ein `Assembly`-Objekt zurück, das die Assembly der *Klassenbibliothek* repräsentiert. Der ausgelesene Dateiname ist also in diesem Fall nicht der der Anwendung, sondern der der Klassenbibliothek.

In allen Fällen erhalten Sie den Dateinamen inklusive Pfad. Wenn Sie nur den Namen der Datei auslesen wollen, können Sie diesen über ein `FileInfo`-Objekt ermitteln. Ein solches Objekt dient dem Auslesen von Informationen zu einer Datei und liefert u.a. über Eigenschaft `Name` den Dateinamen. Die Methode `GetApplicationFilename` in Listing 147 liest den Dateinamen der Anwendung aus und liefert diesen zurück. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

```
public static string GetApplicationFilename()
{
    // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-Assembly
    // speichert
    FileInfo fi = new FileInfo(Assembly.GetEntryAssembly().Location);

    // Dateiname auslesen und zurückgeben
    return fi.Name;
}
```

Listing 147: Methode zum Auslesen des Dateinamens einer Windows- oder Konsolenanwendung

79 Dateiname einer Klassenbibliothek ermitteln

Innerhalb einer Klassenbibliothek können Sie deren Dateinamen ermitteln, indem Sie eine Referenz auf ein `Assembly`-Objekt für die »ausführende Assembly« auslesen und deren `Location`-Eigenschaft abfragen:

```
string filename = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

Wie bereits im vorigen Rezept erhalten Sie den Dateinamen inklusive Pfad. Die Methode `GetLibraryFilename` in Listing 148 liest nur den Dateinamen aus. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

```
public static string GetLibraryFilename()
{
    // FileInfo-Objekt für die Datei erzeugen, die die ausführende
    // Assembly speichert
    FileInfo fi = new FileInfo(Assembly.GetExecutingAssembly().Location);

    // Den Dateinamen auslesen und zurückgeben
    return fi.Name;
}
```

Listing 148: Methode zum Auslesen des Dateinamens einer Klassenbibliothek

Die in die Klassenbibliothek *Addison_Wesley.Codebook.Filesystem.DemoLib.dll* kompilierte Klasse *Demo* in Listing 149 ruft diese Methode in ihrer *GetFilename*-Methode auf:

```
using System;

namespace Addison_Wesley.Codebook.Filesystem.DemoLib
{
    public class Demo
    {
        /* Methode zum Auslesen des Dateinamens der Klassenbibliothek.
         * Wird zu Demozwecken von der Anwendung aus aufgerufen */
        public string GetFilename()
        {
            return GetLibraryFilename();
        }
    }
}
```

Listing 149: Ermittlung des Dateinamens einer Klassenbibliothek in einer solchen

Die folgende Konsolenanwendung, die die Klassenbibliothek referenziert, erzeugt eine Instanz der Klasse *Demo* und ermittelt über deren Methode *GetFilename* den Dateinamen der Klassenbibliothek:

```
using System;
using Addison_Wesley.Codebook.Filesystem.DemoLib;

namespace Bibliotheks_Dateiname
{
    class Start
    {
        [STAThread]
        static void Main(string[] args)
        {
            // Instanz der Demo-Klasse in der Klassenbibliothek erzeugen
            Demo d = new Demo();

            // Dateiname der Klassenbibliothek ermitteln
            Console.WriteLine("Dateiname der Klassenbibliothek:\r\n{0}",
                d.GetFilename());
        }
    }
}
```

Listing 150: Konsolenanwendung, die die Methode *GetFilename* der Bibliotheks-Klasse *Demo* aufruft

Abbildung 47 zeigt das ausgeführte Programm.



Abbildung 47: Das Demo-Programm zur Ermittlung des Dateinamens einer Klassenbibliothek

80 Ordner einer Windowsanwendung ermitteln

In einer Anwendung mit Referenz auf die Assembly *System.Windows.Forms.dll* können Sie den Pfad der Anwendung über die Eigenschaft *StartupPath* der *Application*-Klasse ermitteln:

```
string applicationPath = System.Windows.Forms.Application.StartupPath;
```

Ohne Referenz auf die *System.Windows.Forms.dll* können Sie stattdessen auch den vollen Dateinamen der »Eintritts-Assembly« ermitteln und über ein *FileInfo*-Objekt den Pfad auslesen, wie es die Methode *GetApplicationFolderName* in Listing 151 zeigt.

```
public static string GetApplicationFolderName()
{
    // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-Assembly
    // speichert
    FileInfo fi = new FileInfo(Assembly.GetEntryAssembly().Location);

    // Den Pfad des Ordners der Datei zurückgeben
    return fi.DirectoryName;
}
```

Listing 151: Methode zum Auslesen des Ordernamens einer Windows- oder Konsolenanwendung

Um diese Methode kompilieren zu können, müssen Sie die Namensräume *System*, *System.IO* und *System.Reflection* importieren.

Beide Varianten funktionieren auch ohne Probleme in einer Klassenbibliothek. Sie erhalten dann allerdings den Ordernamen der Anwendung zurück, nicht den der Klassenbibliothek.

81 Ordner einer Klassenbibliothek ermitteln

Zur Ermittlung des Ordners einer Klassenbibliothek innerhalb deren Programmcodes können Sie wie beim Auslesen des Dateinamens (Rezept 79) ein *Assembly*-Objekt für die ausführende Assembly erzeugen, dessen *Location*-Eigenschaft auslesen und den Pfad über die *DirectoryName*-Eigenschaft eines *FileInfo*-Objekts extrahieren. Die Methode *GetLibraryFolderName* in Listing 152 nimmt Ihnen diese Arbeit ab. Zum Kompilieren dieser Methode müssen Sie die Namensräume *System*, *System.IO* und *System.Reflection* einbinden.

```
public static string GetLibraryFolderName()
{
    // FileInfo-Objekt für die Datei erzeugen, die die ausführende
    // Assembly speichert
    FileInfo fi = new FileInfo(Assembly.GetExecutingAssembly().Location);

    // Den Pfad des Verzeichnisses der Datei zurückgeben
    return fi.DirectoryName;
}
```

Listing 152: Methode zum Auslesen des Ordernamens einer Klassenbibliothek

82 Dateiname für eine temporäre Datei erzeugen

Wenn Sie eine temporäre Datei erzeugen wollen bzw. müssen, sollten Sie für diese einen eindeutigen Dateinamen ermitteln, dessen Pfad dem Systempfad für temporäre Dateien entspricht. Dazu können Sie einfach die Methode `GetTempFileName` der `Path`-Klasse verwenden:

```
string tempFileName = System.IO.Path.GetTempFileName();
```

Sie können absolut sicher sein, dass der ermittelte Dateiname eindeutig ist, d.h. dass noch keine Datei gleichen Namens existiert: `GetTempFileName` ermittelt einen Dateinamen in der Form `tmpnn.tmp`, wobei `nn` für eine hexadezimale Zahl steht, die bei einem im System verwalteten letzten Startwert beginnt. Zur Überprüfung, ob im Temp-Ordner bereits eine Datei gleichen Namens existiert, versucht `GetTempFileName` eine 0-Byte-Datei mit diesem Namen anzulegen. Tritt dabei kein (interner) Fehler auf, ist der Dateiname eindeutig.

Daraus folgt allerdings auch, dass die von `GetTempFileName` erzeugten 0-Byte-Dateien nicht automatisch gelöscht werden und Sie beim Erstellen einer »echten« Datei diese überschreiben müssen.

83 Ordner für temporäre Dateien ermitteln

Den Systemordner für temporäre Dateien erhalten Sie über die Methode `GetTempPath` der `Path`:

```
string tempFolder = System.IO.Path.GetTempPath();
```

84 Windows-(Spezial-)Ordner auslesen

Den Windows-Systemordner erhalten Sie recht einfach über die Eigenschaft `SystemDirectory` der Klasse `System.Environment`:

```
string systemDirectory = Environment.SystemDirectory;
```

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Den Pfad spezieller Ordner wie z.B. den des Programmordners können Sie über die Methode `GetFolderPath` ermitteln, der Sie eine Konstante der Aufzählung `SpecialFolder` übergeben. Den Ordner für Programme können Sie z.B. folgendermaßen auslesen:

```
string programDirectory = Environment.GetFolderPath(
    Environment.SpecialFolder.ProgramFiles);
```

Die Vielzahl der möglichen Pfade wird in der Hilfe zur `SpecialFolder`-Aufzählung ausführlich erläutert.

Leider können Sie über die `Environment`-Klasse nicht ohne weiteres das Windows-Verzeichnis ermitteln. Dazu können Sie die Umgebungsvariable `windir` abfragen:

```
string windowsDirectory = Environment.GetEnvironmentVariable("windir");
```

Da Umgebungsvariablen auch gelöscht werden können und in neueren Windows-Versionen nicht mehr unbedingt vorhanden sein müssen, sollten Sie stattdessen lieber die API-Funktion `GetWindowsDirectory` verwenden.

Listing 153 kapselt den Aufruf dieser Funktion in die Methode `GetWindowsDirectoryName`. Weitere Methoden geben den Namen der (in meinen Augen 😊) wichtigsten Windows-Systemordner zurück. Die Methoden erfordern den Import der Namensräume `System`, `System.Runtime`, `InteropServices` und `System.Text`.

```
/* Deklaration der API-Funktion GetWindowsDirectory */
[DllImport("kernel32.dll", SetLastError=true)]
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,
    uint uSize);

/* Methode zur Ermittlung des Windows-Ordners */
public static string GetWindowsDirectoryName()
{
    const int MAX_PATH = 260;
    StringBuilder buffer = new StringBuilder(MAX_PATH + 1);
    if (GetWindowsDirectory(buffer, MAX_PATH + 1) > 0)
        return buffer.ToString();
    else
        throw new Exception("Windows-Fehler " + Marshal.GetLastWin32Error() +
            " beim Ermitteln des Windows-Ordners");
}

/* Methode zur Ermittlung des Windows-System-Ordners */
public static string GetSystemDirectoryName()
{
    return Environment.SystemDirectory;
}
```

Listing 153: Methoden zur Ermittlung der Pfade zu den wichtigsten System-Ordern

```
/* Methode zur Ermittlung des Programmordners */  
public static string GetProgramDirectoryName()  
{  
    return Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles);  
}  
  
/* Methode zur Ermittlung des Desktop-Ordners */  
public static string GetDesktopDirectoryName()  
{  
    return Environment.GetFolderPath(  
        Environment.SpecialFolder.DesktopDirectory);  
}  
  
/* Methode zur Ermittlung des Favoriten-Ordners */  
public static string GetFavoritesDirectoryName()  
{  
    return Environment.GetFolderPath(Environment.SpecialFolder.Favorites);  
}  
  
/* Methode zur Ermittlung des Ordners für zuletzt geöffnete Dokumente */  
public static string GetRecentDirectoryName()  
{  
    return Environment.GetFolderPath(Environment.SpecialFolder.Recent);  
}
```

Listing 153: Methoden zur Ermittlung der Pfade zu den wichtigsten System-Ordnern (Forts.)

Abbildung 48 zeigt eine Konsolenanwendung, die über diese Methoden die Namen der wichtigen Windows-Systemordner ermittelt hat.

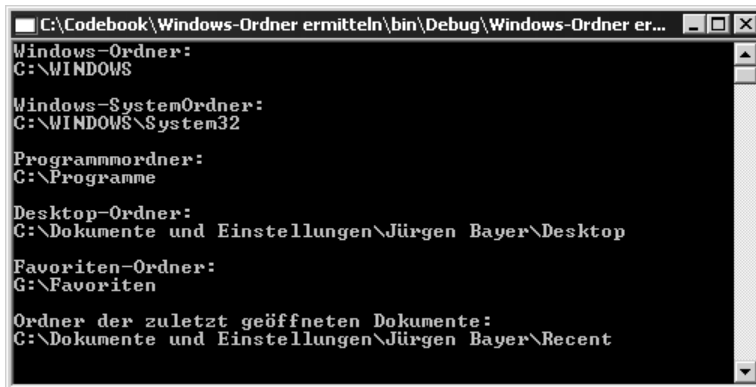


Abbildung 48: Beispielprogramm zur Ermittlung der Windows-Systemordner

85 Dateiname mit anderer Endung ermitteln

In vielen Fällen, z.B. beim Konvertieren einer Grafikdatei in einen anderen Grafiktyp, werden Sie einen Dateinamen mit einer anderen Dateiendung ermitteln wollen. Die Methode `ChangeExtension` in Listing 154 übernimmt diese Aufgabe unter Verwendung eines `FileInfo`-Objekts. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
public static string ChangeExtension(string filename, string newExtension)
{
    FileInfo fi = new FileInfo(filename);
    return fi.FullName.Replace(fi.Extension, newExtension);
}
```

Listing 154: Methode zur Ermittlung eines Dateinamens mit einer anderen Endung

86 Die logischen Laufwerke des Systems ermitteln

Sie können alle logischen Laufwerke eines Systems ermitteln, indem Sie die Methode `GetLogicalDrives` der `Environment`-Klasse (aus dem Namensraum `System`) aufrufen. Diese Methode gibt ein `String`-Array mit den Wurzel-Ordnernamen der ermittelten Laufwerke zurück:

```
// Namen der logischen Laufwerke des Systems ermitteln
string[] driveNames = Environment.GetLogicalDrives();

// Alle logischen Laufwerke durchgehen
for (int i = 0; i < driveNames.Length; i++)
    Console.WriteLine(driveNames[i]);
```

Zum Kompilieren dieses Beispiels müssen Sie den Namensraum `System` importieren.

`GetLogicalDrives` findet alle Laufwerke des Systems, neben den Festplattenlaufwerken sind das auch entfernbare (wie Disketten-Laufwerke) und CD-ROM-Laufwerke und solche, die mit Netzwerkordnern verknüpft sind. Abbildung 49 zeigt das Beispielprogramm auf meinem System. A: ist ein Disketten-, J: ein CD-ROM- und Z: ein Laufwerk, das mit einem Netzwerkordner verknüpft ist.

Leider können Sie auf diese Weise nicht herausfinden, um welchen Laufwerktyp es sich handelt, wie groß das Laufwerk ist und wie viel Speicherplatz noch zurzeit frei ist. Wenn Sie diese Information benötigen (z.B. um nur die Festplattenpartitionen zu ermitteln), müssen Sie eine andere Technik einsetzen. Lesen Sie dazu die nachfolgenden Rezepte.

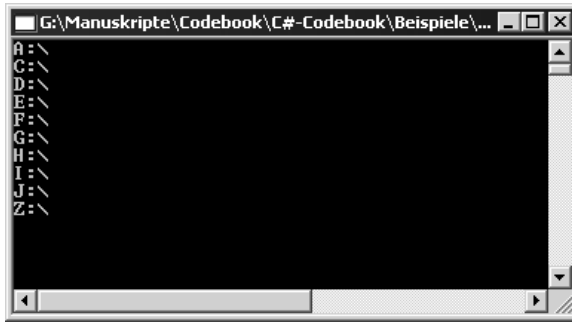


Abbildung 49: Ermittlung aller logischen Laufwerke meines Systems

87 Den Typ der Laufwerke eines Systems ermitteln

Zur Ermittlung der Typen aller Laufwerke eines Systems existieren scheinbar leider keine Klassen bzw. Methoden in der .NET-Bibliothek.

Um dieses Problem zu lösen können Sie mehrere Wege gehen. Sie können die Klasse `FileSystemObject` aus der normalerweise auf allen Windows-Systemen (über den Internet Explorer oder über Office) installierten COM-Komponente *Microsoft Scripting Runtime* verwenden, die das Durchgehen der einzelnen Laufwerke eines Systems über eine `Drives`-Auflistung sehr einfach macht. Alternativ können Sie WMI (Windows Management Instrumentation) einsetzen, das für Zwecke wie den unseren geschaffen wurde. Eine dritte (aber nicht unbedingt letzte ☺) Möglichkeit ist das Auslesen des Laufwerktyps über die API-Funktion `GetDriveType`. Im Folgenden beschreibe ich die Verwendung der `FileSystemObject`-Klasse und des Windows-API. Die WMI-Variante lasse ich außen vor, da ich WMI in der Einführung und in den Rezepten ab dem Rezept 150 recht ausführlich behandle. Mit dem dort erworbenen Wissen, ein wenig Gehirnschmalz und dem Lesen der Dokumentation können Sie dann auch recht einfach die Laufwerke eines Systems über WMI erforschen.

Laufwerkinformationen über die `FileSystemObject`-Klasse auslesen

Die Verwendung der `FileSystemObject`-Klasse bietet den Vorteil, dass Sie neben dem Typ eines Laufwerks auf eine einfache Weise auch weitere Informationen wie z.B. die Größe, den freien Speicherplatz und das Dateisystem auslesen können. Ein eventueller Nachteil ist, dass die *Scripting Runtime* -Komponente auf dem Computer installiert sein muss, auf dem das Programm ausgeführt wird. Eine Setup-Anwendung, die Sie mit dem Setup-Assistenten von Visual Studio .NET erzeugen, integriert die COM-Komponente aber automatisch.

Um die `FileSystemObject`-Klasse verfügbar zu haben, müssen Sie Ihrer Anwendung eine Referenz auf die COM-Komponente *Microsoft Scripting Runtime* hinzufügen. In Visual Studio .NET können Sie dies recht einfach über den Verweis-hinzufügen-Dialog erreichen (Abbildung 50).

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

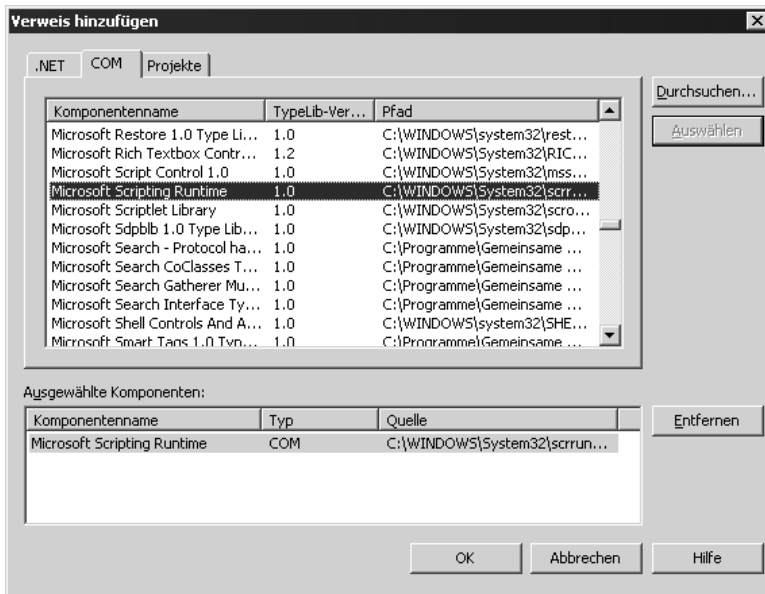


Abbildung 50: Hinzufügen einer Referenz auf die COM-Komponente Microsoft Scripting Runtime

Nun erzeugen Sie ein `FileSystemObject`-Objekt, dessen Klasse Sie im Namensraum `Scripting` finden. Über dessen `Drives`-Auflistung können Sie die einzelnen Laufwerke des Systems in Form von `Drive`-Objekten durchgehen. Diese Auflistung müssen Sie über eine `foreach`-Schleife durchlaufen, da der »Indizierer« der Auflistung keinen `int`-Index erlaubt (sondern lediglich einen Laufwerksbuchstaben). Über die Eigenschaft `DriveLetter` können Sie den Laufwerksbuchstaben (mit dem Doppelpunkt) auslesen. Die Eigenschaft `DriveType` verwaltet den Typ des Laufwerks als Wert der Aufzählung `DriveTypeConst`. Daneben können Sie noch weitere Eigenschaften wie z. B. den freien Speicherplatz ermitteln.

Um das Auslesen der Laufwerkinformationen möglichst einfach zu machen, habe ich eine `.NET`-Klasse `Drive` entwickelt, die ein Laufwerk repräsentiert und über die Methode `GetLocalDrives` alle lokalen Laufwerke in einem Array von `Drive`-Objekten zurückgibt. Eine Aufzählung `DriveType` verwaltet Konstanten für den Laufwerktyp. Zur Vereinfachung korrespondieren diese Konstanten mit den Werten der entsprechenden Aufzählung aus der `Scripting`-Bibliothek.

Beim Ermitteln der Laufwerkinformationen können die Laufwerkgröße und der freie Speicherplatz nur dann ausgelesen werden, wenn das Laufwerk bereit ist. Deshalb liest die `GetLocalDrives`-Methode die Eigenschaft `IsReady` aus und speichert im Fall eines nicht bereiten Laufwerks jeweils -1 in den entsprechenden Eigenschaften. Dummerweise besitzen die Größen-Eigenschaften den COM-Datentyp `Variant`, der ähnlich `object` alle Datentypen verwalten kann. Einen `Variant`-Typ können Sie nicht einfach über einen `Typecast` konvertieren, weil dieser prinzipiell auch für die Konvertierung ungültige Daten speichern kann. Dazu müssen Sie die Methoden der `Convert`-Klasse verwenden, wie ich es in der Methode `GetLocalDrives` umgesetzt habe.

Zum Kompilieren dieser Klasse müssen Sie wie bereits gezeigt die COM-Komponente *Microsoft Scripting Runtime* referenzieren. Daneben müssen Sie den Namensraum `System` importieren.

```

/* Klasse, die ein Laufwerk repräsentiert */
public class Drive
{
    /* Aufzählung für den Laufwerkstyp */
    public enum DriveType
    {
        CDRom = Scripting.DriveTypeConst.CDRom,
        Fixed = Scripting.DriveTypeConst.Fixed,
        RamDisk = Scripting.DriveTypeConst.RamDisk,
        Remote = Scripting.DriveTypeConst.Remote,
        Removable = Scripting.DriveTypeConst.Removable,
        UnknownType = Scripting.DriveTypeConst.UnknownType
    }

    /* Eigenschaften */
    public string DriveLetter;
    public DriveType Type;
    public long TotalSize;
    public long AvailableSpace;
    public long FreeSpace;
    public bool IsReady;

    /* Statische Methode zum Auslesen aller Laufwerke des Systems */
    public static Drive[] GetLocalDrives()
    {
        // Instanz der Klasse FileSystemObject erzeugen
        Scripting.FileSystemObject fso = new Scripting.FileSystemObjectClass();

        // Ergebnis-Array erzeugen
        Drive[] drives = new Drive[fso.Drives.Count];

        // Alle Laufwerke über die Drives-Auflistung durchgehen, deren Typ,
        // Größe und freien Platz ermitteln und als Drive-Objekt im Array
        // speichern
        int index = 0;
        foreach (Scripting.Drive drive in fso.Drives)
        {
            drives[index] = new Drive();
            drives[index].DriveLetter = drive.DriveLetter;
            drives[index].Type = (DriveType)drive.DriveType;
            drives[index].IsReady = drive.IsReady;

            // Voreinstellung für die Größenangaben
            drives[index].AvailableSpace = -1;
            drives[index].FreeSpace = -1;
        }
    }
}

```

Listing 155: Methode zur Ermittlung von Informationen zu allen logischen Laufwerken eines Systems über die *FileSystemObject*-Klasse

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        drives[index].TotalSize = -1;

        if (drive.IsReady)
        {
            // Die als COM-Variant angegebenen Größenangaben in einen
            // Long-Wert umwandeln
            try
            {
                drives[index].AvailableSpace =
                    Convert.ToInt64(drive.AvailableSpace);
                drives[index].FreeSpace = Convert.ToInt64(drive.FreeSpace);
                drives[index].TotalSize = Convert.ToInt64(drive.TotalSize);
            }
            catch {}
        }
        index++;
    }

    // Ergebnis-Array zurückgeben
    return drives;
}
}

```

Listing 155: Methode zur Ermittlung von Informationen zu allen logischen Laufwerken eines Systems über die `FileSystemObject`-Klasse (Forts.)

Listing 156 zeigt eine beispielhafte Anwendung dieser Klasse. Das Programm gibt Informationen zu allen logischen Systemlaufwerken aus.

```

// Alle Laufwerke des Systems auslesen
Drive[] drives = Drive.GetLocalDrives();

// Das zurückgegebene Array durchgehen und den Laufwerk-Typ ermitteln
for (int i = 0; i < drives.Length; i++)
{
    // Laufwerk-Buchstabe auslesen
    Console.Write(drives[i].DriveLetter + ": ");

    // Typ des Laufwerks auslesen
    switch (drives[i].Type)
    {
        case Drive.DriveType.CDRom:
            Console.WriteLine("CD-ROM");
            break;

        case Drive.DriveType.Fixed:
            Console.WriteLine("Festplatten-Partition");
            break;
    }
}

```

Listing 156: Anwendung der Methode zum Ermitteln von Laufwerkinformationen

```
case Drive.DriveType.RamDisk:
    Console.WriteLine("Ram-Disk");
    break;

case Drive.DriveType.Remote:
    Console.WriteLine("Netzwerklaufwerk");
    break;

case Drive.DriveType.Removable:
    Console.WriteLine("Entfernbares Laufwerk");
    break;

case Drive.DriveType.UnknownType:
    Console.WriteLine("Unbekannter Typ");
    break;
}

// Größenangaben ausgeben
Console.WriteLine("Größe: {0} Byte.", drives[i].TotalSize);
Console.WriteLine("Freier Platz: {0} Byte.", drives[i].FreeSpace);
Console.WriteLine();
}
```

Listing 156: Anwendung der Methode zum Ermitteln von Laufwerkinformationen (Forts.)

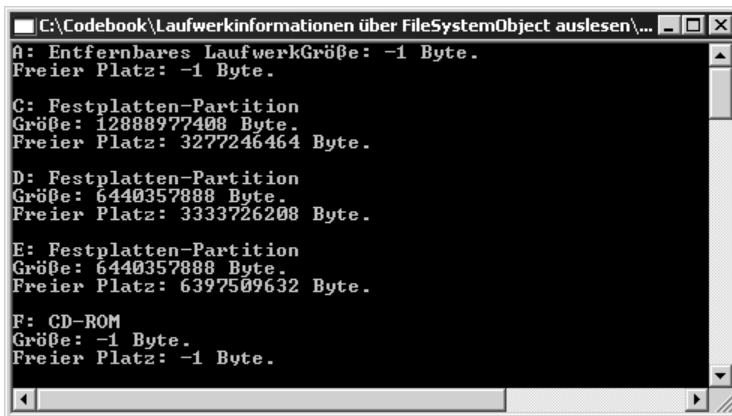


Abbildung 51: Auslesen von Laufwerkinformationen in einer Konsolenanwendung

Laufwerkinformationen über die API-Funktion `GetDriveType` auslesen

Die Windows-API-Funktion `GetDriveType` ermöglicht es, den Typ eines Laufwerks auszulesen, ohne dass weitere Komponenten installiert sein müssen. Sie übergeben dieser Funktion einen String mit dem Wurzelordner des Laufwerks und erhalten einen `uint`-Wert zurück, der den Typ definiert. Die für die einzelnen Typen verwendeten Konstanten können Sie der Klasse `Drive` entnehmen, die ich basierend auf dem vorhergehenden Rezept so abgeändert habe, dass diese nun die Funktion `GetDriveType` verwendet. Da über diese Funktion nur der

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Laufwerktyp ermittelt werden kann, habe ich die Ermittlung der Größe des Laufwerks und des freien Speicherplatzes zunächst entfernt. Diese finden Sie aber im Rezept 89.

```

/* Klasse, die ein Laufwerk repräsentiert */
public class Drive
{
    /* Konstanten für GetDriveType */
    private const uint DRIVE_UNKNOWN = 0;    // Typ konnte nicht ermittelt
                                              // werden
    private const uint DRIVE_NO_ROOT_DIR = 1; // Pfad ist ungültig
    private const uint DRIVE_REMOVABLE = 2;   // Entfernbares LW
    private const uint DRIVE_FIXED = 3;       // Festplatten-LW
    private const uint DRIVE_REMOTE = 4;       // Netzwerklaufwerk
    private const uint DRIVE_CDROM = 5;        // CD-ROM-Laufwerk
    private const uint DRIVE_RAMDISK = 6;      // RAM-Disk

    /* Deklaration der API-Funktion GetDriveType */
    [DllImport("Kernel32.dll")]
    private static extern uint GetDriveType(string lpRootPathName);

    /* Aufzählung für den Laufwerkstyp */
    public enum DriveType: uint
    {
        CDROM = DRIVE_CDROM,
        Fixed = DRIVE_FIXED,
        RamDisk = DRIVE_RAMDISK,
        Remote = DRIVE_REMOTE,
        Removable = DRIVE_REMOVABLE,
        UnknownType = DRIVE_UNKNOWN
    }

    /* Eigenschaften */
    public string DriveLetter;
    public DriveType Type;

    /* Statische Methode zum Auslesen aller Laufwerke des Systems */
    public static Drive[] GetLocalDrives()
    {
        // Alle Laufwerke über Environment.GetLogicalDrives
        string[] logicalDrives = Environment.GetLogicalDrives();

        // Ergebnis-Array erzeugen
        Drive[] drives = new Drive[logicalDrives.Length];

        // Die logischen Laufwerke durchgehen
        for (int i = 0; i < logicalDrives.Length; i++)
        {
            // Drive-Instanz erzeugen
            drives[i] = new Drive();
        }
    }
}

```

Listing 157: Ermitteln aller Laufwerke eines Systems mit Informationen zum Typ über die API-Funktion `GetDriveType`

```

        // Laufwerk-Wurzelordner ohne Doppelpunkt und Backslash speichern
        drives[i].DriveLetter = logicalDrives[i].Replace(":\\", "");

        // Typ des Laufwerks ermitteln und speichern
        drives[i].Type = (DriveType)GetDriveType(logicalDrives[i]);
    }

    // Ergebnis-Array zurückgeben
    return drives;
}
}

```

Listing 157: Ermitteln aller Laufwerke eines Systems mit Informationen zum Typ über die API-Funktion `GetDriveType` (Forts.)

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

88 Alle Laufwerke eines bestimmten Typs ermitteln

Um bei der Ermittlung der Laufwerke eines Systems eine Vorauswahl des Laufwerktyps treffen zu können, habe ich die Methode `GetLocalDrives` des vorherigen Rezepts mit einer weiteren Variante überladen (die erste Variante wird allerdings in der Klasse `Drive` in Listing 89 nicht dargestellt).

`GetLocalDrives` ermittelt alle Laufwerke eines bestimmten, anzugebenden Typs, erzeugt mit den gelesenen Informationen Instanzen der Klasse `Drive` und gibt diese in einem Array zurück. Damit auf dem Zielsystem möglichst wenig vorausgesetzt wird, setze ich dazu die API-Funktion `GetDriveType` ein.

Die neue Variante der Methode `GetLocalDrives` erwartet ein Argument vom Aufzählungstyp `DriveType`. Die konstanten Werte dieser Aufzählung korrespondieren (wie im vorhergehenden Rezept) mit den Werten, die `GetDriveType` zurückgibt, und können deswegen direkt verglichen werden. Etwas problematisch ist die Rückgabe des Ergebnis-Arrays, da vor dem Lesen aller Laufwerke nicht ermittelt werden kann, wie viele Laufwerke des angefragten Typs existieren. Deswegen geht die Methode die ermittelten logischen Systemlaufwerke zweimal durch. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` einbinden.

```

/* Klasse, die ein Laufwerk repräsentiert */
public class Drive
{
    /* Konstanten für GetDriveType */
    private const uint DRIVE_UNKNOWN = 0;        // Unbekannter Typ
    private const uint DRIVE_NO_ROOT_DIR = 1;    // Pfad ist ungültig
    private const uint DRIVE_REMOVABLE = 2;      // Entfernbare LW
    private const uint DRIVE_FIXED = 3;         // Festplatten-LW

```

Listing 158: Methode zur Ermittlung aller Laufwerke eines bestimmten Typs

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

private const uint DRIVE_REMOTE = 4;      // Netzwerklaufwerk
private const uint DRIVE_CDROM = 5;       // CD-ROM-Laufwerk
private const uint DRIVE_RAMDISK = 6;     // RAM-Disk

/* Deklaration der API-Funktion GetDriveType */
[DllImport("Kernel32.dll")]
private static extern uint GetDriveType(string lpRootPathName);

/* Aufzählung für den Laufwerkstyp */
public enum DriveType: uint
{
    CDRom = DRIVE_CDROM,
    Fixed = DRIVE_FIXED,
    RamDisk = DRIVE_RAMDISK,
    Remote = DRIVE_REMOTE,
    Removable = DRIVE_REMOVABLE,
    UnknownType = DRIVE_UNKNOWN
}

/* Eigenschaften */
public string DriveLetter;
public DriveType Type;

/* Statische Methode zum Auslesen aller Laufwerke des Systems */
public static Drive[] GetLocalDrives(DriveType type)
{
    // Alle Laufwerke über Environment.GetLogicalDrives ermitteln
    string[] logicalDrives = Environment.GetLogicalDrives();

    // Die logischen Laufwerke durchgehen, den Typ vergleichen und die
    // Anzahl der zu erzeugenden Drive-Instanzen ermitteln
    int count = 0;
    for (int i = 0; i < logicalDrives.Length; i++)
        if ((DriveType)GetDriveType(logicalDrives[i]) == type)
            count++;

    // Ergebnis-Array erzeugen
    Drive[] drives = new Drive[count];

    // Die logischen Laufwerke noch einmal durchgehen und Drive-Instanzen
    // für die Laufwerke des angefragten Typs im Array speichern
    int index = 0;
    for (int i = 0; i < logicalDrives.Length; i++)
    {
        if ((DriveType)GetDriveType(logicalDrives[i]) == type)
        {
            // Drive-Instanz erzeugen
            drives[index] = new Drive();

            // Laufwerk-Wurzelordner ohne Doppelpunkt und Backslash speichern
            drives[index].DriveLetter = logicalDrives[i].Replace(":\\", "");
        }
    }
}

```

Listing 158: Methode zur Ermittlung aller Laufwerke eines bestimmten Typs (Forts.)


```
// Typ des Laufwerks speichern
drives[index].Type = type;

    index++;
}
}

// Ergebnis-Array zurückgeben
return drives;
}
}
```

Listing 158: Methode zur Ermittlung aller Laufwerke eines bestimmten Typs (Forts.)

Das folgende Beispiel zeigt eine Anwendung dieser Methode in einer einfachen Konsolenanwendung.

```
// Festplatten-Partitionen ermitteln ...
Console.WriteLine("Alle Festplatten-Partitionen:");
Drive[] drives = Drive.GetLocalDrives(Drive.DriveType.Fixed);

// ...und durchgehen
for (int i = 0; i < drives.Length; i++)
    Console.WriteLine(drives[i].DriveLetter);

// CD-ROM-Laufwerke ermitteln ...
Console.WriteLine("\r\nAlle CD-ROM-Laufwerke:");
drives = Drive.GetLocalDrives(Drive.DriveType.CDRom);

// ... und durchgehen
for (int i = 0; i < drives.Length; i++)
    Console.WriteLine(drives[i].DriveLetter);
```

Listing 159: Beispiel-Anwendung der GetLocalDrives-Methode



Abbildung 52: Die Beispielanwendung zur Ermittlung der Festplattenpartitionen und der CD-ROM-Laufwerke in Aktion

Basics

Datum
Zeit

Anwen-
dunge

Datei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

89 Größe und freien Speicherplatz eines Laufwerks ermitteln

Zur Ermittlung der Größe und des freien Speicherplatzes eines Laufwerks können Sie, wie schon bei der Ermittlung der Laufwerke eines Systems (im Rezept 87), eine Instanz der COM-Klasse `FileSystemObject`, WMI oder das Windows-API verwenden. Das .NET Framework bietet zurzeit leider noch keine Möglichkeit, diese Informationen auszulesen.

Da WMI und die `FileSystemObject`-Klasse die Installation von Komponenten auf dem Zielrechner voraussetzen, setze ich zur Ermittlung der Größe und des freien Speicherplatzes eines Laufwerks die Windows-API-Funktion `GetDiskFreeSpaceEx`, die alle gewünschten Informationen liefert. Diese Funktion, die in beiden Windows-Linien ab Windows 95 OSR2 und NT 4 verfügbar ist, wird folgendermaßen deklariert:

```
[DllImport("Kernel32", SetLastError=true)]
private static extern int GetDiskFreeSpaceEx(
    string lpDirectoryName,
    ref ulong lpFreeBytesAvailable,
    ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);
```

Im Attribut `DllImport` ist zunächst, neben der DLL dieser Funktion, mit `SetLastError=true` angegeben, dass `GetDiskFreeSpaceEx` die API-Funktion `SetLastError` aufruft, damit eventuelle Fehler später ausgelesen werden können.

Am Argument `lpDirectoryName` übergeben Sie das Kürzel des Laufwerks (*c:*, *d:* etc.). In der am Argument `lpFreeBytesAvailable` übergebenen Variablen liefert die Funktion dann die Anzahl der freien Bytes zurück, die dem aktuellen Benutzer zur Verfügung steht. Diese Anzahl kann kleiner sein als die Anzahl der insgesamt freien Bytes, falls das System eine benutzerspezifische Verwaltung des freien Platzes unterstützt. Am Argument `lpTotalNumberOfBytes` erhalten Sie die Größe des Laufwerks, das Argument `lpTotalNumberOfFreeBytes` wird mit der Gesamtanzahl der freien Bytes (benutzerunabhängig) belegt. Im Erfolgsfall gibt `GetDiskFreeSpaceEx` einen Wert ungleich Null zurück. Falls Sie einen ungültigen Laufwerk-Bezeichner übergeben oder ein sonstiger Fehler auftritt, wird Null zurückgegeben.

Die Methode `GetDriveSpace` in Listing 160 setzt den Aufruf dieser Funktion um. Um Fehler besser auswerten zu können, ermittelt `GetDriveSpace` bei der Rückgabe von `false` den aufgetretenen API-Fehler über die Methode `GetLastWin32Error` der Klasse `System.Runtime.InteropServices.Marshal` und setzt diese Fehlernummer über die API-Funktion `FormatMessage` in eine Fehlerbeschreibung um. Mit dieser erzeugt `GetDriveSpace` schließlich im Fehlerfall eine Ausnahme vom Typ `IOException`.

`GetDriveSpace` erfordert den Import der Namensräume `System`, `System.IO`, `System.Text` und `System.Runtime.InteropServices`.

```

/* Deklaration der verwendeten API-Funktionen und Konstanten */
[DllImport("Kernel32.dll", SetLastError=true)]
private static extern int GetDiskFreeSpaceEx(string lpDirectoryName,
    ref ulong lpFreeBytesAvailable, ref ulong lpTotalNumberOfBytes,
    ref ulong lpTotalNumberOfFreeBytes);

[DllImport("Kernel32.dll")]
private static extern int FormatMessage(int dwFlags, string lpSource,
    int dwMessageId, int dwLanguageId, StringBuilder lpBuffer, int nSize,
    string [] Arguments);

private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;

/* Methode zur Ermittlung der Größe und des freien Speicherplatzes eines
 * Laufwerks */
public static void GetDriveSpace(string drive, ref ulong totalSize,
    ref ulong totalSpace, ref ulong userSpace)
{
    // GetDiskFreeSpaceEx aufrufen
    if (GetDiskFreeSpaceEx(drive, ref userSpace, ref totalSize,
        ref totalSpace) == 0)
    {
        // Fehler bei der Ausführung: API-Fehler ermitteln, dessen Beschreibung
        // auslesen und eine Ausnahme werfen
        int apiError = Marshal.GetLastWin32Error();
        StringBuilder errorMessage = new StringBuilder(1024);
        FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, null, apiError, 0,
            errorMessage, 1024, null);
        throw new IOException(errorMessage.ToString());
    }
}

```

Listing 160: Methode zur Ermittlung der Größe und des freien Speicherplatzes eines Laufwerks

Listing 161 zeigt eine Beispiel-Anwendung dieser Methode:

```

// Variablen für die Aufnahme der Größeninformationen
ulong totalSize = 0, totalSpace = 0, userSpace = 0;

try
{
    // Größe und freien Speicherplatz des Laufwerks C: ermitteln
    GetDriveSpace("c:", ref totalSize, ref totalSpace, ref userSpace);

    // Ergebnis ausgeben
    Console.WriteLine("Größe: {0} Byte", totalSize);
    Console.WriteLine("Gesamter freier Platz: {0} Byte", totalSpace);
    Console.WriteLine("Benutzerspezifischer freier Platz: {0} Byte",
        userSpace);
}

```

Listing 161: Anwendung der Methode GetDriveSpace

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
catch (IOException ex)
{
    // Fehlerauswertung
    Console.WriteLine("Fehler beim Lesen der Größeninformationen " +
        "von Laufwerk c: {0}.", ex.Message);
}
```

Listing 161: Anwendung der Methode `GetDriveSpace` (Forts.)



Abbildung 53: Das Beispielprogramm zum Auslesen von Laufwerk-Größeninformationen

90 Überprüfen, ob ein Ordner existiert

Mit .NET-Features können Sie zwei Wege gehen um zu überprüfen, ob ein Ordner existiert. Der einfachste Weg ist, dazu die statische `Exists`-Methode der `File`-Klasse aus dem Namensraum `System.IO` zu verwenden:

```
string folderName = "c:\\temp";
if (Directory.Exists(folderName))
    Console.WriteLine(folderName + " existiert");
else
    Console.WriteLine(folderName + " existiert nicht");
```

Alternativ können Sie auch ein `DirectoryInfo`-Objekt (ebenfalls aus dem Namensraum `System.IO`) erzeugen und dessen `Exists`-Eigenschaft abfragen:

```
string folderName = "c:\\temp";
DirectoryInfo di = new DirectoryInfo(folderName);
if (di.Exists)
    Console.WriteLine(folderName + " existiert");
else
    Console.WriteLine(folderName + " existiert nicht");
```

Beide Alternativen funktionieren auch ohne Probleme mit Netzwerkordnern, sofern der in Windows eingeloggte Benutzer entsprechende Zugriffsrechte für die Netzwerkressource besitzt.

91 Ordner erzeugen

Ordner können Sie sehr einfach über die Methode `CreateDirectory` der Klasse `System.IO.Directory` erzeugen. Dieser Methode übergeben Sie dazu einen kompletten Pfadnamen. Sie können auch Ordner erzeugen, deren übergeordnete Ordner noch nicht existieren.

`CreateDirectory` führt immer zum Erfolg, wenn die Pfadangabe syntaktisch in Ordnung ist, auch wenn der angegebene Ordner bereits existiert. Für den Fall, dass die Pfadangabe nicht korrekt ist, fängt der folgende Programmcode, der den Ordner `C:\Test\Test\Test` erzeugt, die dann erzeugte Ausnahme ab. Das Programm erfordert aufgrund der Verwendung der `MessageBox`-Klasse die Referenzierung der Assembly `System.Windows.Forms.dll` und die Einbindung des Namensraums `System.Windows.Forms`. Außerdem müssen die Namensräume `System` und `System.IO` importiert werden.

```
string folderName = @"C:\Test\Test\Test";
try
{
    Directory.CreateDirectory(folderName);
}
catch (Exception ex)
{
    MessageBox.Show(
        "Fehler beim Erzeugen des Ordners '" + folderName + "': " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```

Listing 162: Erzeugen eines Ordners

92 Ordner mit .NET-Features kopieren

Das .NET Framework bietet nach meinen Recherchen keine direkte Möglichkeit, einen ganzen Ordner mit Inhalt zu kopieren. Wollen Sie dazu .NET-Features verwenden, müssen Sie den Zielordner zunächst selbst erzeugen. Dann gehen Sie rekursiv den Quellordner und dessen Unterordner durch, erzeugen entsprechende Unterordner im Zielordner und kopieren die einzelnen Dateien in den jeweiligen Ordner. Die Methode `Copy` in Listing 163 implementiert ein solches Kopieren.

`Copy` überprüft zunächst, ob der angegebene Zielordner bereits existiert, und erzeugt in diesem Fall eine Ausnahme vom Typ `IOException` um zu verhindern, dass die im Zielordner bereits vorhandenen Dateien beim Kopieren mit den neuen Dateien unkontrolliert vermischt werden. Wenn Sie das Mischen vorhandener mit neuen Dateien ermöglichen wollen, können Sie die Überprüfung auch weglassen.

Danach erzeugt `Copy` über `System.IO.Directory.CreateDirectory` den Zielordner, allerdings ohne dessen Unterordner. Diese werden in der privaten Methode `CopySubFoldersAndFiles` erzeugt. `CopySubFoldersAndFiles`, der ein `DirectoryInfo`-Objekt und der Name des Quell- und des Zielordners übergeben wird, geht zunächst alle Unterordner des jeweiligen Ordners durch.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Diese werden über die Methode `GetDirectories` des aktuellen Ordners ermittelt. `GetDirectories` gibt ein Array mit `DirectoryInfo`-Objekten zurück, das dann in einer `for`-Schleife durchlaufen wird. Innerhalb dieser Schleife ermittelt `CopySubFoldersAndFiles` zunächst den Pfad des Ziel-Untersordners, indem der Pfad des Quellordners einfach durch den Pfad des Zielordners ersetzt wird. Danach erzeugt die Methode den jeweiligen Zielordner und ruft sich selbst unter Übergabe eines `DirectoryInfo`-Objekts für den Zielordner auf um dessen Unterordner zu erzeugen und die enthaltenen Dateien zu kopieren.

Nach dem Durchgehen der Unterordner kopiert `CopySubFoldersAndFiles` die im jeweiligen Ordner enthaltenen Dateien, wobei der Zieldateiname wieder über das Ersetzen des Quellordnernamens durch den Zielordnernamen ermittelt wird.

Tritt beim Erzeugen eines Ordners oder beim Kopieren einer Datei eine Ausnahme auf, wird diese einfach an den Aufrufer weitergegeben, da Ausnahmen innerhalb der Methoden `CopySubFoldersAndFiles` und `Copy` nicht abgefangen werden. Deshalb bricht die `Copy`-Methode sofort ab, sobald das Erzeugen eines Ordners oder das Kopieren einer Datei fehlschlägt.

Zum Kompilieren dieser Methode müssen Sie den die Namensräume `System` und `System.IO` einbinden.

```
public static void Copy(string sourceFolderName, string destFolderName)
{
    // DirectoryInfo-Objekt für den Quellordner erzeugen
    DirectoryInfo sourceFolder = new DirectoryInfo(sourceFolderName);

    // Überprüfen, ob der Zielordner bereits existiert
    if (Directory.Exists(destFolderName))
    {
        // Ausnahme erzeugen
        throw new IOException("Der Zielordner '" + destFolderName +
            "' existiert bereits");
    }

    // Zielordner anlegen
    Directory.CreateDirectory(destFolderName);

    // Methode zum Kopieren der Unterordner und Dateien aufrufen
    CopySubFoldersAndFiles(sourceFolder, sourceFolderName, destFolderName);
}

/* Methode zum rekursiven Kopieren eines Ordners */
private static void CopySubFoldersAndFiles(DirectoryInfo folder,
    string sourceFolderName, string destFolderName)
{
    // Alle Unterordner des übergebenen Ordners durchgehen
    DirectoryInfo[] subFolders = folder.GetDirectories();
    for (int i = 0; i < subFolders.Length; i++)
    {
        // Pfad für den Ziel-Untersordner ermitteln, indem der Pfad zum
```

Listing 163: Methode zum rekursiven Kopieren eines Ordners

```

// Quellordner durch den Pfad zum Zielordner ersetzt wird
string destSubFolderName = subFolders[i].FullName.Replace(
    sourceFolderName, destFolderName);

// Unterordner im Zielordner erzeugen
Directory.CreateDirectory(destSubFolderName);

// Funktion rekursiv aufrufen um zunächst die weiteren Unterordner
// zu erzeugen
CopySubFoldersAndFiles(subFolders[i], sourceFolderName,
    destFolderName);
}

// Die im Ordner enthaltenen Dateien ermitteln
FileInfo[] files = folder.GetFiles();

// Alle Dateien durchgehen
for (int i = 0; i < files.Length; i++)
{
    // Ziel-Dateiname ermitteln, indem der Pfad zum Quellordner
    // durch den Pfad zum Zielordner ersetzt wird
    string destFileName = files[i].FullName.Replace(
        sourceFolderName, destFolderName);

    // Datei kopieren
    File.Copy(files[i].FullName, destFileName);
}
}

```

Listing 163: Methode zum rekursiven Kopieren eines Ordners (Forts.)

Hinweis

Da der Abbruch beim Fehlschlagen eines Kopiervorgangs in einigen Fällen nicht gewünscht ist, implementiert das Rezept 94 eine Methode, die beim Fehlschlagen einer Dateioperation die Ausnahme abfängt, den Fehler protokolliert und mit den weiteren Dateioperationen fortfährt.

93 Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage über SHFileOperation kopieren

Zum Kopieren eines kompletten Ordners können Sie neben den .NET-Features auch die Windows-API-Funktion `SHFileOperation` verwenden. Diese Funktion steht ab Windows 95 zur Verfügung.

`SHFileOperation` erlaubt das Kopieren, Verschieben, Umbenennen und Löschen von Ordnern und Dateien. Beim Kopieren eines Ordners wird automatisch der gesamte Inhalt kopiert. Dabei zeigt `SHFileOperation` beim Kopieren größerer Inhalte automatisch einen Fortschrittsdialog an. Über Flags können Sie festlegen, ob die Dateioperation rückgängig gemacht werden kann und der Anwender vor dem Überschreiben vorhandener Ordner gefragt wird, ob

Basics

Datum
Zeit

Anwer-
dungen

Datei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

diese überschrieben werden sollen. Aus einem speziellen Feld der zu übergebenden Struktur können Sie nach dem Kopieren auslesen, ob der Anwender den Kopiervorgang unterbrochen hat.

Die standardmäßige Arbeitsweise von `SHFileOperation` entspricht dem Kopieren von Ordner über den Windows Explorer (bzw. den Arbeitsplatz). Diese Funktion ermöglicht damit das komfortable und einfache Kopieren ganzer Ordner.

`SHFileOperation` erwartet als Argument eine als Referenz zu übergebende, normale (C-) Struktur mit Informationen zu der auszuführenden Aktion. In dieser Struktur, die im Windows-API mit `SHFILEOPSTRUCT` bezeichnet wird, geben Sie die auszuführende Aktion und die Datei- bzw. Ordnernamen an. In einem »Flags«-Feld können Sie die auszuführende Aktion näher spezifizieren. Für das Kopieren eines Ordners können Sie hier z.B. festlegen, dass der Vorgang rückgängig gemacht werden kann und dass Windows vor dem Überschreiben vorhandener Ordner beim Anwender nachfragt.

Die `SHFILEOPSTRUCT`-Struktur sieht in C# folgendermaßen aus:

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
    public IntPtr hwnd;
    public uint wFunc;
    public string pFrom;
    public string pTo;
    public short fFlags;
    [MarshalAs(UnmanagedType.Bool)]
    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
}
```

Das `StructLayout`-Attribut legt zunächst fest, dass Elemente der Struktur beim Export in den nicht von der CLR verwalteten Speicher der (in C oder C++ geschriebenen) API-Funktion sequentiell im Speicher angelegt werden, wie es bei klassischen C-Strukturen der Fall ist. Das `CharSet`-Feld legt fest, dass Zeichenketten innerhalb der Struktur automatisch, je nach Betriebssystem, als Unicode oder ASCII exportiert werden.

Innerhalb der Struktur sind die einzelnen Felder genauso deklariert, wie diese von der API-Funktion erwartet werden. Tabelle 8 beschreibt die Bedeutung der einzelnen Felder.

Feld	Bedeutung
hwnd	Hier können Sie den Handle eines Fensters eintragen, in dem <code>SHFileOperation</code> den Fortschritt beim Kopieren oder Verschieben ausgeben soll.

Tabelle 8: Die Felder der Struktur `SHFILEOPSTRUCT`

Feld	Bedeutung
wFunc	bestimmt die Art der auszuführenden Dateioperation mit den folgenden int-Konstanten: ▶ FO_MOVE (0x0001): Verschieben ▶ FO_COPY (0x0002): Kopieren ▶ FO_DELETE (0x0003): Löschen ▶ FO_RENAME (0x0004): Umbenennen
pFrom	Dieses Feld wird mit einer oder mehreren Datei- oder Ordnerangaben belegt, die die Quelldateien bzw. -ordner festlegen. Mehrere Pfadangaben müssen durch ein 0-Zeichen voneinander getrennt werden. Am Ende muss ein zusätzliches 0-Zeichen stehen.
pTo	Dieses Feld verwaltet beim Kopieren, Verschieben und Umbenennen ähnlich dem pFrom-Feld Pfadangaben zu Zielordner bzw. Zieldateien. Beim Löschen von Dateien oder Ordnern muss dieses Feld leer bleiben (also einen Leerstring oder null speichern, was per Voreinstellung der Fall ist).
fFlags	In fFlags können Sie mit festgelegten Flag-Konstanten spezielle Einstellungen zur ausgewählten Dateioperation festlegen. Die einzelnen Konstanten müssen dazu Oder-verknüpft werden (mit). Die verwendeten Konstanten finden Sie bei der Beschreibung der SHFILEOPSTRUCT-Struktur in der Referenz am Ende des Buchs.
fAnyOperationsAborted	In diesem Feld legt die SHFileOperation-Funktion true ab, wenn der Anwender die Operation abgebrochen hat.
hNameMappings	Dieses Feld können Sie mit dem Handle eines Name-Mapping-Objekts belegen, in das SHFileOperation beim Kopieren/Verschieben automatisch umbenannte Dateien einträgt, wenn die Flags FOF_RENAMEONCOLLISION und FOF_WANTMAPPINGHANDLE gesetzt sind.
lpszProgressTitle	In diesem Feld können Sie den Titel des Fortschritt-Fensters festlegen, wenn Flags den Wert FOF_SIMPLEPROGRESS enthält.

Tabelle 8: Die Felder der Struktur SHFILEOPSTRUCT (Forts.)

Da der originale BOOL-Datentyp des Feldes fAnyOperationsAborted in der API-Funktion einen anderen Speicherbereich verwendet als der C#-bool-Typ, muss dieser beim Übergeben der Struktur in entsprechende Speicherstruktur umgewandelt werden. Dafür steht das MarshalAs-Attribut, das die Umwandlung von C#-Datentypen bei der Übergabe bestimmt. Das Feld fAnyOperationsAborted wird damit in einen 4-Byte-Boolean-Wert konvertiert, bei dem true ungleich 0 und false gleich 0 ist.

Das Kopieren mit Hilfe von SHFileOperation ist nun recht einfach. Zunächst erzeugen Sie eine Instanz der SHFILEOPSTRUCT-Struktur, definieren in dieser Struktur die Funktion, den Quell- und den Zielordner und setzen die Flags so, dass die Aktion rückgängig gemacht werden kann und dass – je nach Wunsch – der Anwender das Überschreiben vorhandener Ordner bestätigen muss. Dann rufen Sie SHFileOperation unter Übergabe dieser Struktur auf. Die

Basics
Datum
Zeit
Anwen-
dungen
Datei-
system
Datei
XML
System
Interne
Formu-
lare
Benut-
Multi-
media
Autom-
tion
Sons-
tiges

Rückgabe von `SHFileOperation` ist laut der Dokumentation 0, wenn die Operation erfolgreich war. In Fehlerfall zeigt die Funktion (normalerweise) automatisch einen Fehlerdialog an, wenn das Feld `fFlags` nicht die Konstante `FOF_NOERRORUI` (0x0400) oder `FOF_SILENT` (0x0004) enthält, und gibt einen (leider nicht dokumentierten) Wert ungleich 0 zurück.

ACHTUNG

Bei meinen Versuchen unter Windows XP und Windows 2000 meldete `SHFileOperation` keinen Fehler beim Versuch, einen Ordner auf ein Laufwerk zu kopieren, das gar nicht existierte (z. B. den Ordner `C:\Projekte` nach `X:\Backup\Projekte`). Auch der Rückgabewert war 0, was es so scheinen lässt, als wäre der Ordner kopiert worden. Nur wenn das Laufwerk vorhanden war, aber der Quell- oder Zielordner nicht existierte, generierte `SHFileOperation` einen Fehler. Außerdem führt der Aufruf von `SHFileOperation` zu einem Fehler, wenn der Quell- oder der dem Zielordner übergeordnete Ordner nicht existieren. Leider ist es nicht sinnvoll, diese automatischen und leider schwer verständlichen Fehlermeldungen¹ von `SHFileOperation` abzuschalten (wie ich am Ende dieses Rezepts erläutere).

Um diese Probleme zu vermeiden, fragt die Methode `APICopy` in Listing 164, die das Kopieren über `SHFileOperation` implementiert, vor dem Kopieren ab, ob der Quellordner und der dem Zielordner übergeordnete Ordner existieren, und generiert im negativen Fall eine Ausnahme vom Typ `IOException`. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Runtime.InteropServices` einbinden.

```
/* Deklaration der verwendeten API-Funktionen, -Konstanten und -Strukturen */
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

private const int FO_COPY = 0x0002;           // Datei oder Ordner kopieren
private const int FOF_ALLOWUNDO = 0x0040;    // Rückgängigmachen erlauben
private const int FOF_NOCONFIRMATION = 0x0010; // keine Nachfrage beim
                                                // Anwender

/* Struktur zur Definition der Dateioperation */
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
    public IntPtr hwnd;
    public int wFunc;
    public string pFrom;
    public string pTo;
    public short fFlags;
    [MarshalAs(UnmanagedType.Bool)]
```

Listing 164: Methode zum rekursiven Kopieren eines Ordners über `SHFileOperation` mit Fortschrittsdialog und Undo-Option

1. Schwer verständlich ist z. B. der Fehler »Datei kann nicht kopiert werden. Die Quelldatei oder vom Quelldatenträger kann nicht gelesen werden« beim Versuch, einen nicht existierenden Ordner zu kopieren.

```

    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
}

/* Methode zum Kopieren eines Ordners */
public static bool APICopy(string sourceFolderName, string destFolderName,
    bool confirmOverwrites)
{
    // Überprüfen, ob der dem Zielordner übergeordnete Ordner existiert, um zum
    // einen das Problem zu vermeiden, dass SHFileOperation beim Kopieren auf
    // ein nicht existierendes Laufwerk ohne Fehler ausgeführt wird und zum
    // anderen den Fehler zu vermeiden, den SHFileOperation meldet, wenn dieser
    // Ordner nicht existiert.
    int i = destFolderName.LastIndexOf("\\");
    string destParentFolder = null;
    if (i > 0)
    {
        destParentFolder = destFolderName.Substring(0, i);
        if (Directory.Exists(destParentFolder) == false)
            // Ziel-Parent-Ordner existiert nicht: Ausnahme werfen
            throw new IOException("Der Ziel-Ordner " + destParentFolder +
                " existiert nicht");
    }

    // Überprüfen, ob der Quellordner existiert
    if (Directory.Exists(sourceFolderName) == false)
        throw new IOException("Der Quell-Ordner " + sourceFolderName +
            " existiert nicht");

    // Struktur für die Dateiinformationen erzeugen
    SHFILEOPSTRUCT fileOp = new SHFILEOPSTRUCT();

    // (Unter-)Funktion definieren (ShFileOperation kann auch Dateien und
    // Ordner löschen, verschieben oder umbenennen)
    fileOp.wFunc = FO_COPY;

    // Quelle und Ziel definieren. Dabei müssen mehrere Datei- oder
    // Ordnerangaben über 0-Zeichen getrennt werden. Am Ende muss ein
    // zusätzliches 0-Zeichen stehen
    fileOp.pFrom = sourceFolderName + "\\x0\\x0";
    fileOp.pTo = destFolderName + "\\x0\\x0";

    // Flags setzen, sodass ein Rückgängigmachen möglich ist und
    // dass - je nach Argument confirmOverwrites - keine Nachfrage
    // beim Überschreiben von Ordnern beim Anwender erfolgt
    if (confirmOverwrites)
        fileOp.fFlags = FOF_ALLOWUNDO;
    else
        fileOp.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;
}

```

Listing 164: Methode zum rekursiven Kopieren eines Ordners über SHFileOperation mit Fortschritts-dialog und Undo-Option (Forts.)

Basics

Datum
Zeit

Anwen-
dungen

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

```

// ShFileOperation unter Übergabe der Struktur aufrufen
int result = SHFileOperation(&fileOp);

// Erfolg oder Misserfolg zurückgeben. SHFileOperation liefert
// 0 zurück, wenn kein Fehler aufgetreten ist, ansonsten einen
// Wert ungleich 0
return (result == 0);
}

```

Listing 164: Methode zum rekursiven Kopieren eines Ordners über SHFileOperation mit Fortschritts-dialog und Undo-Option (Forts.)

Abbildung 54 zeigt eine Anwendung, die über APICopy den Ordner C:\Codebook nach G:\Backup\Codebook kopiert. Da der Zielordner bereits existiert, fragt SHFileOperation den Anwender, ob er die eventuell vorhandenen Dateien überschreiben will. In Abbildung 55 wird der Ordner gerade kopiert, Abbildung 56 zeigt die Fehlermeldung beim Versuch eine Datei zu kopieren, die gerade in Verwendung ist.

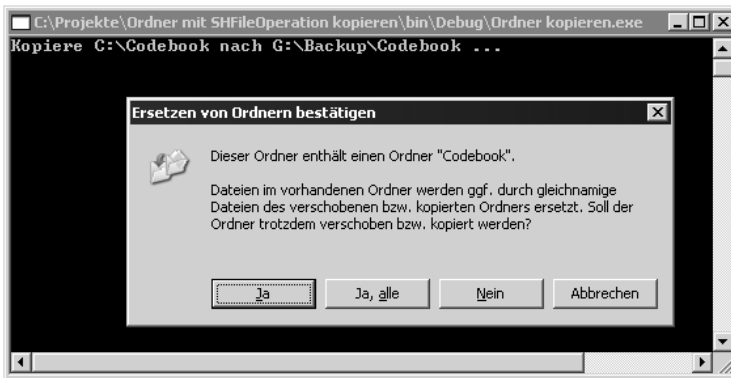


Abbildung 54: Das Beispielprogramm beim Kopieren eines Ordners, der auf dem Zielsystem bereits existiert

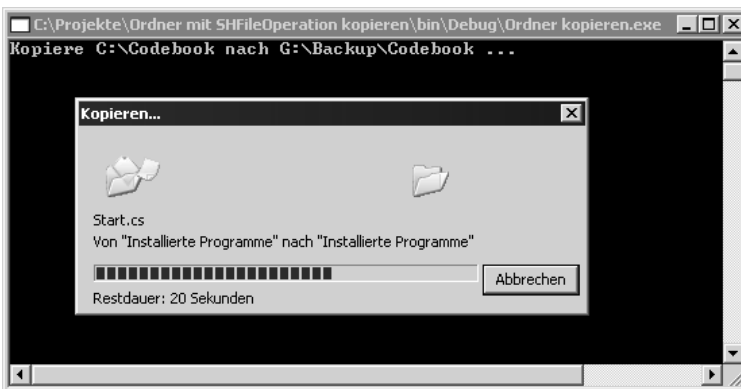


Abbildung 55: Das Beispielprogramm beim Kopieren eines Ordners



Abbildung 56: Automatische Fehlermeldung beim Kopieren einer Datei, die nicht gelesen werden kann

Hinweis

Im Fehlerfall bricht `SHFileOperation` die Dateioperation ab. Wenn beim Kopieren eines Ordners also eine Datei nicht kopiert werden kann, werden auch die noch ausstehenden Dateien und Unterordner nicht weiter kopiert. Eine Lösung dieses Problems liefert das Rezept 94.

An Stelle der systemeigenen Meldungen, die `SHFileOperation` im Fehlerfall ausgibt, wäre es zudem wesentlich schöner, wenn die Methode `APICopy` die Systemfehlermeldungen über das Flag `FOF_NOERRORUI` ausschalten, den Rückgabecode von `SHFileOperation` auswerten, daraus eine Fehlerbeschreibung generieren und schließlich mit dieser Beschreibung eine Ausnahme vom Typ `IOException` erzeugen würde. Leider sind aber die Rückgabewerte von `SHFileOperation` nicht dokumentiert. Die Microsoft-API-Dokumentation sagt lediglich aus, dass diese Funktion im Erfolgsfall 0 und im Fehlerfall einen Wert ungleich 0 zurückgibt (siehe msdn.microsoft.com/library/en-us/shellcc/platform/shell/reference/functions/shfileoperation.asp). Der zurückgegebene Fehlercode lässt sich leider auch nicht über die API-Funktion `FormatMessage` in eine passende Fehlermeldung umsetzen (die damit ermittelten Fehlermeldungen sind absolut unpassend). Die Idee, stattdessen den letzten Windows-Fehlercode zu verwenden (den Sie über die `GetLastWin32Error`-Methode der Klasse `System.Runtime.InteropServices.Marshal` auslesen können) schlug bei meinen Versuchen ebenfalls mit vollkommen unpassenden Fehlermeldungen fehl. Hinzu kommt, dass umfangreiche Diskussionen in verschiedenen Newsgroups die scheinbare Unmöglichkeit, aus dem Rückgabewert von `SHFileOperation` eine Fehlerbeschreibung abzuleiten, belegen. Schade.

94 Ordner mit .NET-Features ohne Abbruch bei einer Ausnahme kopieren

Mich ärgert es immer wieder, wenn Windows beim Kopieren eines umfangreichen Ordners den Kopiervorgang abbricht, nur weil einzelne Dateien gerade nicht kopiert werden können (weil diese z.B. geöffnet sind). Schöner wäre es, wenn die Kopiervorgänge beim Auftreten eines Fehlers einfach mit der nächsten Datei oder dem nächsten Unterordner fortgeführt und die jeweiligen Fehler am Ende gemeldet würden.

Die im Folgenden beschriebene Methode `ExtCopy` implementiert ein solches Vorgehen. Diese Methode kopiert alle Dateien des Quellordners rekursiv ähnlich der `Copy`-Methode des Rezepts 92. Zusätzlich wird der Anwender vor dem Überschreiben einer vorhandenen Datei gefragt, ob diese überschrieben werden soll (Abbildung 57).

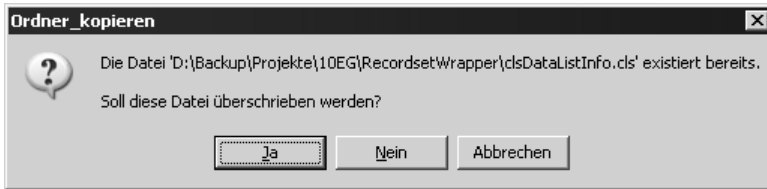


Abbildung 57: Abfrage der ExtCopy-Methode zum Überschreiben einer vorhandenen Datei

Um dem Anwender die Möglichkeit zu geben, alle vorhandenen Dateien zu überschreiben, fragt die Methode einmalig ab, ob dies gewünscht ist (Abbildung 58).

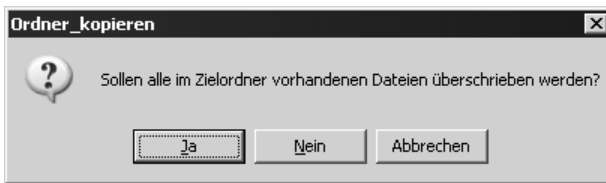


Abbildung 58: Abfrage der ExtCopy-Methode zum Überschreiben aller Dateien

Die dazu notwendigen Informationen werden in zwei privaten Eigenschaften verwaltet. Beide Dialoge ermöglichen einen Abbruch, der aus Sicherheitsgründen in eine Ausnahme vom Typ `IOException` umgesetzt wird.

Fehler beim Kopieren einer Datei werden abgefangen und in einer speziellen Auflistung von Kopierfehler-Objekten protokolliert. Um dies zu realisieren habe ich zunächst eine Klasse `CopyFault` entwickelt, deren Instanzen je einen Kopierfehler speichern sollen. Die weitere Klasse `CopyFaults` implementiert eine Auflistung von `CopyFault`-Objekten und ist dazu von der Klasse `CollectionBase` abgeleitet. `CopyFaults` implementiert lediglich die Methode `Add` und den Indexer `this`, spezialisiert auf `CopyFault`-Objekte.

Um das Programm kompilieren zu können, müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO` und `System.Windows.Forms` importieren.

```

/* Klasse für die Speicherung fehlgeschlagener Kopieroperationen */
public class CopyFault
{
    // Eigenschaften
    public bool IsFile;
    public string Source;
    public string Destination;
    public string Error;

    // Konstruktor
    public CopyFault(bool isFile, string source, string destination,

```

Listing 165: Klassen für die Verwaltung von Kopierfehlern

```

        string error)
    {
        this.IsFile = isFile;
        this.Source = source;
        this.Destination = destination;
        this.Error = error;
    }
}

/* Klasse für eine Auflistung von CopyFault-Objekten */
public class CopyFaults: System.Collections.CollectionBase
{
    // Methode zum Hinzufügen
    public void Add(CopyFault cf)
    {
        this.List.Add(cf);
    }

    // Indexer zum Zugriff auf die Elemente
    public CopyFault this[int index]
    {
        get {return (CopyFault)this.List[index];}
        set {this.List[index] = value;}
    }
}

```

Listing 165: Klassen für die Verwaltung von Kopierfehlern (Forts.)

Die Klasse `FolderUtil` besitzt neben zwei privaten Eigenschaften, die intern für das Überschreiben von Dateien verwendet werden, eine Methode `ExtCopy`, die einen Ordner kopiert. Diese Methode initialisiert zunächst die privaten Eigenschaften und ruft dann die private Methode `CopySubFoldersAndFiles` auf:

```

public class FolderUtil
{
    /* Eigenschaften, die für das Überschreiben aller Dateien verwendet
     * werden */
    private static bool overwriteAllFiles;
    private static bool alreadyAskedForOverwriteAllFiles;

    /* Methode zum Kopieren eines Ordners */
    public static bool ExtCopy(string sourceFolderName, string destFolderName,
        CopyFaults copyFaults)
    {
        /* Datei-Überschreib-Flags voreinstellen */
        overwriteAllFiles = false;
        alreadyAskedForOverwriteAllFiles = false;

        // Rekursive Methode zum Kopieren der Unterordner und Dateien aufrufen

```

Listing 166: Methode zum Start des rekursiven Kopierens eines Ordners

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

CopySubFoldersAndFiles(new DirectoryInfo(sourceFolderName),
    sourceFolderName, destFolderName, copyFaults);

// Erfolg zurückmelden
return (copyFaults.Count == 0) ;
}

```

Listing 166: Methode zum Start des rekursiven Kopierens eines Ordners (Forts.)

Die Methode `CopySubFoldersAndFiles` geht zuerst alle Unterordner des Ordners durch, erzeugt den jeweiligen Unterordner im Zielordner und ruft sich danach selbst auf, um die weiteren Unterordner zu erzeugen. Danach kopiert `CopySubFoldersAndFiles` die im jeweiligen Ordner enthaltenen Dateien in den jeweiligen Zielordner.

`CopySubFoldersAndFiles` wird ein `DirectoryInfo`-Objekt übergeben, das den Ordner repräsentiert. Beim rekursiven Aufruf übergibt `CopySubFoldersAndFiles` ein solches für jeden Unterordner an sich selbst. Da innerhalb der Methode beim Kopieren der Dateien Informationen über den Haupt-Quell- und -Zielordner benötigt werden (um den zum jeweiligen Zielordner relativen Dateinamen zu ermitteln), werden dieser Methode zudem die Pfade dieser Ordner übergeben.

`CopySubFoldersAndFiles` erzeugt im Fehlerfall ein neues `CopyFault`-Objekt und hängt dieses an die `CopyFaults`-Auflistung an, die ebenfalls (über die `ExtCopy`-Methode) übergeben wurde.

```

private static bool CopySubFoldersAndFiles(DirectoryInfo folder,
    string mainSourceFolderName, string mainDestFolderName,
    CopyFaults copyFaults)
{
    // Zielordner anlegen
    try
    {
        // Zielordnername ermitteln
        string destFolderName = folder.FullName.Replace(mainSourceFolderName,
            mainDestFolderName);

        // Ordner anlegen
        Directory.CreateDirectory(destFolderName);
    }
    catch (IOException ex)
    {
        // Fehler in der CopyFaults-Auflistung dokumentieren
        copyFaults.Add(new CopyFault(false, mainSourceFolderName,
            mainDestFolderName, ex.Message));
    }

    // Alle Unterordner des übergebenen Ordners durchgehen
    DirectoryInfo[] subFolders = folder.GetDirectories();
    for (int i = 0; i < subFolders.Length; i++)
    {

```

Listing 167: Methode zum rekursiven Kopieren eines Ordners

```

        // Pfad für den Ziel-Unterordner ermitteln, indem der Pfad zum
        // Quellordner durch den Pfad zum Zielordner ersetzt wird
        string destSubFolderName = subFolders[i].FullName.Replace(
            mainSourceFolderName, mainDestFolderName);

        // Funktion rekursiv aufrufen um zunächst die weiteren Unterordner
        // zu erzeugen
        CopySubFoldersAndFiles(subFolders[i], mainSourceFolderName,
            mainDestFolderName, copyFaults);
    }

    // Die im Ordner enthaltenen Dateien ermitteln
    FileInfo[] files = folder.GetFiles();

    // Alle Dateien durchgehen
    for (int i = 0; i < files.Length; i++)
    {
        // Ziel-Dateiname ermitteln, indem der Pfad zum Quellordner
        // durch den Pfad zum Zielordner ersetzt wird
        string destFileName = files[i].FullName.Replace(
            mainSourceFolderName, mainDestFolderName);

        // Flag setzen, das festlegt, ob die Datei kopiert werden soll
        bool performCopyOperation;
        performCopyOperation = true;

        // Überprüfen, ob die Datei bereits existiert
        if (File.Exists(destFileName))
        {
            // Fragen, ob die Datei überschrieben werden soll, falls der
            // Anwender dies zuvor noch nicht für alle Dateien und Ordner
            // gemeinsam bestätigt hat
            if (overwriteAllFiles == false)
            {
                switch (MessageBox.Show("Die Datei '" + destFileName +
                    "' existiert bereits.\r\n\r\n" +
                    "Soll diese Datei überschrieben werden?",
                    Application.ProductName, MessageBoxButtons.YesNoCancel,
                    MessageBoxIcon.Question))
                {
                    case DialogResult.Yes:
                        // Nachfragen, ob alle Dateien überschrieben werden
                        // sollen, falls dies noch nicht geschehen ist
                        if (alreadyAskedForOverwriteAllFiles == false)
                        {
                            switch (MessageBox.Show("Sollen alle im Zielordner " +
                                "vorhandenen Dateien überschrieben werden?",
                                Application.ProductName,
                                MessageBoxButtons.YesNoCancel,
                                MessageBoxIcon.Question))
                            {

```

Listing 167: Methode zum rekursiven Kopieren eines Ordners (Forts.)

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        case DialogResult.Yes:
            // Flag setzen, das das Überschreiben steuert
            overwriteAllFiles = true;
            break;

        case DialogResult.Cancel:
            // Anwender hat abgebrochen: Ausnahme erzeugen,
            // da der Ordner nicht komplett kopiert werden
            // konnte
            throw new IOException("Benutzerabbruch");
    }

    // Festlegen, dass nicht mehr nachgefragt werden muss
    alreadyAskedForOverwriteAllFiles = true;
}

break;

case DialogResult.No:
    // Festlegen, dass die aktuelle Datei nicht kopiert
    // werden soll
    performCopyOperation = false;
    break;

case DialogResult.Cancel:
    // Anwender hat abgebrochen: Ausnahme erzeugen, da der
    // Ordner nicht komplett kopiert werden konnte
    throw new IOException(
        "Benutzerabbruch");
    }
}

// Datei kopieren, wenn die Operation ausgeführt werden soll
if (performCopyOperation)
{
    try
    {
        File.Copy(files[i].FullName, destFileName,
            true);
    }
    catch (Exception ex)
    {
        // Fehler in der CopyFaults-Auflistung dokumentieren
        copyFaults.Add(new CopyFault(true, files[i].FullName,
            destFileName, ex.Message));
    }
}
else
{
    // Datei sollte nicht überschrieben werden: Fehler in der
    // CopyFaults-Auflistung dokumentieren

```

Listing 167: Methode zum rekursiven Kopieren eines Ordners (Forts.)

```

        copyFaults.Add(new CopyFault(true, files[i].FullName,
            destFileName,
            "Fehlende Anwender-Erlaubnis zum Überschreiben"));
    }
}

// Erfolg zurückmelden
return true;
}
}

```

Listing 167: Methode zum rekursiven Kopieren eines Ordners (Forts.)

Bei der Anwendung der ExtCopy-Methode muss eine CopyFaults-Auflistung übergeben werden. Diese können Sie nach dem Kopieren dazu verwenden, die nicht kopierbaren Dateien aufzulisten, in eine Textdatei zu schreiben oder dem Anwender zum erneuten Kopieren zur Verfügung zu stellen. Der folgende Beispielcode listet die Fehler lediglich auf:

```

string sourceFolderName = @"C:\Codebook";
string destFolderName = @"G:\Backup\Codebook";

Console.WriteLine("Kopiere '" + sourceFolderName + "' nach '" +
    destFolderName + "' ...");

// CopyFaults-Auflistung erzeugen
CopyFaults copyFaults = new CopyFaults();

// Ordner kopieren
try
{
    if (FolderUtil.ExtCopy(sourceFolderName, destFolderName,
        copyFaults))
    {
        Console.WriteLine("Fertig");
    }
}
else
{
    // Beim Kopieren sind Fehler aufgetreten: Alle Fehler
    // durchgehen und ausgeben
    for (int i = 0; i < copyFaults.Count; i++)
    {
        if (copyFaults[i].IsFile)
        {
            // Es handelt sich um eine Datei
            Console.WriteLine("Fehler beim Kopieren der " +
                "Datei '{0}' nach '{1}': {2}",
                copyFaults[i].Source, copyFaults[i].Destination,
                copyFaults[i].Error);
        }
    }
}

```

Listing 168: Anwendung der Klasse zum komfortablen Kopieren

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

else
{
    // Es handelt sich um einen Ordner
    Console.WriteLine("Fehler beim Kopieren des " +
        "Ordners '{0}' nach '{1}': {2}",
        copyFaults[i].Source, copyFaults[i].Destination,
        copyFaults[i].Error);
}
}
}
}
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Kopieren des Ordners: {0}",
        ex.Message);
}
}

```

Listing 168: Anwendung der Klasse zum komfortablen Kopieren (Forts.)

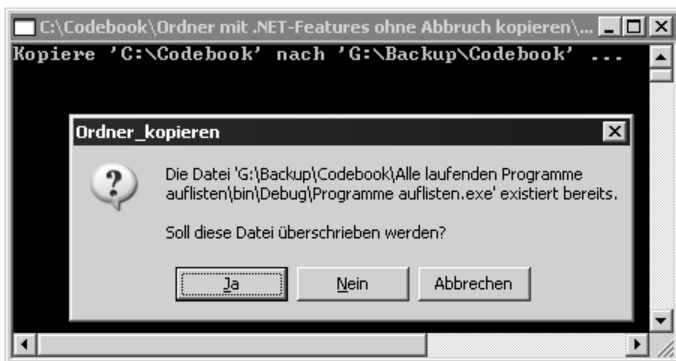


Abbildung 59: Kopieren eines Ordners, dessen Dateien im Zielordner (teilweise) bereits existieren

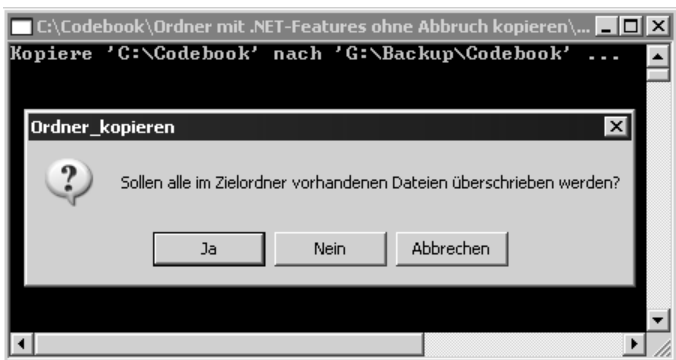


Abbildung 60: Zweite Nachfrage beim Kopieren eines Ordners, dessen Dateien im Zielordner (teilweise) bereits existieren nach der Auswahl der Überschreib-Option

95 Ordner umbenennen

Einen Ordner können Sie sehr einfach über die `Move`-Methode der `Directory`-Klasse (aus dem Namensraum `System.IO`) umbenennen. `Move` erzeugt eine Ausnahme vom Typ `IOException`, wenn das Umbenennen nicht möglich ist. Das folgende Beispiel beachtet diese Tatsache:

```
string sourceDirName = @"C:\Temp\DemoFolder"; // Alter Name
string destDirName = @"C:\Temp\Demo Folder";  // Neuer Name

try
{
    // Ordner umbenennen
    Directory.Move(sourceDirName, destDirName);

    Console.WriteLine("Ordner erfolgreich umbenannt");
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Umbenennen des Ordners: " + ex.Message);
}
```

Listing 169: Umbenennen eines Ordners ohne Undo-Option

96 Ordner verschieben

Wenn Sie einen Ordner verschieben wollen, verwenden Sie prinzipiell denselben Programmcode wie im vorhergehenden Rezept. Sie geben als Zielordner lediglich einen Ordner an, der einen anderen übergeordneten Ordner oder ein anderes Laufwerk aufweist.

Das Beispiel in Listing 170 erfordert den Import der Namensräume `System` und `System.IO`.

```
string sourceFolderName = @"C:\DemoFolder";
string destFolderName = @"C:\Temp\DemoFolder";

try
{
    // Ordner verschieben
    Directory.Move(sourceFolderName, destFolderName);

    Console.WriteLine("Ordner erfolgreich verschoben");
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Verschieben des Ordners: " +
        ex.Message);
}
```

Listing 170: Verschieben eines Ordners ohne Undo-Option

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Das Verschieben von Ordnern über die `Move`-Methode ist nicht allzu komfortabel. Problematisch ist z. B., dass diese Methode mit einer Ausnahme abbricht, wenn eine Zieldatei oder ein Zielordner bereits existiert. Im Rezept 114 finden Sie eine Methode, die das erweiterte Umbenennen und Verschieben von Dateien oder Ordnern über die API-Funktion `SHFileOperation` implementiert. Diese Methode zeigt einen automatischen Fortschrittsdialog an, ermöglicht den Abbruch der Aktion und das Überschreiben vorhandener Dateien bzw. Ordner mit oder ohne Nachfrage beim Anwender.

97 Ordner löschen

Wenn Sie einen Ordner löschen wollen, können Sie dazu die Methode `Delete` der Klasse `System.IO.Directory` verwenden. Am ersten Argument übergeben Sie den Pfadnamen des Ordners, im optionalen zweiten Argument können Sie mit `true` bestimmen, dass auch alle Unterordner und enthaltenen Dateien gelöscht werden:

```
string folderName = @"C:\Test\Test\Test";
try
{
    // Ordner löschen
    Directory.Delete(folderName, true);
}
catch (Exception ex)
{
    MessageBox.Show(
        "Fehler beim Löschen des Ordners '" + folderName + "': " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```

Listing 171: Löschen eines Ordners

Dieses Beispiel benötigt aufgrund der Verwendung der `MessageBox`-Klasse die Referenzierung der Assembly `System.Windows.Forms.dll` und die Einbindung des Namensraums `System.Windows.Forms`. Daneben müssen die Namensräume `System` und `System.IO` importiert werden.

Übergeben Sie am zweiten Argument nichts oder `false`, führt das Löschen eines nicht leeren Ordners zu einer Ausnahme vom Typ `IOException`. Der zu löschende Ordner muss außerdem existieren und löschar sein (nicht schreibgeschützt), ansonsten erzeugt `Delete` ebenfalls eine Ausnahme. Dieselbe Ausnahme wird natürlich auch erzeugt, wenn die Pfadangabe syntaktisch nicht korrekt ist.

Beachten Sie, dass `Delete` Ordner und alle enthaltenen Dateien dauerhaft löscht und nicht in den Papierkorb verschiebt. Das Verschieben in den Papierkorb finden Sie im Rezept 115.

98 Ordnergröße ermitteln

Die für allgemeine Verzeichnisinformationen zuständige Klasse `DirectoryInfo` bietet keine direkte Möglichkeit, die gesamte Größe eines Ordners abzufragen. Leider besteht auch scheinbar keine Möglichkeit, die Ordnergröße über Windows-API-Funktionen oder über WMI abzufragen. Um die Summe der Dateigrößen aller in einem Ordner und dessen Unterordnern enthaltenen Dateien zu ermitteln, müssen Sie den Ordner rekursiv durchgehen und die Dateigröße jeder enthaltenen Datei ermitteln und summieren.

Dazu können Sie ein `DirectoryInfo`-Objekt verwenden. Über dessen `GetDirectories`-Methode erhalten Sie ein `DirectoryInfo`-Array für alle Unterordner. Damit können Sie einen Ordner rekursiv durchgehen. Die `GetFiles`-Methode eines solchen Objekts gibt Ihnen Zugriff auf die enthaltenen Dateien in Form eines Arrays von `FileInfo`-Objekten, aus dessen Eigenschaft `Size` Sie die Größe auslesen können.

Die Methode `GetFolderSize` in Listing 172 setzt das rekursive Durchgehen eines Ordners und das Summieren der Dateigrößen um. `GetFolderSize` ruft dazu die private Methode `SumFolderSize` auf und übergibt ein `DirectoryInfo`-Objekt, das den Ordner repräsentiert. `SumFolderSize` geht zuerst alle Dateien des übergebenen Ordners durch um deren Größe in der Variablen `folderSize` zu summieren. Danach geht diese Methode alle Unterordner durch und ruft sich selbst für jeden dieser Ordner auf. `SumFolderSize` gibt die jeweils ermittelte Ordnergröße zurück. Deshalb kann die Variable `folderSize` um den jeweiligen Rückgabewert erhöht werden. Der erste Aufruf in der rekursiven Kette gibt schließlich die ermittelte Größe an die Methode `GetFolderSize` zurück.

Um die Methoden aus Listing 172 kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
public static long GetFolderSize(string folderName)
{
    // Die rekursive Methode zur Ermittlung der Größe eines Ordners aufrufen
    return SumFolderSize(new DirectoryInfo(folderName));
}

/* Methode zum rekursiven Durchgehen eines Ordners und zur Ermittlung der
 * Gesamtgröße der in einem Ordner und dessen Unterordnern
 * enthaltenen Dateien */
private static long SumFolderSize(DirectoryInfo folder)
{
    // Variable zur Speicherung der jeweils ermittelten Dateigrößen-Summe
    // eines (Unter-)Ordners
    long folderSize = 0;

    // Summe der Größen der Dateien im Ordner ermitteln
    FileInfo[] files = folder.GetFiles();
    for (int i = 0; i < files.Length; i++)
        folderSize += files[i].Length;
}
```

Listing 172: Methoden zum rekursiven Ermitteln der Größe eines Ordners

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

// Unterordner ermitteln und die Methode rekursiv für jeden Unterordner
// aufrufen und dabei die Variable folderSize hochzählen
DirectoryInfo[] subFolders = folder.GetDirectories();
for (int i = 0; i < subFolders.Length; i++)
    folderSize += SumFolderSize(subFolders[i]);

// Die ermittelte Ordnergröße zurückgeben
return folderSize;
}

```

Listing 172: Methoden zum rekursiven Ermitteln der Größe eines Ordners (Forts.)

Abbildung 61 zeigt eine Anwendung, die die Größe des Windows-Systemordners über `GetFolderSize` ermittelt hat.

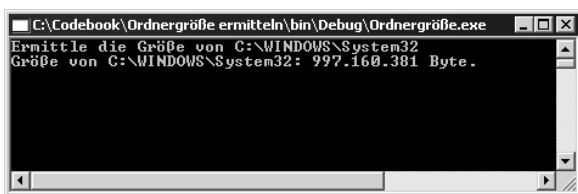


Abbildung 61: Beispielprogramm zur Ermittlung der Größe des Windows-Systemordners

Tip

Den ermittelten Bytewert können Sie über das Rezept 9 in ein besser lesbares Format mit der Einheit Byte, KB, MB oder GB umwandeln.

99 Ordnergröße aller direkten Unterordner ermitteln

Basierend auf dem vorhergehenden Rezept ist es ein Leichtes, die Größe aller direkten Unterordner eines Ordners zu ermitteln. So können Sie z.B. feststellen, welche Ordner besonders viele und/oder große Dateien speichern.

Als Lösung dieses »Problems« habe ich eine Methode `GetSubFolderSizes` entwickelt. Diese Methode verwendet ein `DirectoryInfo`-Objekt zum Durchgehen der Unterordner eines anzugebenden Ordners und ruft für jeden dieser Ordner die Methode `GetFolderSize` aus dem vorhergehenden Rezept auf. Um dem Programmierer zu ermöglichen, dem Anwender den Fortschritt der bei großen Ordnern recht langwierigen Operation anzuzeigen, ruft `GetSubFolderSizes` für jeden neuen Ordner einen Delegate auf, der auf eine Methode verweist, der der Name des aktuellen Ordners übergeben wird.

`GetSubFolderSizes` gibt eine Auflistung von speziellen `FolderSize`-Objekten zurück. Diese Auflistung wird von der Basisklasse `CollectionBase` abgeleitet und auf `FolderSize`-Objekte spezialisiert, sodass bei der Auswertung der Auflistung nicht konvertiert werden muss. Ich habe mich für die Verwendung einer Auflistung entschieden, da diese dynamisch gefüllt werden kann. Bei der Verwendung eines Arrays müsste die Methode vor dem Füllen wissen, wie viele Elemente insgesamt gespeichert werden müssen, was bei einer Ordnerstruktur nur über ein vorheriges rekursives (und damit verlangsames) Durchgehen möglich wäre.

Zum Kompilieren der im Folgenden beschriebenen Klassen müssen Sie die Namensräume `System` und `System.IO` importieren.

Für die Implementierung muss zunächst der Delegate deklariert werden:

```
public delegate void FolderProgress(string currentFolderName);
```

Listing 173: Deklaration eines Delegate für den Fortschritt des Auslesens der Ordnergrößen

Dann folgt die Klasse, deren Instanzen in der Auflistung gespeichert werden sollen, die von `GetSubFolderSizes` zurückgegeben wird:

```
public class FolderSize
{
    /* Eigenschaften */
    public string FolderName;
    public long Size;

    /* Konstruktor */
    public FolderSize(string folderName, long size)
    {
        this.FolderName = folderName;
        this.Size = size;
    }
}
```

Listing 174: Klasse für die Speicherung von Ordner-Größen

Die Auflistung dieser Objekte wird von der Klasse `CollectionBase` abgeleitet, die als Basis-klassse für eigene, typsichere Auflistungen vorgesehen ist:

```
public class FolderSizes: System.Collections.CollectionBase
{
    /* Implementierung des Indizierers */
    public FolderSize this[int index]
    {
        set {List[index] = value;}
        get {return (FolderSize)List[index];}
    }

    /* Implementierung der Add-Methode */
    public void Add(FolderSize fs)
    {
        this.List.Add(fs);
    }
}
```

Listing 175: Spezialisierte Collection für FolderSize-Objekte

Um eine typischere Auflistung zu erzeugen, sollten Sie zumindest den Indizierer (`this`) und die `Add`-Methode programmieren, wie es im Beispiel der Fall ist. Beide Methoden sollten auf die Klasse spezialisiert sein, deren Instanzen die Auflistung verwalten soll. Zusätzlich können Sie die `Remove`-Methode implementieren, wenn Sie auch das Löschen von Elementen zulassen wollen.

Der Methode `GetSubFolderSizes` in Listing 176 wird neben dem Namen des Ordners eine Instanz des Fortschritt-Delegates (oder `null`) übergeben. Der Aufrufer muss dann eine dem Delegate entsprechende Methode zur Verfügung stellen, mit dieser eine neue Instanz des Delegates erzeugen und diese dann an `GetSubFolderSizes` übergeben. Alternativ kann der Aufrufer auch einfach `null` übergeben, wenn er die Fortschrittsmeldung nicht wünscht.

`GetSubFolderSizes` macht dann nicht mehr allzu viel. Zunächst werden die Ergebnis-Auflistung und ein `DirectoryInfo`-Objekt für den Ordner erzeugt. Dann ruft diese Methode den Fortschritt-Delegate zunächst für den übergeordneten Ordner auf, ermittelt die Gesamtgröße der in diesem Ordner gespeicherten Dateien und geht schließlich alle Unterordner durch, wobei deren Größe über die Methode `SumFolderSize` ermittelt wird. Für jeden Unterordner wird natürlich wieder der Fortschritt-Delegate aufgerufen. Alle ermittelten Ordnergrößen werden in der `FolderSize`-Auflistung gespeichert, die schließlich von `GetSubFolderSizes` zurückgegeben wird.

```
public static FolderSizes GetSubFolderSizes(string folderName,
    FolderProgress progress)
{
    // FolderSizes-Objekt für die Speicherung der Ordnergrößen erzeugen
    FolderSizes fs = new FolderSizes();

    // DirectoryInfo-Objekt für den Ordner erzeugen
    DirectoryInfo folder = new DirectoryInfo(folderName);

    // Fortschritts-Delegate aufrufen
    if (progress != null)
        progress(folder.Name);

    // Gesamtgröße der Dateien des Ordners ermitteln
    FileInfo[] files = folder.GetFiles();
    long folderSize = 0;
    for (int i = 0; i < files.Length; i++)
        folderSize += files[i].Length;

    // Ordnername und Größe als erstes Element in der Auflistung ablegen
    fs.Add(new FolderSize(folderName, folderSize));

    // Unterordner ermitteln
    DirectoryInfo[] subFolders = folder.GetDirectories();

    // Unterordner durchgehen
    for (int i = 0; i < subFolders.Length; i++)
```

Listing 176: Methode zum Ermitteln der Größe aller direkten Unterordner eines Ordners

```

    {
        // Fortschritts-Delegate aufrufen
        if (progress != null)
            progress(subFolders[i].Name);

        // Die jeweilige Größe ermitteln
        folderSize = SumFolderSize(subFolders[i]);

        // Ordnername und Größe in der Auflistung ablegen
        fs.Add(new FolderSize(subFolders[i].FullName, folderSize));
    }

    // Referenz auf die Auflistung zurückgeben
    return fs;
}

/* Methode zur Ermittlung der Gesamtgröße der in einem Ordner und dessen
 * Unterordnern enthaltenen Dateien */
public static long GetFolderSize(string folderName)
{
    // DirectoryInfo-Objekt für den Ordner erzeugen
    DirectoryInfo di = new DirectoryInfo(folderName);

    // Die rekursive Methode zur Ermittlung der Größe eines Ordners aufrufen
    return SumFolderSize(di);
}

/* Methode zum rekursiven Durchgehen eines Ordners und zur Ermittlung der
 * Gesamtgröße der in einem Ordner und dessen Unterordnern enthaltenen
 * Dateien. */
private static long SumFolderSize(DirectoryInfo folder)
{
    // Variable zur Speicherung der jeweils ermittelten Dateigrößen-Summe
    // eines (Unter-)Ordners
    long folderSize = 0;

    // Summe der Größen der Dateien im Ordner ermitteln
    FileInfo[] files = folder.GetFiles();
    for (int i = 0; i < files.Length; i++)
        folderSize += files[i].Length;

    // Unterordner ermitteln und die Methode rekursiv für jeden Unterordner
    // aufrufen und dabei die Variable folderSize hochzählen
    DirectoryInfo[] subFolders = folder.GetDirectories();
    for (int i = 0; i < subFolders.Length; i++)
        folderSize += SumFolderSize(subFolders[i]);

    // Die ermittelte Ordnergröße zurückgeben
    return folderSize;
}

```

Listing 176: Methode zum Ermitteln der Größe aller direkten Unterordner eines Ordners (Forts.)

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Listing 177 zeigt eine Beispielanwendung der `GetSubFolderSizes`-Methode. Diese Konsolenanwendung ermittelt die Größen der Unterordner von `C:\Inetpub`. Der Fortschritt wird über die Delegate-Methode `OnProgress` an der Konsole ausgegeben. Das Programm benötigt den Import der Namensräume `System` und `System.IO`.

```

/* Methode für den Delegate */
public static void OnProgress(string folderName)
{
    Console.WriteLine("Durchsuche {0} ...", folderName);
}

[STAThread]
static void Main(string[] args)
{
    // Ordner festlegen
    string folderName = "C:\\Inetpub";

    try
    {
        // Rekursives Ermitteln der Größe der im Ordner direkt und in allen
        // Unterordnern gespeicherten Dateien. Übergeben werden der Ordnername
        // und eine neue Instanz des Fortschritts-Delegates
        FolderSizes fs = GetSubFolderSizes(folderName,
            new FolderProgress(OnProgress));

        // Die Auflistung der Ordnergrößen durchgehen
        Console.WriteLine("\r\nErmittelte Größen:");
        for (int i = 0; i < fs.Count; i++)
            Console.WriteLine("{0}: {1:#,##0} Byte",
                fs[i].FolderName, fs[i].Size);
    }
    catch (IOException ex)
    {
        Console.WriteLine("Fehler: " + ex.Message);
    }
}

```

Listing 177: Anwendung der Methode zum Ermitteln der Größen aller Unterordner eines Ordners

Das Ergebnis dieses Programms zeigt Abbildung 62.



Abbildung 62: Das Beispielprogramm zur Ermittlung von Ordnergrößen in Aktion

100 Überprüfen, ob eine Datei existiert

Wenn Sie überprüfen wollen, ob eine Datei existiert, können Sie ähnlich der Überprüfung der Existenz eines Ordners (Rezept 90) wieder zwischen zwei Varianten wählen. Zum einen können Sie die statische Methode `Exists` der `File`-Klasse (aus dem Namensraum `System.IO`) einsetzen:

```

string fileName = "c:\\autoexec.bat";
if (File.Exists(fileName))
    Console.WriteLine(fileName + " existiert");
else
    Console.WriteLine(fileName + " existiert nicht");

```

Die zweite Variante setzt eine Instanz der Klasse `FileInfo` (ebenfalls aus dem Namensraum `System.IO`) ein und fragt deren `Exists`-Methode ab:

```

string fileName = "c:\\autoexec.bat";
FileInfo fi = new FileInfo(fileName);
if (fi.Exists)
    Console.WriteLine(fileName + " existiert");
else
    Console.WriteLine(fileName + " existiert nicht");

```

Beide Varianten können Sie auch ohne Probleme im Netzwerk einsetzen. Voraussetzung dafür ist natürlich, dass der aktuell in Windows eingeloggte Benutzer Leserechte für die Netzwerk-Ressource besitzt.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

101 Programmdateien in den Systempfaden suchen

Programme, die im Windows-, im System-Ordner oder in einem der Ordner gespeichert sind, die in der Umgebungsvariable `Path` eingetragen sind, können Sie zum Starten üblicherweise ohne Pfad angeben. Windows sucht automatisch in diesen Ordnern nach der Programmdatei.

Wenn Sie ermitteln wollen, ob ein zu startendes Programm in einem der Systempfad-Ordner gespeichert ist, können Sie die folgende Methode `FindFileInSystemPaths` einsetzen, die eine Datei im System-, im Windows- und in den in `Path` angegebenen Ordnern sucht.

Der Windows-Ordner wird dabei über die API-Funktion `GetWindowsDirectory` ermittelt, da das .NET Framework dazu keine direkte Möglichkeit bietet. Der Inhalt der `Path`-Variable wird über die Methode `GetEnvironmentVariable` der Klasse `System.Environment` ausgelesen. `FindFileInSystemPaths` sucht die angegebene Datei in der Reihenfolge, die auch Windows einsetzt (Systemordner – Windows-Ordner – Path-Pfade).

`FindFileInSystemPaths` erfordert den Import der Namensräume `System`, `System.Text`, `System.IO` und `System.Runtime.InteropServices`.

```

/* Deklaration der API-Funktion GetWindowsDirectory */
[DllImport("kernel32.dll", SetLastError=true)]
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,
    uint uSize);

/* Methode zur Suche einer Datei in den Systempfaden */
public static string FindFileInSystemPaths(string fileName)
{
    string path = null;

    // Im Windows-Systemordner suchen
    path = Path.Combine(Environment.SystemDirectory, fileName);
    if (File.Exists(path))
        return path;

    // Im Windows-Ordner suchen
    const int MAX_PATH = 160;
    string windowsDirectoryName = null;
    StringBuilder buffer = new StringBuilder(MAX_PATH + 1);
    if (GetWindowsDirectory(buffer, 260) > 0)
    {
        windowsDirectoryName = buffer.ToString();
        path = Path.Combine(windowsDirectoryName, fileName);
        if (File.Exists(path))
            return path;
    }

    // In den Ordnern suchen, die in der Umgebungsvariablen Path eingestellt
    // sind

```

Listing 178: Methode zum Suchen von Dateien in den Systempfaden

```
string[] systemPaths = Environment.GetEnvironmentVariable(
    "path").Split(new char[] {';'});
for (int i = 0; i < systemPaths.Length; i++)
{
    path = Path.Combine(systemPaths[i].Trim(), fileName);
    if (File.Exists(path))
        return path;
}

return null;
}
```

Listing 178: Methode zum Suchen von Dateien in den Systempfaden (Forts.)

Listing 179 zeigt eine Beispielanwendung der `FindFileInSystemPaths`-Methode.

```
// Suchen der Datei notepad.exe
string fileName = FindFileInSystemPaths("notepad.exe");
if (fileName != null)
    Console.WriteLine("Gefunden: {0}", fileName);
else
    Console.WriteLine("Nicht gefunden");

// Suchen der Datei java.exe
fileName = FindFileInSystemPaths("java.exe");
if (fileName != null)
    Console.WriteLine("Gefunden: {0}", fileName);
else
    Console.WriteLine("Nicht gefunden");
```

Listing 179: Suchen der Dateien Notepad.exe und Java.exe in den Systempfaden

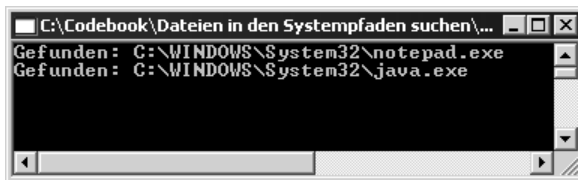


Abbildung 63: Das Beispielprogramm hat die Dateien gefunden

102 Dateien kopieren

Dateien können Sie sehr einfach über die `Copy`-Methode der `File`-Klasse kopieren. Am ersten Argument übergeben Sie dieser Methode den vollen Namen der Quelldatei, am zweiten den der Zieldatei. Über das dritte Argument können Sie festlegen, ob eine vorhandene Datei überschrieben werden soll. Tragen Sie hier `true` ein, wird eine eventuell vorhandene Datei ohne weitere Warnung überschrieben.

Tritt beim Kopieren ein Fehler auf, z.B. weil die Quelldatei gerade geöffnet oder die Zieldatei vorhanden und schreibgeschützt ist, erzeugt `Copy` eine Ausnahme vom Typ `IOException`, die Sie abfangen sollten.

Listing 180 zeigt das Kopieren einer Datei am Beispiel. Aufgrund der Verwendung der `MessageBox`-Klasse erfordert dieses Beispiel die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import des Namensraums `System.Windows.Forms`. Daneben müssen Sie die Namensräume `System` und `System.IO` einbinden.

```

/* Variablen für die Argumente der Copy-Methode */
string sourceFileName = @"C:\boot.ini";
string destFileName = @"C:\boot.ini.copy";
bool overwrite = true;

try
{
    /* Datei kopieren */
    File.Copy(sourceFileName, destFileName, overwrite);
}
catch (IOException ex)
{
    MessageBox.Show("Die Datei kann nicht kopiert werden: " + ex.Message,
        Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Exclamation);
}

```

Listing 180: Einfaches Kopieren einer Datei

Wenn Sie am dritten Argument der `Copy`-Methode `false` eintragen, erzeugt diese Methode eine Ausnahme vom Typ `IOException`, falls eine Datei mit identischem Namen bereits existiert. Da diese Ausnahme auch in anderen Fällen generiert wird, können Sie dem Anwender für den Fall, dass eine Datei gleichen Namens bereits vorhanden ist, leider nur eine allgemeine Meldung übergeben. Um etwas flexibler zu sein und den Anwender zu fragen, ob eine vorhandene Datei überschrieben werden soll, können Sie vor dem Kopieren überprüfen, ob die Datei existiert. Die Funktion `CopyFile` in Listing 181 übernimmt diese Aufgabe und wertet daneben auch eventuelle Ausnahmen aus.

Die Abfrage nach einer eventuell existierenden Datei gleichen Namens erfolgt über die `Exists`-Methode der `File`-Klasse. Im positiven Fall fragt `CopyFile` den Anwender über die `MessageBox`-Klasse, ob er die Datei überschreiben will. Wählt der Anwender den Nein-Schalter (was in der Rückgabe von `DialogResult.No` resultiert), wird die Funktion mit der Rückgabe von `false` beendet. Ansonsten kopiert `CopyFile` die Datei über die `Copy`-Methode der `File`-Klasse, wobei am dritten Argument `true` übergeben wird, damit eine eventuell vorhandene Datei gleichen Namens überschrieben wird.

Konnte die Datei kopiert werden, liefert `CopyFile` `true` zurück, ansonsten `false`.

Wie bereits beim Programm in Listing 180 müssen Sie zum Kompilieren der `CopyFile`-Methode die *Assembly `System.Windows.Forms.dll`* referenzieren und die Namensräume `System.Windows.Forms` und `System.IO` importieren.

```
public static bool CopyFile(string sourceFileName, string destFileName)
{
    // Abfragen, ob die Datei existiert
    if (System.IO.File.Exists(destFileName))
    {
        // Den Anwender fragen, ob die Datei überschrieben werden soll
        if (MessageBox.Show("Die Datei '" + destFileName +
            "' existiert bereits.\r\n\r\nSoll diese Datei überschrieben werden?",
            Application.ProductName, MessageBoxButtons.YesNo,
            MessageBoxIcon.Exclamation) == DialogResult.No)

            // Funktion beenden und false zurückgeben
            return false;
    }

    try
    {
        // Datei kopieren
        System.IO.File.Copy(sourceFileName, destFileName, true);

        return true;
    }
    catch (Exception ex)
    {
        // Fehlermeldung ausgeben
        MessageBox.Show("Die Datei '" + sourceFileName + "' kann nicht " +
            "kopiert werden: " + ex.Message, Application.ProductName,
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

        return false;
    }
}
```

Listing 181: Erweiterte Methode zum Kopieren einer Datei

103 Dateien umbenennen

Dateien können Sie sehr einfach über die `Move`-Methode der `File`-Klasse aus dem Namensraum `System.IO` umbenennen. Diese Methode generiert eine Ausnahme vom Typ `IOException`, wenn das Umbenennen nicht möglich ist. Der folgende Programmcode, der den Import der Namensräume `System` und `System.IO` voraussetzt, nennt eine Datei `c:\autoexec.bat` in `c:\autoexec.bat.bak` um und fängt mögliche Ausnahmen ab:

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
string sourceFileName = @"c:\autoexec.bat.copy";
string destFileName = @"c:\autoexec.bat.bak";

try
{
    // Datei umbenennen
    File.Move(sourceFileName, destFileName);

    Console.WriteLine("Datei erfolgreich umbenannt");
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Umbenennen der Datei: " + ex.Message);
}
```

Listing 182: Umbenennen einer Datei

104 Dateien verschieben

Das Verschieben einer Datei entspricht dem Umbenennen aus Rezept 103, nur dass Sie hier einen anderen Zielordner angeben. Das Programm benötigt den Import der Namensräume `System` und `System.IO`.

```
string sourceFileName = @"C:\autoexec.bat.copy";
string destFileName = @"C:\Temp\autoexec.bat.copy";

try
{
    // Datei verschieben
    File.Move(sourceFileName, destFileName);

    Console.WriteLine("Datei erfolgreich verschoben");
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Verschieben der Datei: " +
        ex.Message);
}
```

Listing 183: Verschieben einer Datei

105 Dateien löschen

Dateien können Sie über die Methode `Delete` der Klasse `System.IO.File` dauerhaft löschen:

```
string fileName = @"C:\boot.ini.copy";
System.IO.File.Delete(fileName);
```

Da beim Löschen Fehler auftreten können, die in einer Ausnahme vom Typ `IOException` resultieren, sollten Sie diese abfangen. Die Methode `DeleteFile` in Listing 184 setzt dies um. Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.Windows.Forms` und `System.IO` importieren.

```
public static bool DeleteFile(string fileName)
{
    try
    {
        // Datei löschen
        System.IO.File.Delete(fileName);
    }
    catch (Exception ex)
    {
        // Fehlermeldung ausgeben
        MessageBox.Show("Die Datei '" + fileName + "' kann nicht " +
            "gelöscht werden: " + ex.Message, Application.ProductName,
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

        // Misserfolg zurückmelden
        return false;
    }

    // Erfolg zurückmelden
    return true;
}
```

Listing 184: Methode zum Löschen einer Datei mit Fehlermeldung

Hinweis

So gelöschte Dateien werden leider nicht in den Papierkorb verschoben, sondern dauerhaft gelöscht. Eine Lösung zum Verschieben von Dateien und Ordner finden Sie im Rezept 115.

106 Dateien vergleichen

Für den Vergleich zweier Dateien bietet das .NET Framework keinen Support. Wenn Sie zwei Dateien daraufhin vergleichen wollen, ob diese identisch sind, müssen Sie eine eigene Vergleichsmethode programmieren. Sie können natürlich auch einfach die Methode `CompareFiles` aus Listing 185 verwenden ☺. Diese Methode erwartet neben den Dateinamen der zu vergleichenden Dateien am dritten Argument einen Wert der in Listing 185 deklarierten Aufzählung `FileCompareMethod`. Über dieses Argument können Sie bestimmen, ob nur das Datum der letzten Änderung (`FileCompareMethod.Date`) oder nur der Inhalt (`FileCompareMethod.Content`) oder beides verglichen wird (`FileCompareMethod.DateAndContent`).

`CompareFiles` überprüft zunächst, ob die Dateinamen identisch sind, und beendet sich mit `true`, wenn dies der Fall ist. Danach vergleicht `CompareFiles` über zwei `FileInfo`-Objekte das Datum der letzten Änderung der Dateien, wenn am Argument `compareMethod` ein entspre-

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

chender Wert übergeben wurde. Ist dieses Datum unterschiedlich, kehrt `CompareFiles` mit der Rückgabe von `false` zurück. Das Datum der Erstellung der Datei, das Sie aus einem `FileInfo`-Objekt ebenfalls auslesen können, wird übrigens nicht verglichen. Der Grund dafür liegt darin, dass Windows (XP, 2000) beim Kopieren einer Datei von einem Ordner in einen anderen dieses Datum auf das aktuelle setzt, was den Vergleich der Erstelldaten untauglich macht.

Zum Vergleich des Inhalts öffnet `CompareFiles` die beiden Dateien über `FileStream`-Objekte und vergleicht direkt die Größe der Dateien. Ist diese unterschiedlich, sind die Dateien ungleich und die Methode wird mit der Rückgabe von `false` beendet. Damit ist sichergestellt, dass der dem Größenvergleich folgende zeitaufwändige byteweise Vergleich nur dann ausgeführt werden muss, wenn die Dateien dieselbe Größe besitzen.

Ist die Größe der Dateien identisch, liest `CompareFiles` die Streams in einer Schleife byteweise ein und vergleicht jedes Byte mit dem der anderen Datei. Die dazu verwendete Methode `ReadByte` gibt einen `int`-Wert zurück, der mit `-1` aussagt, dass kein weiteres Byte mehr gelesen werden konnte. Im Schleifenfuß überprüft `CompareFiles` deswegen die Gleichheit der Werte und ob das aktuell gelesene Byte der ersten Datei `-1` ist.

Nach dem Schließen der Streams wird dann nur noch überprüft, ob die beiden zuletzt ermittelten `int`-Werte gleich sind, und das Ergebnis dieses Vergleichs zurückgegeben.

`CompareFiles` erfordert die Einbindung der Namensräume `System` und `System.IO`.

```

/* Aufzählung für die möglichen Vergleichstypen */
public enum FileCompareMethod
{
    Date,
    Content,
    DateAndContent
}

/* Methode zum Vergleichen von zwei Dateien */
public static bool CompareFiles(string fileName1, string fileName2,
    FileCompareMethod compareMethod)
{
    // Wenn beide Dateinamen identisch sind, true zurückgeben
    if (fileName1 == fileName2)
        return true;

    // Über FileInfo-Objekte das Datum der letzten Änderung vergleichen, sofern
    // dies gewünscht ist. Das Erstelldatum wird übrigens nicht verglichen,
    // weil dieses beim Kopieren von Dateien auf das aktuelle Datum gesetzt
    // wird
    if (compareMethod == FileCompareMethod.DateAndContent ||
        compareMethod == FileCompareMethod.Date)
    {
        FileInfo fi1 = new FileInfo(fileName1);
        FileInfo fi2 = new FileInfo(fileName2);

```

Listing 185: Methode zum Vergleich von zwei Dateien

```

        if (fi1.LastWriteTime != fi2.LastWriteTime)
            return false;
    }

    // Den Inhalt vergleichen, sofern dies gewünscht ist
    if (compareMethod == FileCompareMethod.DateAndContent ||
        compareMethod == FileCompareMethod.Content)
    {
        // FileStream-Objekte für den Vergleich erzeugen
        FileStream fs1 = new FileStream(fileName1, FileMode.Open);
        FileStream fs2 = new FileStream(fileName2, FileMode.Open);

        // Die Dateigröße vergleichen
        if (fs1.Length != fs2.Length)
            return false;

        // Die Dateien Byte für Byte vergleichen
        int fileByte1, fileByte2;
        do
        {
            fileByte1 = fs1.ReadByte();
            fileByte2 = fs2.ReadByte();
        } while (fileByte1 == fileByte2 && fileByte1 != -1);

        // Die Streams schließen
        fs1.Close();
        fs2.Close();

        // Das Ergebnis zurückgeben: Die Dateien sind gleich, wenn an dieser
        // Stelle die zuletzt gelesenen Bytes identisch sind
        return (fileByte1 == fileByte2);
    }

    return true;
}

```

Listing 185: Methode zum Vergleich von zwei Dateien (Forts.)

107 Ordner vergleichen

Das .NET Framework und das Windows-API bieten keine Möglichkeiten, zwei Ordner miteinander zu vergleichen. Ein solcher Vergleich muss einmal wieder selbst programmiert werden. Und das ist gar nicht so einfach.

Die Methode `CompareFolders` in Listing 186 implementiert den Vergleich zweier Ordner. Am ersten und zweiten Argument übergeben Sie die Ordnernamen, am dritten definieren Sie, ob die Methode den Inhalt der in den Ordnern enthaltenen Dateien vergleichen soll oder nur das Datum der letzten Änderung.

`CompareFolders` geht über die private Methode `CompareFolderContent` den am linken Argument angegebenen Ordner rekursiv durch und überprüft für den Ordner und jeden seiner Unterordner:

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

- ob die Anzahl der Unterordner im korrespondierenden anderen Ordner identisch ist,
- ob der korrespondierende andere Ordner dieselben Unterordner besitzt und
- ob der korrespondierende andere Ordner dieselben Dateien besitzt.

Die Methode `CompareFolderContent` übernimmt den rekursiven Vergleich. Dieser Methode werden je ein `DirectoryInfo`-Objekt für die aktuell zu vergleichenden Ordner und eine Info darüber übergeben, ob der Inhalt der Dateien im Ordner verglichen werden soll.

`CompareFolderContent` vergleicht zunächst über das `FileInfo`-Array, das die `GetFiles`-Methode der jeweiligen `DirectoryInfo`-Instanz zurückgibt, die Anzahl der Dateien in den beiden Ordnern und gibt `false` zurück, wenn diese unterschiedlich ist.

Danach geht `CompareFolderContent` alle Dateien des linken Ordners durch, ermittelt den korrespondierenden Dateinamen im rechten Ordner und überprüft, ob die Datei in diesem vorhanden ist. Wird eine nicht vorhandene Datei ermittelt, wird die Methode mit `false` beendet. Ist die Datei vorhanden, vergleicht `CompareFolderContent` je nach dem Argument `compareFileContent` entweder nur das Datum der letzten Änderung oder den Inhalt der Dateien. Zum Vergleich wird die Methode `CompareFiles` aus dem Rezept 106 eingesetzt. Das von dieser Methode erwartete Argument vom Typ `FileCompareMethod` wird dabei passend zum Wert des `CompareFolderContent`-Arguments eingestellt.

Die Anzahl der Unterordner und die Existenz der Unterordner des linken im rechten Ordner werden dann auf eine ähnliche Weise geprüft wie die der Dateien. Innerhalb der Schleife, die die Unterordner durchgeht, ruft `CompareFolderContent` sich nach der Überprüfung der Existenz des Unterordners dann für die jeweiligen Unterordner selbst auf um diese auf Gleichheit zu überprüfen. Die Rückgabe des rekursiven Aufrufs wird dabei ausgewertet und im Fall, dass die Methode `false` zurückgibt (weil die Unterordner nicht identisch sind), `return false` aufgerufen. Damit ist sichergestellt, dass der rekursive Aufruf möglichst schnell zurückkehrt. Am Ende gibt `CompareFolderContent` dann schließlich `true` zurück, weil an dieser Stelle, nach allen Überprüfungen, bewiesen ist, dass beide Ordner identisch sind.

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
public static bool CompareFolders(string folderName1, string folderName2,
    bool compareFileContent)
{
    // Die übergebenen Ordnernamen angleichen
    if (folderName1.EndsWith("\\") == false) folderName1 += "\\";
    if (folderName2.EndsWith("\\") == false) folderName2 += "\\";

    // Die rekursive Vergleichsmethode mit zwei neuen DirectoryInfo-
    // Objekten für die beiden Ordner aufrufen
    DirectoryInfo folder1 = new DirectoryInfo(folderName1);
    DirectoryInfo folder2 = new DirectoryInfo(folderName2);
    return CompareFolderContent(folder1, folder2, compareFileContent);
}
```

Listing 186: Methode zum Vergleichen zweier Ordner

```

}

/* Private Methode zum rekursiven Vergleich des Inhalts zweier Ordner */
private static bool CompareFolderContent(DirectoryInfo folder1,
    DirectoryInfo folder2, bool compareFileContent)
{
    // Die Anzahl der Dateien im Ordner vergleichen
    FileInfo[] files1 = folder1.GetFiles();
    FileInfo[] files2 = folder2.GetFiles();
    if (files1.Length != files2.Length)
        return false;

    // Die Dateien durchgehen und diese vergleichen
    foreach (FileInfo file1 in files1)
    {
        // Dateiname im zweiten Ordner ermitteln und überprüfen, ob die Datei
        // existiert
        string fileName2 = Path.Combine(folder2.FullName, file1.Name);
        if (System.IO.File.Exists(fileName2))
        {
            // Dateien vergleichen
            FileCompareMethod compareMethod =
                compareFileContent ? FileCompareMethod.Content :
                FileCompareMethod.Date;

            if (CompareFiles(file1.FullName, fileName2,
                compareMethod) == false)
                return false;
        }
        else
        {
            // Datei existiert nicht, also sind die Ordner nicht identisch
            return false;
        }
    }

    // Die Anzahl der Unterordner vergleichen
    DirectoryInfo[] subFolders1 = folder1.GetDirectories();
    DirectoryInfo[] subFolders2 = folder2.GetDirectories();
    if (subFolders1.Length != subFolders2.Length)
        return false;

    // Die Unterordner des ersten Ordners durchgehen um zu vergleichen, ob der
    // zweite Ordner dieselben Unterordner besitzt und um diese rekursiv
    // durchzugehen
    foreach (DirectoryInfo subFolder1 in subFolders1)
    {
        // Ordnername des zweiten Ordners ermitteln und überprüfen, ob der
        // Ordner existiert
        string folderName2 = Path.Combine(folder2.FullName, subFolder1.Name);
        DirectoryInfo subFolder2 = new DirectoryInfo(folderName2);
        if (subFolder2.Exists == false)
            return false;
    }
}

```

Listing 186: Methode zum Vergleichen zweier Ordner (Forts.)

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateien

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    // Rekursiver Aufruf zum Vergleich der beiden Unterordner
    if (CompareFolderContent(subFolder1, subFolder2,
        compareFileContent) == false)
        return false;
}

// Wenn die Methode hier ankommt, enthalten beide (Unter-)Ordner einen
// identischen Inhalt
return true;
}

```

Listing 186: Methode zum Vergleichen zweier Ordner (Forts.)

108 Die Größe einer Datei ermitteln

Zur Ermittlung der Größe einer Datei können Sie ein `FileInfo`-Objekt verwenden und dessen `Length`-Eigenschaft auslesen.

Das folgende Beispiel setzt ein solches Objekt ein und integriert eine Ausnahmebehandlung für den Fall, dass die Datei nicht existiert oder dass die Größen-Information nicht gelesen werden kann. Das Beispiel erfordert den Import der Namensräume `System` und `System.IO`.

```

string fileName = "c:\\pagefile.sys";

/* Ermitteln der Größe der Datei */
try
{
    // Größe über ein FileInfo-Objekt auslesen
    FileInfo fi = new FileInfo(fileName);
    long fileSize = fi.Length;

    Console.WriteLine("{0}: {1:#,#0} Byte.", fileName, fileSize);
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Ermitteln der Dateigröße " +
        "der Datei '{0}': {1}.", fileName, ex.Message);
}

```

Listing 187: Ermitteln der Größe einer Datei

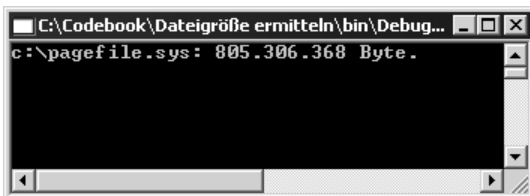


Abbildung 64: Ermittlung der Dateigröße der Datei `c:\pagefile.sys`

109 Dateiattribute auslesen

Über die Klasse `FileInfo` (aus dem Namensraum `System.IO`) erhalten Sie neben der Größe einer Datei auch Informationen über die Dateiattribute. Das folgende Beispiel ist ähnlich dem vorhergehenden Rezept programmiert. Das Programm ermittelt die Attribute der Datei über die Eigenschaft `Attributes`.

`Attributes` ist ein Bitfeld, das aus den Konstanten der Aufzählung (Enumeration) `FileAttributes` zusammengesetzt sein kann. Das Programm, das den Import der Namensräume `System` und `System.IO` voraussetzt, vergleicht die ermittelten Attribute mit den für Dateien möglichen Konstanten:

```
string fileName = "c:\\pagefile.sys";

/* Ermitteln der Attribute der Datei */
try
{
    // FileInfo-Objekt erzeugen
    FileInfo fi = new FileInfo(fileName);

    // Attribute auslesen
    Console.WriteLine("Dateiattribute von {0}", fileName);

    if ((fi.Attributes & FileAttributes.Archive) > 0)
        Console.WriteLine("Archiv");

    if ((fi.Attributes & FileAttributes.Compressed) > 0)
        Console.WriteLine("Komprimiert");

    if ((fi.Attributes & FileAttributes.Encrypted) > 0)
        Console.WriteLine("Verschlüsselt");

    if ((fi.Attributes & FileAttributes.Hidden) > 0)
        Console.WriteLine("Versteckt");

    if ((fi.Attributes & FileAttributes.Normal) > 0)
        Console.WriteLine("Normale Datei");

    if ((fi.Attributes & FileAttributes.
        NotContentIndexed) > 0)
        Console.WriteLine("Nicht inhalts-indiziert");

    if ((fi.Attributes & FileAttributes.Offline) > 0)
        Console.WriteLine("Offline");

    if ((fi.Attributes & FileAttributes.ReadOnly) > 0)
        Console.WriteLine("Schreibgeschützt");

    if ((fi.Attributes & FileAttributes.ReparsePoint) > 0)
```

Listing 188: Auslesen der Attribute einer Datei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    Console.WriteLine("Datei enthält einen Analysepunkt");

    if ((fi.Attributes & FileAttributes.SparseFile) > 0)
        Console.WriteLine("Dünn besetzte Datei (mit vielen 0-Bytes)");

    if ((fi.Attributes & FileAttributes.System) > 0)
        Console.WriteLine("System");

    if ((fi.Attributes & FileAttributes.Temporary) > 0)
        Console.WriteLine("Temporär");
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Ermitteln der Dateiattribute " +
        "der Datei '{0}': {1}.", fileName, ex.Message);
}

```

Listing 188: Auslesen der Attribute einer Datei (Forts.)

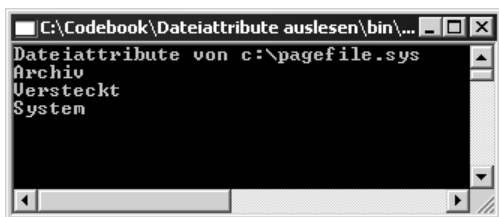


Abbildung 65: Ermittlung der Dateiattribute der Datei c:\pagefile.sys

110 Die Version einer Datei auslesen

Unter Windows werden alle Dateien mit Versionsinformationen versehen, die Sie z.B. im Explorer in den Eigenschaften einer Datei auslesen können. Die Version besteht aus vier 16 Bit großen Zahlen, die intern zu einem 64-Bit-Wert verknüpft werden.

Die erste 16-Bit-Zahl bestimmt die Haupt- (Major) und die zweite die Nebenversion (Minor). Haupt- und Nebenversion identifiziert nach der Microsoft-Konvention (wenigstens für Assemblies, siehe in der Dokumentation der `Version`-Klasse) große Änderungen in einer Datei (hauptsächlich in einem Programm oder einer Klassenbibliothek o.Ä.). Änderungen in der Hauptversion stehen dafür, dass eine Datei nicht mehr zu älteren Versionen kompatibel ist, Änderungen in der Nebenversion deuten auf eine eventuell noch vorhandene Rückwärts-Kompatibilität hin.

Die dritte Zahl steht für die Build-Nummer. Änderungen in dieser Nummer stehen für eine Neukompilierung desselben Programmcodes, lediglich mit geänderten Compilereinstellungen. Eine Erhöhung der Build-Nummer sollte also keine Inkompatibilitäten bewirken.

Die letzte Zahl verwaltet die Revisions-Nummer. Dateien mit unterschiedlichen Revisionsnummern sollten vollständig austauschbar sein. Diese Nummer ist hauptsächlich für Bugfixes und für die Beseitigung von Sicherheitsmängeln vorgesehen.

Um die Versionsinformationen einer Datei zu ermitteln, können Sie eine Instanz der Klasse `FileVersionInfo` aus dem Namensraum `System.Diagnostics` verwenden. Die statische Methode `GetVersionInfo` dieser Klasse gibt unter der Übergabe eines Dateinamens eine `FileVersionInfo`-Instanz zurück. Über verschiedene Eigenschaften erhalten Sie Informationen zur Version, wie es Listing 189 zeigt. Das Programm erfordert den Import der Namensräume `System`, `System.IO` und `System.Diagnostics`.

```
// Vollen Dateinamen der Datei kernel32.dll ermitteln
string fileName = Path.Combine(Environment.SystemDirectory,
    "kernel32.dll");

// Versionsinformationen auslesen
FileVersionInfo fileVersionInfo = FileVersionInfo.GetVersionInfo(fileName);

// Version ausgeben
Console.WriteLine("Version der Datei {0}", fileName);
Console.WriteLine("Major: {0}", fileVersionInfo.FileMajorPart);
Console.WriteLine("Minor: {0}", fileVersionInfo.FileMinorPart);
Console.WriteLine("Build: {0}", fileVersionInfo.FileBuildPart);
Console.WriteLine("Revision: {0}", fileVersionInfo.FilePrivatePart);
Console.WriteLine("Version: {0}", fileVersionInfo.FileVersion);
```

Listing 189: Auslesen der Versionsinformationen einer Datei

Die Ausgabe des Programms zeigt Abbildung 66.

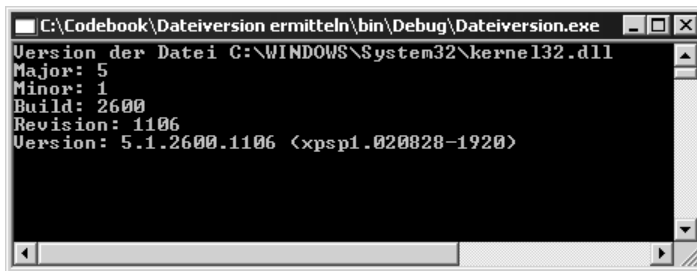


Abbildung 66: Das Beispielprogramm zum Auslesen von Versionsinformationen in Aktion

111 Den kurzen (8.3-)Dateinamen einer Datei auslesen

Windows verwaltet für jede Datei neben dem normalen (langen) Dateinamen einen kurzen, zu alten DOS-Programmen kompatiblen Dateinamen. Dieser besteht aus einem acht Zeichen großen Namen und einer dreistelligen Endung. Der Name wird am Ende automatisch mit einer Tilde und einer Ziffer versehen, um eine Eindeutigkeit bei langen Dateinamen sicherzustellen, die mit denselben Zeichen beginnen.

Wenn Sie den kurzen Dateinamen einer Datei (inklusive Pfad) ermitteln wollen, weil Sie z.B. mit einem alten DOS-Programm kommunizieren müssen, finden Sie im .NET Framework

hierfür keine Möglichkeit. Dazu müssen Sie die Windows-API-Funktion `GetShortPathName` verwenden, der Sie den langen Dateinamen inklusive Pfad übergeben. `GetShortPathName` wird folgendermaßen deklariert:

```
[DllImport("kernel32.dll", SetLastError=true)]
private static extern int GetShortPathName(string lpzLongPath,
    System.Text.StringBuilder lpzShortPath, int cchBuffer);
```

Am Argument `lpzLongPath` übergeben Sie den langen Dateinamen. Das Argument `lpzShortPath` muss mit einem für Pfadangaben ausreichend groß dimensionierten `StringBuilder`-Objekt belegt werden. Im letzten Argument übergeben Sie die Größe dieses Objekts. `GetShortPathName` ermittelt an diesem Argument, wie viele Zeichen in den String geschrieben werden können.

Die Methode `GetShortName` in Listing 190 setzt diese Funktion zur Ermittlung des kurzen Dateinamens einer Datei ein. `GetShortName` ruft `GetShortPathName` mit dem übergebenen Dateinamen und einem `StringBuilder`-Objekt auf, das auf 261 Zeichen Größe initialisiert wurde (die maximale Pfadgröße plus Platz für das 0-Zeichen, mit dem die in C oder C++ geschriebene Funktion den String abschließt). Dazu ist anzumerken, dass `GetShortPathName` nicht nur den Dateinamen selbst, sondern auch den kompletten Pfad zurückgibt, wenn auch der Quelldateiname einen Pfad beinhaltet.

Im Erfolgsfall gibt `GetShortPathName` die Anzahl der gelesenen Zeichen zurück. In diesem Fall gibt `GetShortName` den ermittelten Dateinamen direkt zurück. Im Fehlerfall, bei dem `GetShortPathName` 0 zurückgibt, liest `GetShortName` den letzten Windows-Fehlercode aus und ermittelt mit diesem Code über `FormatMessage` eine Fehlerbeschreibung. Damit der Windows-Fehlercode überhaupt verfügbar ist, habe ich `GetShortPathName` mit `SetLastError=true` im `DllImport`-Attribut deklariert. Mit der so ermittelten (aussagekräftigen) Fehlerbeschreibung erzeugt `GetShortName` dann eine Ausnahme vom Typ `IOException`.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration der benötigten API-Funktionen und -Konstanten */
[DllImport("kernel32.dll", SetLastError=true)]
private static extern int GetShortPathName(string lpzLongPath,
    StringBuilder lpzShortPath, int cchBuffer);

[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, string lpSource,
    int dwMessageId, int dwLanguageId, StringBuilder lpBuffer, int nSize,
    string [] Arguments);

private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;
private const int MAX_PATH = 260;
```

Listing 190: Methode zum Auslesen des kurzen Dateinamens einer Datei

```

/* Methode zur Ermittlung des kurzen Dateinamens einer Datei */
public static string GetShortName(string fileName)
{
    // GetShortPathName mit einem ausreichend großen StringBuilder-Objekt
    // aufrufen
    StringBuilder shortName = new StringBuilder(MAX_PATH + 1);
    if (GetShortPathName(fileName, shortName, MAX_PATH + 1) > 0)
        return shortName.ToString();
    else
    {
        // Fehler bei der Ausführung: API-Fehler ermitteln, diesen in eine
        // Fehlerbeschreibung umwandeln und damit eine Ausnahme werfen
        System.Text.StringBuilder errorMessage = new StringBuilder(1024);
        FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, null,
            Marshal.GetLastWin32Error(), 0, errorMessage, 1024, null);
        throw new IOException(errorMessage.ToString());
    }
}

```

Listing 190: Methode zum Auslesen des kurzen Dateinamens einer Datei (Forts.)

Die Anwendung dieser Methode ist sehr einfach. Sie sollten natürlich eine Ausnahmebehandlung vorsehen, für den Fall, dass die Datei nicht existiert oder ein sonstiger Fehler auftritt. Abbildung 67 zeigt ein Beispielprogramm, das über `GetShortName` den kurzen Namen der Datei `license.txt` im Ordner des .NET Framework SDK ermittelt hat.

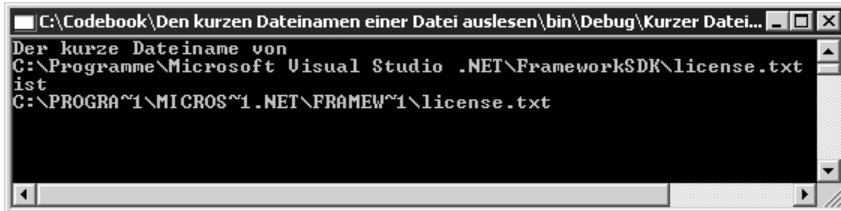


Abbildung 67: Ermittlung des kurzen Dateinamens der .NET Framework-Lizenzdatei

112 Alle Dateien eines Ordners auflisten

Um alle Dateien eines Ordners aufzulisten, können Sie ein `DirectoryInfo`-Objekt (aus dem Namensraum `System.IO`) verwenden und über dessen `GetFiles`-Methode Informationen über alle enthaltenen Dateien auslesen. `GetFiles` gibt ein Array mit `FileInfo`-Objekten zurück, das Sie sehr einfach durchgehen können. Aus der Eigenschaft `Name` der `FileInfo`-Objekte können Sie den Dateinamen auslesen. Für den Fall, dass ein ungültiger Ordner angegeben ist oder ein anderer Fehler auftritt, sollten Sie eine Ausnahmebehandlung vorsehen:

```
// Dateiname festlegen
string folderName = "c:\\\\";

try
{
    // DirectoryInfo-Objekt erzeugen
    DirectoryInfo di = new DirectoryInfo(folderName);

    // Dateien ermitteln
    FileInfo[] files = di.GetFiles();

    // Alle Dateien durchgehen und den Namen ausgeben
    for (int i = 0; i < files.Length; i++)
        Console.WriteLine(files[i].Name);
}
catch (IOException ex)
{
    Console.WriteLine("Fehler beim Ermitteln der Dateien: {0}.", ex.Message);
}
```

Listing 191: Ermitteln aller Dateien eines Ordners

113 Dateien suchen

Dateien können Sie recht einfach suchen, indem Sie ein `DirectoryInfo`-Objekt für den Ordner erzeugen, in dem die Suche beginnen soll, und dann dessen `GetFiles`-Methode den Namen der Datei oder ein Dateimuster (z.B. `*.txt`) übergeben. `GetFiles` gibt ein Array von `FileInfo`-Objekten für alle Dateien zurück, die dem übergebenen Namen bzw. Muster entsprechen. Wollen Sie ermöglichen, dass auch alle Unterordner durchsucht werden, müssen Sie eine rekursive Methode schreiben, die sich selbst für jeden Unterordner eines Ordners aufruft um in diesem nach Dateien zu suchen.

Die Methode `FindFiles` in Listing 192 setzt diese Idee um. `FindFiles` erwartet den Namen des Ordners, ab dem gesucht werden soll, am ersten Argument. Das zweite Argument bestimmt das Suchmuster. Hier können Sie Dateinamen einsetzen, die auch die unter Windows üblichen Wildcards (`*`, `?`) beinhalten können. Das Argument `recurse` bestimmt, ob die Methode auch in den Unterordnern des angegebenen Ordners suchen soll.

Da beim Suchen nach Dateimustern oder beim rekursiven Suchen auch mehrere Dateien gefunden werden können, gibt die Methode einer Instanz der Klasse `StringCollection` zurück, die die Namen der gefundenen Dateien (inklusive Pfad) verwaltet.

Die eigentliche Arbeit übernimmt die private Methode `FindFilesInFolder`, die in einem Ordner alle Dateien ermittelt, die dem Suchmuster entsprechen, aber zuvor sich selbst für alle Unterordner des aktuellen Ordners rekursiv aufruft. Die `StringCollection`-Instanz wird dabei einfach als Argument übergeben.

Um dem Anwender beim umfangreichen Suchen einen Fortschritt anzeigen zu können, arbeitet `FindFiles` mit einem Delegate namens `FindProgress`, der beim Aufruf am letzten Argument übergeben werden kann. Diesen Delegate gibt `FindFiles` an die Methode `Find-`

FilesInFolder weiter, die den Delegate für jeden Ordner unter der Übergabe des aktuellen Ordnersnamens aufruft.

FindFiles erfordert den Import der Namensräume System, System.IO und System.Collections.Specialized.

```
/* Delegate für eine Fortschrittsmeldung */
public delegate void FindProgress(string folderName);

/* Methode zum Suchen von Dateien in einem Ordner */
public static StringCollection FindFiles(string folderName,
    string searchPattern, bool recurse, FindProgress findProgress)
{
    // StringCollection-Objekt für die Rückgabe erzeugen
    StringCollection resultFiles = new StringCollection();

    // DirectoryInfo-Objekt für den Ordner erzeugen
    DirectoryInfo folder = new DirectoryInfo(folderName);

    // Die rekursive Methode zum Suchen in einem Ordner aufrufen
    FindFilesInFolder(searchPattern, folder, recurse, resultFiles,
        findProgress);

    // StringCollection zurückgeben
    return resultFiles;
}

/* Rekursive Methode zum Suchen von Dateien */
private static void FindFilesInFolder(string searchPattern,
    DirectoryInfo folder, bool recurse, StringCollection resultFiles,
    FindProgress findProgress)
{
    // Delegate aufrufen, falls dieser übergeben wurde
    if (findProgress != null)
        findProgress(folder.FullName);

    // Zunächst für alle Unterordner FindFilesInFolder rekursiv aufrufen,
    // wenn dies gewünscht ist
    if (recurse)
    {
        DirectoryInfo[] subFolders = folder.GetDirectories();
        for (int i = 0; i < subFolders.Length; i++)
            FindFilesInFolder(searchPattern, subFolders[i], true, resultFiles,
                findProgress);
    }

    // Alle Dateien ermitteln, die dem übergebenen Suchmuster entsprechen
    FileInfo[] files = folder.GetFiles(searchPattern);

    // Die gefundenen Dateien durchgehen und deren vollen Namen an die
```

Listing 192: Methode zum Suchen von Dateien

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateien

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// StringCollection anhängen
for (int i = 0; i < files.Length; i++)
    resultFiles.Add(files[i].FullName);
}
```

Listing 192: Methode zum Suchen von Dateien (Forts.)

Vor dem Aufruf der `FindFiles`-Methode müssen Sie zunächst eine dem Fortschritts-Delegate entsprechende Methode schreiben, wenn Sie eine Fortschrittsmeldung wünschen:

```
private static void FindProgressHandler(string folderName)
{
    Console.WriteLine("Durchsuche {0} ...", folderName);
}
```

Listing 193: Methode für den Fortschritts-Delegate

Beim Aufruf von `FindFiles` übergeben Sie dann neben dem Start-Ordernamen, dem Suchmuster und der Rekursions-Info eine neue Instanz dieses Delegates. Zur Auswertung der Suche gehen Sie dann die zurückgegebene `StringCollection`-Auflistung durch, wie ich dies im folgenden Beispiel programmiert habe:

```
/* Im Inetpub-Ordner rekursiv nach Textdateien suchen */
StringCollection files = FileUtil.FindFiles("c:\\inetpub", "*.txt", true,
    new FileUtil.FindProgress(FindProgressHandler));

if (files.Count > 0)
{
    Console.WriteLine("\r\nGefundene Dateien:");
    for (int i = 0; i < files.Count; i++)
        Console.WriteLine(files[i]);
}
else
{
    Console.WriteLine("Keine Dateien gefunden");
}
```

Listing 194: Beispiel-Anwendung der FindFiles-Methode

Das Ergebnis zeigt Abbildung 68.

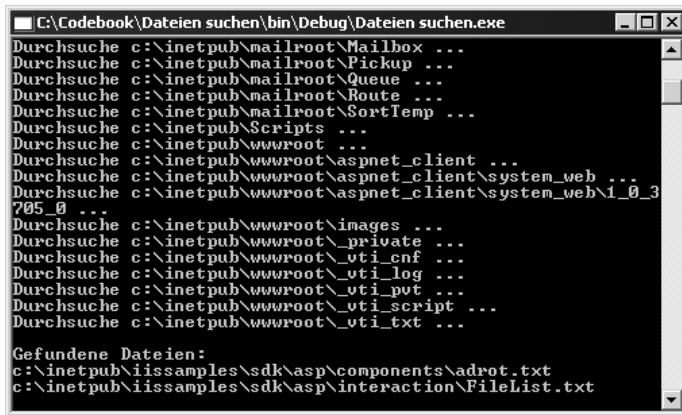


Abbildung 68: Ausgabe des FindFiles-Beispielprogramms

Wenn Sie keine Fortschrittsmeldung wünschen (was die Suche nicht unerheblich beschleunigt), übergeben Sie an Stelle des Delegates einfach `null`:

```
StringCollection files = File.FindFiles("c:\\windows", "*.txt", true, null);
```

114 Dateien und Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage umbenennen oder verschieben

Das Umbenennen und Verschieben einer Datei bzw. eines Ordners über die `Move`-Methode der `File`- bzw. der `Directory`-Klasse ermöglicht leider kein Überschreiben vorhandener Dateien. Wenn Sie dies dem Anwender ermöglichen wollen, müssen Sie die Windows-API-Funktion `SHFileOperation` verwenden, die ich bereits im Rezept 93 für das Kopieren von Ordnern beschrieben habe. Damit können Sie, wie bereits beim Kopieren, dem Anwender die Entscheidung überlassen, bereits vorhandene Dateien bzw. Ordner zu überschreiben (Abbildung 69), und zudem einen automatischen Fortschrittsdialog anzeigen lassen.

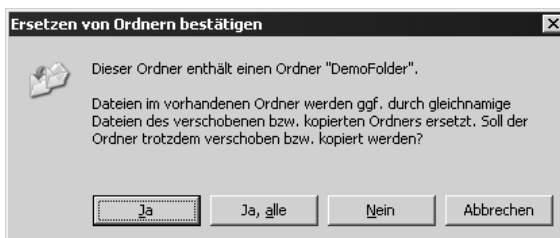


Abbildung 69: Der Überschreib-Bestätigungsdialog von `SHFileOperation` beim Verschieben eines Ordners

Die Methode `APIMove` in Listing 195 zum Verschieben einer Datei bzw. eines Ordners entspricht fast komplett der Methode `APICopy` zum Kopieren von Ordnern aus dem Rezept 93. Als einziger Unterschied wird als Funktion für `SHFileOperation` nun `FO_MOVE` im Feld `wFunc` angegeben.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```

/* Deklaration benötigter API-Funktionen, -Konstanten und -Strukturen */
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

private const int FO_MOVE = 0x0001;           // Verschieben
private const int FOF_ALLOWUNDO = 0x0040;    // Undo ermöglichen
private const int FOF_NOCONFIRMATION = 0x0010; // Keine Nachfrage

[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
    public IntPtr hwnd;
    public int wFunc;
    public string pFrom;
    public string pTo;
    public short fFlags;
    [MarshalAs(UnmanagedType.Bool)]
    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
}

/* Methode zum Verschieben */
public static bool APIMove(string source, string dest, bool confirmOverwrites)
{
    // Struktur für die Dateiinformationen erzeugen
    SHFILEOPSTRUCT fileOp = new SHFILEOPSTRUCT();

    // (Unter-)Funktion definieren (ShFileOperation kann auch Dateien und
    // Ordner löschen und kopieren)
    fileOp.wFunc = FO_MOVE;

    // Quelle und Ziel definieren. Dabei müssen mehrere Datei- oder
    // Ordnerangaben über 0-Zeichen getrennt werden.
    // Am Ende muss ein zusätzliches 0-Zeichen stehen
    fileOp.pFrom = source + "\x0\x0";
    fileOp.pTo = dest + "\x0\x0";

    // Flags setzen, sodass ein Rückgängigmachen möglich ist (was aber
    // beim Verschieben zurzeit scheinbar noch nicht unterstützt wird!)
    // und - je nach Argument confirmOverwrites - keine Nachfrage
    // beim Überschreiben von Ordnern beim Anwender erfolgt

```

Listing 195: Methode zum Verschieben einer Datei oder eines Ordners über `SHFileOperation`

```

    if (confirmOverwrites)
        fileOp.fFlags = FOF_ALLOWUNDO;
    else
        fileOp.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;

    // SHFileOperation unter Übergabe der Struktur aufrufen
    int result = SHFileOperation(ref fileOp);

    // Erfolg oder Misserfolg zurückgeben
    return (result == 0);
}

```

Listing 195: Methode zum Verschieben einer Datei oder eines Ordners über SHFileOperation (Forts.)

Bei der Anwendung dieser Klasse können Sie entweder Datei- oder Ordnerpfade übergeben. Das folgende Beispiel verschiebt den Ordner `c:\demofolder` (der für das Beispiel gegebenenfalls automatisch angelegt wird) nach `c:\temp\demofolder` und die Datei `c:\Demo for API-Move.txt` (die ebenfalls gegebenenfalls automatisch angelegt wird) nach `c:\temp\Demo for API-Move.txt`.

```

/* Ordner verschieben */
string sourceFolderName = @"c:\DemoFolder";    // Quellordner
string destFolderName = @"c:\Temp\DemoFolder"; // Zielordner

// Ordner als Beispiel zum Verschieben anlegen
System.IO.Directory.CreateDirectory(sourceFolderName);

// Ordner rekursiv mit Anwender-Nachfrage für das Überschreiben von
// Ordnern verschieben
Console.WriteLine("Verschiebe {0} nach {1} ... ", sourceFolderName,
    destFolderName);
if (APIMove(sourceFolderName, destFolderName, true))
    Console.WriteLine("Ordner erfolgreich verschoben");
else
    Console.WriteLine("Fehler beim Verschieben des Ordners");

/* Datei verschieben */
string sourceFileName = @"c:\Demo for API-Move.txt";    // Quelldatei
string destFileName = @"c:\Temp\Demo for API-Move.txt"; // Zieldatei

// (Text-)Datei als Beispiel zum Verschieben erzeugen
System.IO.StreamWriter sw = System.IO.File.CreateText(sourceFileName);
sw.WriteLine("Beispieldatei für das Verschieben einer Datei " +
    "über SHFileOperation.");
sw.Close();

// Datei mit Anwender-Nachfrage für das Überschreiben von Dateien verschieben
Console.WriteLine("Verschiebe {0} nach {1} ... ", sourceFileName, destFileName);

```

Listing 196: Anwendung der Methode APIMove zum komfortablen Verschieben von Dateien oder Ordnern

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
if (APIMove(sourceFileName, destFileName, true))  
    Console.WriteLine("Datei erfolgreich verschoben");  
else  
    Console.WriteLine("Fehler beim Verschieben der Datei");
```

Listing 196: Anwendung der Methode `APIMove` zum komfortablen Verschieben von Dateien oder Ordnern (Forts.)

Die Abbildungen 70 und 71 zeigen dieses Programm für den Fall, dass der Zielordner und die Zieldatei bereits existieren.



Abbildung 70: Verschieben eines Ordners, der im Ziel bereits existiert

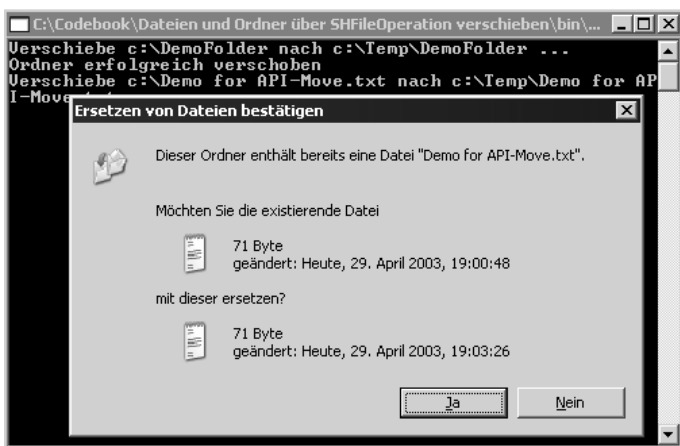


Abbildung 71: Verschieben einer Datei, die im Ziel bereits existiert

115 Dateien und Ordner in den Papierkorb verschieben

Das .NET Framework unterstützt das Verschieben von Dateien und Ordnern in den Papierkorb leider nicht direkt. Dazu müssen Sie die Windows-API-Funktion `SHFileOperation` einsetzen, die Dateien und Ordner kopieren, verschieben, umbenennen und (u.a. in den Papierkorb) löschen kann. Diese Funktion steht ab Windows 95 zur Verfügung.

`SHFileOperation` erwartet als Argument eine als Referenz zu übergebende, normale (C-) Struktur mit Informationen zu der auszuführenden Aktion. In dieser Struktur geben Sie die auszuführende Aktion und die Datei- bzw. Ordnernamen an. In einem »Flags«-Feld können Sie die Aktion näher spezifizieren. Für das Löschen einer Datei bzw. eines Ordners können Sie hier z.B. festlegen, dass der Vorgang rückgängig gemacht werden kann (womit die Datei bzw. der Ordner automatisch in den Papierkorb verschoben wird) und dass Windows beim Anwender nicht nachfragt, ob die Datei oder der Ordner wirklich verschoben werden soll.

Die Methode `MoveToRecycleBin` in Listing 197 implementiert das Verschieben in den Papierkorb über `SHFileOperation`. Als Argument erwartet diese Funktion lediglich den Pfad zu der Datei bzw. zum Ordner, die bzw. der gelöscht werden soll.

Das Prinzip dieser Methode entspricht dem, das ich bereits im Rezept 93 erläutert habe. Die wesentlichen Unterschiede zu diesem Rezept sind die folgenden:

Zunächst schreibt `MoveToRecycleBin` zur Definition der Unterfunktion von `SHFileOperation` die Konstante `FO_DELETE` in das Feld `wFunc` der `SHFILEOPSTRUCT`-Struktur (die an `SHFileOperation` übergeben wird), um zu definieren, dass die Datei bzw. der Ordner gelöscht werden soll. Das Feld `fFlags` wird mit der Konstante `FOF_ALLOWUNDO` beschrieben. Damit werden gelöschte Dateien bzw. Ordner automatisch in den Papierkorb verschoben. Da das Ziel in diesem Fall fest definiert ist, wird nur das Feld `pFrom` mit dem übergebenen Pfad definiert.

Aufgrund der Problematik bei der Auswertung von Fehlern bei der Verwendung von `SHFileOperation` (siehe Rezept 93) habe ich darauf verzichtet, im Fehlerfall eine Ausnahme zu erzeugen. Da das Flag `FOF_NOERRORUI` nicht gesetzt wird, werden Fehler von `SHFileOperation` automatisch in einem Dialog gemeldet. `MoveToRecycleBin` meldet über einen booleschen Wert lediglich zurück, ob das Löschen in den Papierkorb erfolgreich war oder nicht.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration benötigter API-Funktionen, -Konstanten und -Strukturen */
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

private const int FO_DELETE = 0x0003;           // Datei/Ordner löschen
private const int FOF_ALLOWUNDO = 0x0040;      // Undo ermöglichen
private const int FOF_NOCONFIRMATION = 0x0010; // keine Nachfrage

[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
```

Listing 197: Methode zum Verschieben von Dateien und Ordnern in den Papierkorb

```

public struct SHFILEOPSTRUCT
{
    public IntPtr hwnd;
    public int wFunc;
    public string pFrom;
    public string pTo;
    public short fFlags;
    [MarshalAs(UnmanagedType.Bool)]
    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
}

/* Methode zum Verschieben einer Datei in den Papierkorb */
public static bool MoveToRecycleBin(string path)
{
    // Struktur für die Dateiinformationen erzeugen
    SHFILEOPSTRUCT fileOp = new SHFILEOPSTRUCT();

    // Quelle definieren. Dabei müssen mehrere Datei- oder
    // Ordnerangaben über 0-Zeichen getrennt werden.
    // Am Ende muss ein zusätzliches 0-Zeichen stehen
    fileOp.pFrom = path + "\\x0\\x0";

    // Flags setzen, sodass ein Rückgängigmachen möglich ist und
    // keine Nachfrage beim Anwender erfolgt
    fileOp.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;

    // (Unter-)Funktion definieren
    fileOp.wFunc = FO_DELETE;

    // ShFileOperation unter Übergabe der Struktur aufrufen
    int result = SHFileOperation(ref fileOp);

    // Erfolg oder Fehler zurückmelden. SHFileOperation liefert 0
    // zurück, wenn kein Fehler aufgetreten ist
    return (result == 0);
}

```

Listing 197: Methode zum Verschieben von Dateien und Ordnern in den Papierkorb (Forts.)

Abbildung 72 zeigt eine Anwendung, die über `MoveToRecycleBin` die Datei `c:\autoexec.bat.copy` und den Ordner `c:\temp\RecycleBinDemoFolder` in den Papierkorb verschoben hat.

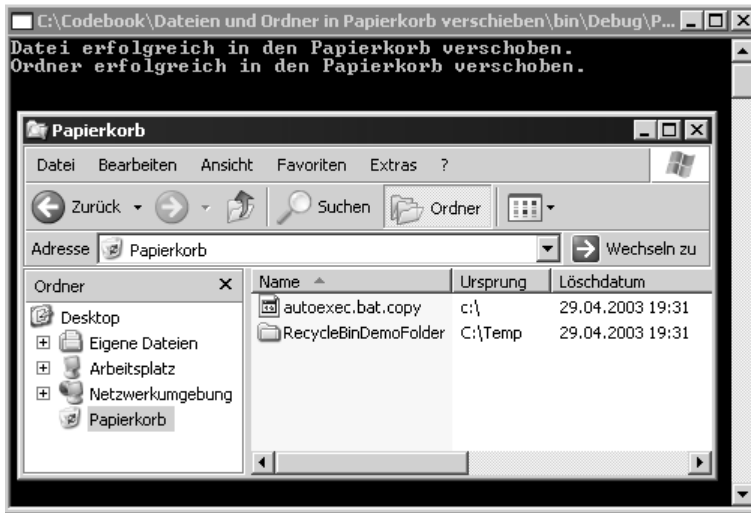


Abbildung 72: In-Den-Papierkorb-Löschen einer Datei und eines Ordners über die Methode `MoveToRecycleBin`

116 Größe des Papierkorbs und Anzahl der Dateien im Papierkorb ermitteln

Über die Windows-API-Funktion `SHQueryRecycleBin` können Sie ab Windows 95 die Größe des Papierkorbs und die Anzahl der enthaltenen Dateien eines Laufwerks oder aller Laufwerke ermitteln. `SHQueryRecycleBin` erwartet am ersten Argument den Wurzel-Pfad des Laufwerks, dessen Papierkorb erforscht werden soll, oder einen leeren String, wenn alle Papierkörbe berücksichtigt werden sollen (was, wie ich gleich noch beschreibe, unter Windows 2000 leider nicht funktioniert). Am zweiten Argument übergeben Sie eine Struktur vom Typ `SHQUERYRBINFO`, in der die Funktion die gelesenen Werte zurückgibt.

Die Methode `GetRecycleBinInfo` in Listing 198 implementiert das Lesen der Papierkorb-Daten. `GetRecycleBinInfo` erwartet einen String, der entweder leer oder `null` ist oder den Stammordner des Laufwerks bestimmt, dessen Papierkorb ausgelesen werden soll. Die Methode gibt eine Instanz der Klasse `RecycleBinInfo` mit den gelesenen Informationen zurück.

Unter Windows 2000 führt der Aufruf von `SHQueryRecycleBin` leider zu dem Fehler `E_INVALIDARG (0x80070057)`, wenn `null` oder ein leerer String übergeben wird. Dieser Bug ist im Microsoft Knowledge Base-Artikel Q266366 dokumentiert.

Um dieses Problem zu umschiffen, geht `GetRecycleBinInfo` bei der Übergabe eines Leerstrings oder von `null` alle Festplatten-Laufwerke des Systems durch und speichert deren Wurzelfade in der `ArrayList`-Instanz `drives`. Die hierbei verwendete Technik basiert auf dem Rezept 88. Wird ein nicht leerer String übergeben, speichert `GetRecycleBinInfo` lediglich diesen in `drives`.

Danach wird die `RecycleBinInfo`-Instanz für die Rückgabe und die Instanz der `SHQUERYRB-INFO`-Struktur erzeugt. Da `SHQueryRecycleBin` die Größe der Struktur im Feld `cbSize` erwartet (um diese beim Schreiben zu kennen), liest `GetRecycleBinInfo` die Größe über `Marshal.SizeOf` so aus, dass die tatsächlich an die externe Funktion übergebene Größe ermittelt wird.

Danach geht `GetRecycleBinInfo` alle ermittelten Laufwerke durch, liest über `SHQueryRecycleBin` die Informationen der einzelnen Papierkörbe und addiert die gelesenen Werte zu den bereits in der `RecycleBinInfo`-Instanz gespeicherten. Wenn nur die Daten des Papierkorbs eines Laufwerks ausgelesen werden sollen, beschränkt sich die Schleife natürlich auf einen Durchlauf. Auf diese Weise wird das Windows-2000-Problem aber mehr oder weniger elegant umgangen ☺.

`GetRecycleBinInfo` gibt die `RecycleBinInfo`-Instanz schließlich zurück. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Collections` und `System.Runtime.InteropServices` importieren.

```

/* Klasse für die Rückgabe der GetRecycleBinInfo-Methode */
public class RecycleBinInfo
{
    public long Size = 0;
    public long Count = 0;
}

/* Struktur, in der SHQueryRecycleBin die gelesenen Daten zurückgibt */
[StructLayout(LayoutKind.Sequential, Pack=1)]
private struct SHQUERYRBINFO
{
    public int cbSize;
    public long i64Size;
    public long i64NumItems;
}

/* Deklaration der benötigten API-Funktionen und -Konstanten */
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHQueryRecycleBin(string rootPath,
    ref SHQUERYRBINFO shQueryRBInfo);

[DllImport("Kernel32.dll")]
private static extern uint GetDriveType(string lpRootPathName);

private const uint DRIVE_FIXED = 3;

/* Methode zum Auslesen der Anzahl und der Größe der Dateien
 * im Papierkorb */
public static RecycleBinInfo GetRecycleBinInfo(string rootPath)
{
    // Die auszulesenden Laufwerke bestimmen
    ArrayList drives = new ArrayList();

```

Listing 198: Methode zum Ermitteln der Anzahl der Elemente im Papierkorb und der Gesamtgröße der enthaltenen Dateien


```

    if (rootPath != null && rootPath != "")
    {
        drives.Add(rootPath);
    }
    else
    {
        // Alle Festplattenpartitionen ermitteln
        string[] logicalDrives = Environment.GetLogicalDrives();
        for (int i = 0; i < logicalDrives.Length; i++)
            if (GetDriveType(logicalDrives[i]) == DRIVE_FIXED)
                drives.Add(logicalDrives[i]);
    }

    // RecycleBinInfo-Instanz für das Ergebnis und SHQUERYRBINFO-Struktur
    // erzeugen
    RecycleBinInfo rbi = new RecycleBinInfo();
    SHQUERYRBINFO shQueryRBInfo = new SHQUERYRBINFO();

    // Größe der Struktur definieren, damit die API-Funktion diese beim
    // Schreiben kennt
    shQueryRBInfo.cbSize = Marshal.Sizeof(typeof(SHQUERYRBINFO));

    // Alle ermittelten Laufwerke durchgehen
    for (int i = 0; i < drives.Count; i++)
    {
        // SHQueryRecycleBin aufrufen und eventuelle Fehler auswerten
        int result = SHQueryRecycleBin(drives[i] + "\\x0",
            ref shQueryRBInfo);
        if (result == 0)
        {
            // Kein Fehler: Die gelesene Daten mit dem Ergebnis addieren
            rbi.Size += shQueryRBInfo.i64Size;
            rbi.Count += shQueryRBInfo.i64NumItems;
        }
        else
        {
            // Fehler beim Lesen: Ausnahme erzeugen
            throw new Exception(String.Format(
                "Fehler 0x{0:X} beim Lesen des Papierkorbs", result));
        }
    }

    return rbi;
}

```

Listing 198: Methode zum Ermitteln der Anzahl der Elemente im Papierkorb und der Gesamtgröße der enthaltenen Dateien (Forts.)

Abbildung 73 zeigt eine Anwendung, die über `GetRecycleBinInfo` die Größe des gesamten Papierkorbs und die des Papierkorbs des Laufwerks C: ermittelt hat.

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

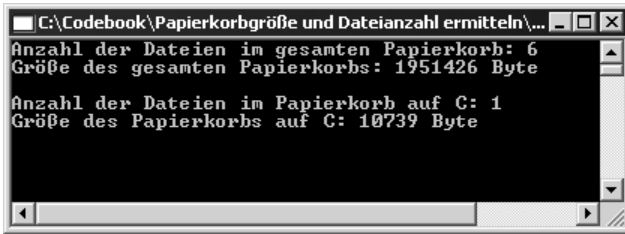


Abbildung 73: Das Programm zur Ermittlung der Papierkorbinfos in Aktion

117 Verknüpfungen anlegen

Wie Sie ja wahrscheinlich schon wissen, erlaubt Windows das Anlegen von Verknüpfungen (Links) zu Dateien und Ordnern. Verknüpfungen verwalten intern einen Pfad zur Datei bzw. zum Ordner, eventuelle Aufrufargumente (bei Programmdateien) und ein Arbeitsverzeichnis. Über Verknüpfungen, die in einem beliebigen Ordner gespeichert sein können, können Sie Dateien und Ordner öffnen, als würden Sie diese direkt öffnen.

Im Explorer erzeugen Sie eine Verknüpfung zum einen über das Ziehen einer Datei oder eines Ordners mit anschließendem Fallenlassen unter der Betätigung der STRG- und der SHIFT-Taste. Zum anderen können Sie eine Verknüpfung auch über das Explorer-Menü anlegen.

Verknüpfungen können Sie aber (natürlich) auch in einem Programm erzeugen. Sinnvoll ist das z. B., wenn eine Anwendung in bestimmten Situationen automatisch E-Mails versendet, die sich beispielsweise auf Dateien im Netzwerk oder auf bestimmte Datensätze in einer Datenbank beziehen. Über eine Verknüpfung zu den Dateien bzw. zu einem Programm, das die Datensätze anzeigen kann, können Sie dem Anwender die Möglichkeit geben, die assoziierten Informationen auf eine einfache Weise zu öffnen.

Leider bietet das .NET Framework scheinbar keine Möglichkeit, Verknüpfungen anzulegen. Dazu müssen Sie einmal wieder das Windows-API bemühen. Etwas problematisch daran ist, dass die Features zur Erzeugung von Verknüpfungen nicht in einer klassischen DLL, sondern in COM-Schnittstellen implementiert sind (die allerdings in diesem Fall funktionsfähige Klassen sind). Die Haupt-Schnittstellen werden mit `IShellLinkA` (ASCII) und `IShellLinkW` (Unicode) bezeichnet. Daneben existiert (u.a.) noch eine Schnittstelle `IPersistFile` zum Lesen und Schreiben von (Verknüpfungs-)Dateien.

Im Internet kursiert dazu eine COM-Typbibliothek *ShellLink.tlb*, in der diese Schnittstellen deklariert sind. (COM-)Anwendungen können diese Typbibliothek referenzieren, um Verknüpfungen anzulegen. In einem .NET-Programm können Sie diese Typbibliothek ebenfalls zur Erzeugung von Verknüpfungen verwenden.

Eine reine .NET-Lösung ist aber immer die bessere. Und die ist möglich, denn Sie können mit speziellen Attributen auch COM-Schnittstellen in ein .NET-Programm importieren. Das Ganze ist leider etwas kompliziert und erfordert eine Menge an COM-Grundlagenwissen.

Erfreulicherweise hat uns diese komplexe Arbeit *Mattias Sjögren* in Form der Dateien *ShellLinkNative.cs* (native Shell-Link-Klassen) und *ShellShortcut.cs* (.NET-Wrapper für die nati-

ven Klassen) abgenommen. Mit Hilfe der Klasse `ShellShortcut` ist es sehr einfach, eine Verknüpfung anzulegen. An der Adresse www.msjogren.net/dotnet/eng/samples/dotnet_shelllink.asp finden Sie ein Beispiel, das diese Klassen einsetzt. Die Klassen finden Sie auch im Komponenten-Ordner auf der CD.

Ich stelle die (relativ umfangreichen) Shell-Link-Klassen hier nicht vor, sondern zeige lediglich, wie Sie mit deren Hilfe eine Verknüpfung anlegen. Dazu erzeugen Sie zunächst eine Instanz der Klasse `ShellShortcut`, der Sie im Konstruktor den vollen Dateinamen der zu erzeugenden Verknüpfung übergeben. Die Dateierweiterung solcher Dateien ist üblicherweise `.lnk`.

Danach stellen Sie zumindest die Eigenschaften `Path` (Pfad zur verknüpften Datei bzw. zum verknüpften Ordner) ein. Handelt es sich bei dieser Datei um ein Programm, können Sie in der Eigenschaft `Arguments` Aufrufargumente definieren. Im Beispiel setze ich den Pfad zum Windows-Editor und übergebe in den Argumenten einen Pfad zu einer Textdatei.

Die Eigenschaft `Description` verwaltet die Beschreibung der Verknüpfung. In der Eigenschaft `WorkingDirectory` können Sie das Arbeitsverzeichnis definieren, das ein Programm vorzugsweise zum Öffnen und Speichern von Dateien verwendet. Über die Eigenschaft `Icon` können Sie die Verknüpfung mit einem Icon versehen, das Sie im Programm selbst erzeugt oder ausgelesen haben. Normalerweise reicht jedoch die Angabe des Pfades zu einer Datei, die das darzustellende Icon speichert (gewöhnlich ist das die verknüpfte Datei selbst), in der Eigenschaft `IconPath`. Dann müssen Sie lediglich noch den Index des Icons angeben (Programm-Dateien können mehrere Icons verwalten). Wenn Sie das erste Icon verwenden wollen, geben Sie hier 0 an.

Der folgende Programmcode zeigt die Anwendung der `ShellShortcut`-Klasse am Beispiel einer Verknüpfung zum Windows-Editor, der die Textdatei `eula.txt` im Windows-Systemordner anzeigen soll. Das Programm erfordert die Einbindung der Dateien `ShellLinkNative.cs` und `ShellShortcut.cs` und den Import der Namensräume `System` und `MSJogren.Samples.ShellLink`.

```
// Name der zu erzeugenden Verknüpfungs-Datei
string linkFileName = "c:\\eula.lnk";

// Pfad zu notepad.exe zusammenstellen
string path = Path.Combine(Environment.SystemDirectory, "notepad.exe");

// Argumente für den Aufruf
string arguments = Path.Combine(Environment.SystemDirectory, "eula.txt");

// Instanz der Klasse ShellShortcut erzeugen
ShellShortcut ssh = new ShellShortcut(linkFileName);

// ShellShortcut-Objekt initialisieren
ssh.Description = "Test-Verknüpfung"; // Beschreibung
ssh.WorkingDirectory = "c:\\";          // Arbeitsverzeichnis
```

Listing 199: Erzeugen einer Verknüpfung über die Klasse `ShellShortcut` von Mattias Sjögren

Basics

Datum
ZeitAnwende-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
ssh.Path = path;                // Pfad zur Datei / zum Ordner
ssh.Arguments = arguments;      // Aufruf-Argumente
ssh.IconPath = linkFileName;    // Pfad zur Icon-Datei
ssh.IconIndex = 0;              // Index des Icons

try
{
    // Verknüpfung speichern
    ssh.Save();

    Console.WriteLine("Fertig");
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Anlegen der Verknüpfung: {0}",
        ex.Message);
}
```

Listing 199: Erzeugen einer Verknüpfung über die Klasse ShellShortcut von Mattias Sjögren (Forts.)

Die `Save`-Methode speichert die Verknüpfung auch dann, wenn bereits eine gleichnamige Datei existiert. Für den Fall, dass Sie eine bereits vorhandene Verknüpfung nicht überschreiben wollen, sollten Sie also vor dem Anlegen der Verknüpfung überprüfen, ob bereits eine solche Datei existiert. Da beim Speichern natürlich auch Fehler auftreten können, sollten Sie, wie im Beispiel, die `Save`-Methode innerhalb einer Ausnahmebehandlung aufrufen.

118 Textdateien lesen

Textdateien können Sie auf eine sehr einfache Weise über eine Instanz der Klasse `System.IO.StreamReader` lesen. Die Methode `Read` erlaubt das Lesen einzelner Zeichen, über die Methode `ReadLine` können Sie zeilenweise lesen, die Methode `ReadToEnd` ermöglicht das Einlesen einer Textdatei in einem Rutsch. `Read` und `ReadLine` geben einen String zurück, wenn (noch) gelesen werden konnte. Ist der Stream am Ende der Datei angelangt, geben beide Methoden `null` zurück. In einer Schleife, die den jeweils eingelesenen String auf ungleich `null` vergleicht, können Sie sehr einfach bis zum Ende der Datei einlesen.

Eine `StreamReader`-Instanz können Sie über die Methode `OpenText` eines `FileInfo`-Objekts erzeugen. Den Dateinamen übergeben Sie am ersten Argument des Konstruktors:

```
System.IO.FileInfo fi = new System.IO.FileInfo(fileName);
System.IO.StreamReader sr = fi.OpenText();
```

Alternativ können Sie die gleichnamige statische Methode der `File`-Klasse verwenden:

```
System.IO.StreamReader sr = System.IO.File.OpenText(fileName);
```

Beide Varianten besitzen den Nachteil, dass Sie die Codierung der Datei nicht angeben können. Textdateien können ja, wie Sie wahrscheinlich bereits wissen, in verschiedenen ASCII- oder Unicode-Codierungen gespeichert sein. Die Methode `OpenText` liest die Datei in beiden Varianten mit der UTF-8-Codierung ein. Das führt spätestens dann zu inkorrekt dargestellten Texten, wenn der Text Sonderzeichen (wie unsere Umlaute) enthält und in einer anderen Codierung gespeichert ist.

Erzeugen Sie die `StreamReader`-Instanz stattdessen explizit, können Sie die Codierung selbst bestimmen. Dazu übergeben Sie am zweiten Argument eine Instanz der Klasse `System.Text.Encoding`. Diese können Sie über einige statische Eigenschaften der `Encoding`-Klasse (die `Encoding`-Instanzen für Standard-Codierungen zurückgeben) oder über die Methode `GetEncoding` (der Sie den Namen oder die Id einer Codierung übergeben) erzeugen.

Eigenschaft/Methode	Bedeutung
ASCII	7-Bit (!) ASCII-Codierung
BigEndianUnicode	Unicode in der Byte-Reihenfolge <i>Big-Endian</i> . Intel-Plattformen verwenden standardmäßig die Reihenfolge <i>Little Endian</i> .

Tabelle 9: Die statischen Eigenschaften der `Encoding`-Klasse zur Ermittlung einer Standard-Codierung

Eigenschaft/Methode	Bedeutung
Default	die vom System verwendete Standard-Codierung
Unicode	Standard-Unicode
UTF7	Codierung im Format UTF-7, bei dem Unicode-Daten so konvertiert werden, dass diese einem 7-Bit-ASCII-Code entsprechen
UTF8	Codierung im Format UTF-8, bei dem Unicode-Daten so konvertiert werden, dass diese einem 8-Bit-ASCII-Code entsprechen

Tabelle 9: Die statischen Eigenschaften der Encoding-Klasse zur Ermittlung einer Standard-Codierung (Forts.)

Voraussetzung dafür ist natürlich, dass Sie die Codierung der Textdatei kennen, was in der Praxis oft nicht allzu einfach herauszufinden ist.

Das folgende Beispiel erzeugt eine `StreamReader`-Instanz für die Codierung *Windows-1252* (Windows-Standard):

```
System.IO.StreamReader sr = new System.IO.StreamReader(fileName,
    System.Text.Encoding.GetEncoding("windows-1252"));
```

Kennen Sie die Codierung nicht, können Sie über `true` am zweiten Argument erreichen, dass die `GetEncoding`-Methode versucht, die Codierung automatisch an den ersten Bytes der Textdatei zu erkennen:

```
System.IO.StreamReader sr = new System.IO.StreamReader(fileName, true);
```

Diese automatische Erkennung funktioniert allerdings nur für Dateien im UTF-8- oder in einem Unicode-Format, und auch nur dann, wenn diese mit einer so genannten Byte-Reihenfolgemarkierung beginnen. In der Praxis können Sie mit dieser Variante also wenig anfangen, besonders dann, wenn Sie 8-Bit-Textdateien einlesen.

Das eigentliche Lesen ist dann sehr einfach. Da in den meisten Fällen wohl zeilenweise gelesen wird, zeige ich diese Variante. Das Beispiel in Listing 200 liest die Datei *hitchhiker.txt* ein und gibt die gelesenen Zeilen an der Konsole aus. Die Datei wird im Ordner der Anwendung erwartet. Da gerade beim Öffnen von Dateien Ausnahmen des Typs `IOException` möglich sind (z.B., weil die Datei nicht vorhanden oder gerade exklusiv gesperrt ist), werden diese abgefangen.

Das Beispiel benötigt die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text` und `System.Windows.Forms`.

```
// Dateiname ermitteln
string fileName = Path.Combine(Application.StartupPath, "hitchhiker.txt");

// StreamReader-Instanz für die Datei erzeugen
StreamReader sr = null;
try
{
    sr = new StreamReader(fileName, Encoding.GetEncoding("windows-1252"));
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Öffnen der Datei '" + fileName + "': " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}

// Datei zeilenweise einlesen
string line = null;
while ((line = sr.ReadLine()) != null)
{
    Console.WriteLine(line);
}

// StreamReader schließen
sr.Close();
```

Listing 200: Zeilenweises Einlesen einer Textdatei mit der Codierung Windows-1252

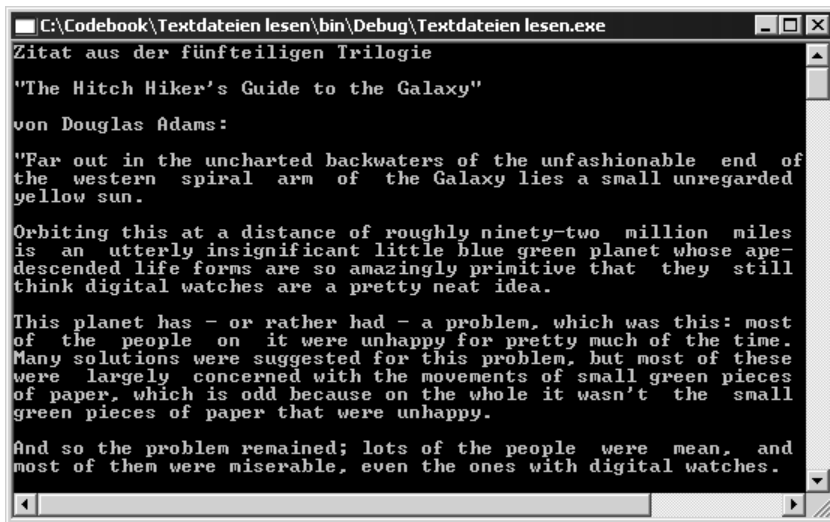


Abbildung 74: Das Beispielprogramm hat die Textdatei eingelesen

Zum Kompilieren dieses Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume *System.IO*, *System.Text*, *System.Reflection* und *System.Windows.Forms* importieren.

Abbildung 74 zeigt das ausgeführte Programm.

119 Textdateien schreiben

Das Schreiben von Textdateien ist ähnlich einfach wie das Lesen (siehe vorhergehendes Rezept). Das einzige Problem ist wieder die Einstellung der korrekten Codierung.

Zum Erstellen einer Textdatei erzeugen Sie eine Instanz der Klasse *StreamWriter* (aus dem Namensraum *System.IO*). Deren Konstruktor existiert in verschiedenen überladenen Varianten. So können Sie einen Dateinamen am ersten Argument des Konstruktors angeben, womit die *StreamWriter*-Instanz implizit einen Stream zum Schreiben der Datei erzeugt. Alternativ können Sie aber auch ein existierendes *Stream*-Objekt (oder eine Instanz einer von *Stream* abgeleiteten Klasse) angeben, um zu erreichen, dass das *StreamWriter*-Objekt in dieses schreibt. In weiteren Argumenten können Sie festlegen, ob an vorhandene Dateien angehängt werden soll, Sie können die Codierung bestimmen etc. Ich verwende im Beispiel die Variante, die einen Dateinamen am ersten Argument erwartet.

Bei dieser Variante übergeben Sie am ersten Argument also den Namen der Textdatei. Am zweiten Argument können Sie festlegen, ob die Textdaten an eine eventuell bereits vorhandene Datei angehängt werden oder ob diese überschrieben wird. Das dritte Argument bestimmt die Codierung der Textdatei.

Über die Methode *Write* können Sie dann einzelne Zeichenketten in die Datei schreiben. Die Methode *WriteLine* schreibt eine Zeichenkette und hängt zusätzlich die für das System gültigen Zeilenendezeichen (unter Windows sind das die Zeichen »\r\n«) an diese an.

Wenn Sie neue Textdateien schreiben (also beim Erzeugen der *StreamWriter*-Instanz am zweiten Argument *false* übergeben), sollten Sie beachten, dass ein *StreamWriter* vorhandene Dateien ohne Warnung überschreibt. Sie sollten also gegebenenfalls vor dem Schreiben überprüfen, ob eine gleichnamige Datei bereits existiert.

Das folgende Beispiel erzeugt eine neue Textdatei *fish.txt* im Ordner der Anwendung mit der Codierung *Windows-1252* und beschreibt diese mit zwei Zeilen Text. Ist die zu erzeugende Datei bereits vorhanden, fragt das Programm den Anwender, ob er diese überschreiben will. Ausnahmen vom Typ *IOException*, die im Besonderen beim Öffnen der Datei erzeugt werden können, werden abgefangen.

Um das Beispiel kompilieren zu können müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume *System*, *System.IO*, *System.Text*, *System.Reflection* und *System.Windows.Forms* importieren.

```
// Dateiname ermitteln
string fileName = Path.Combine(Application.StartupPath, "fish.txt");
```

```
// Überprüfen, ob die Datei bereits existiert
if (File.Exists(fileName))
{
    // Den Anwender fragen, ob er die Datei überschreiben will
    if (MessageBox.Show("Die Datei '" + fileName + "' existiert " +
        "bereits.\r\n\r\nWollen Sie diese Datei überschreiben?",
        Application.ProductName, MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.No)
        return;
}

// StreamWriter-Instanz für die Datei erzeugen
StreamWriter sw = null;
try
{
    sw = new StreamWriter(fileName, false, Encoding.GetEncoding("windows-1252"));
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Öffnen der Datei '" + fileName + "': " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}

// Datei zeilenweise schreiben
sw.WriteLine("Macht's gut");
sw.WriteLine("und danke für den Fisch");

// StreamWriter schließen
sw.Close();
```

Listing 201: Schreiben einer Textdatei in der Codierung Windows-1252 (Forts.)

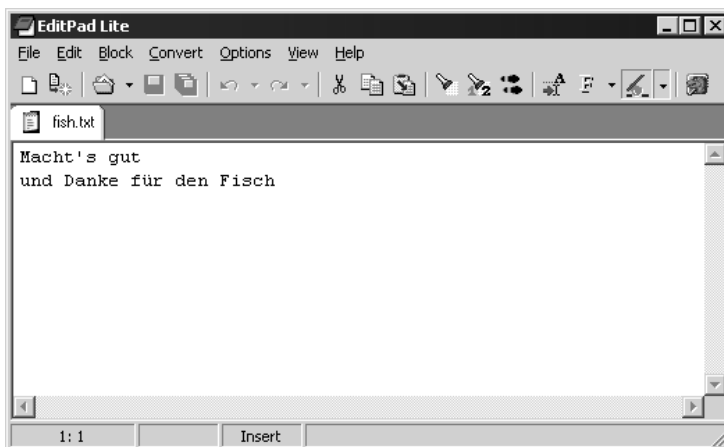


Abbildung 75: Die erzeugte Textdatei in einem Editor

120 Texte an Textdateien anfügen

Das Anfügen von Texten an vorhandene Textdateien ist wie das Schreiben neuer Dateien sehr einfach. Dazu erzeugen Sie wie im vorhergehenden Rezept eine Instanz der Klasse `StreamWriter`, übergeben aber am zweiten Argument `append` nun `true`. Damit wird eine vorhandene Datei so geöffnet, dass neue Zeichen oder Zeichenketten an das Ende angefügt werden. Ist die Datei noch nicht vorhanden, wird sie automatisch angelegt.

Das Beispiel in Listing 202 hängt einen String mit dem aktuellen Datum und der aktuellen Anzahl ausgeführter Prozesse an die Datei *log.txt* an. Diese Datei wird im Ordner der Anwendung erwartet. Ausnahmen, die im Besonderen beim Öffnen der Datei entstehen können, werden abgefangen.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Text` und `System.Windows.Forms` importieren.

```
// Dateiname ermitteln
string fileName = Path.Combine(Application.StartupPath, "log.txt");

// StreamWriter-Instanz für die Datei erzeugen
StreamWriter sw = null;
try
{
    sw = new StreamWriter(fileName, true, Encoding.GetEncoding(
        "windows-1252"));
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Öffnen der Datei '" + fileName + "': " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}

// Zeile anfügen
sw.WriteLine(DateTime.Now.ToString() + ": Anzahl Prozesse: " +
    System.Diagnostics.Process.GetProcesses().Length);

// StreamWriter schließen
sw.Close();
```

Listing 202: Anhängen von Texten an eine Textdatei

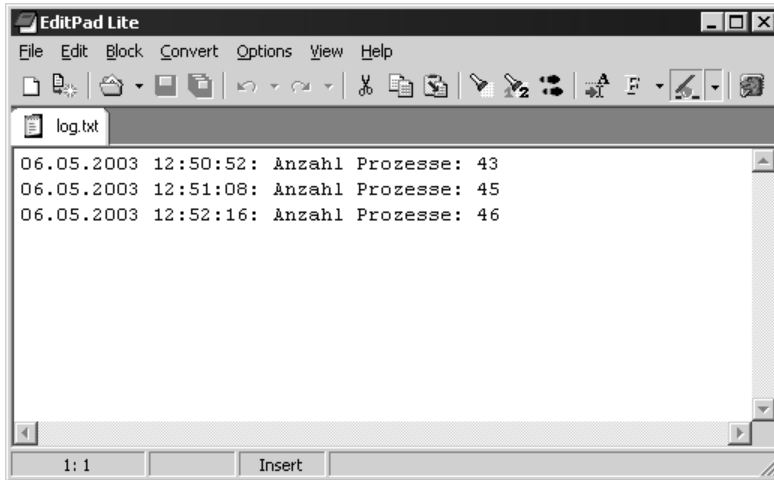


Abbildung 76: Die vom Beispielprogramm nach drei Ausführungen erzeugte Textdatei in einem Editor

121 Textdateien ändern

Das Ändern von Textdateien ist nicht direkt möglich. Dazu müssen Sie die Datei einlesen, im Arbeitsspeicher ändern, die Originaldatei löschen und schließlich neu schreiben. Etwas problematisch wird dies, wenn die Datei sehr groß ist und folglich den Arbeitsspeicher beim kompletten Einlesen überlasten würde. In diesem Fall können Sie so vorgehen, dass Sie die Datei Zeile für Zeile einlesen, die einzelnen Zeilen bearbeiten und die gewünschten (und eventuell veränderten Zeilen) in eine temporäre Datei schreiben. Nach dem Bearbeiten löschen Sie die originale Textdatei und verschieben die temporäre Datei mit dem Namen der Textdatei in den Ordner derselben.

Listing 203 implementiert dieses zeilenweise Verändern einer Textdatei in Form der Methode `AddFileNumbers`, die der Datei Zeilennummern hinzufügt.

Zum Kompilieren der Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Text` importieren.

```
public static void AddFileNumbers(string textFileName)
{
    // Dateiname für die temporäre Datei ermitteln
    string tempFileName = Path.GetTempFileName();

    // Textdatei zum Lesen öffnen
    StreamReader sr = null;
    sr = new StreamReader(textFileName, Encoding.GetEncoding("windows-1252"));

    // Temp-Datei zum Schreiben öffnen
    StreamWriter sw = null;
    sw = new StreamWriter(tempFileName, false,
```

Listing 203: Verändern einer Textdatei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
Encoding.GetEncoding("windows-1252"));

// Textdatei zeilenweise einlesen, Zeilen verändern und in die temporäre
// Datei schreiben
string line;
int lineNumber = 0;
while ((line = sr.ReadLine()) != null)
{
    lineNumber++;
    sw.WriteLine(lineNumber + ": " + line);
}

// Streams schließen
sr.Close();
sw.Close();

// Textdatei löschen und die temporäre Datei auf die Textdatei verschieben
File.Delete(textFileName);
File.Move(tempFileName, textFileName);
}
```

Listing 203: Verändern einer Textdatei (Forts.)

122 Binäre Dateien lesen

Binäre Dateien können Sie über eine Instanz der Klasse `FileStream` (aus dem Namensraum `System.IO`) einlesen. Ähnlich dem Lesen von Textdateien können Sie diese Instanz über die Methode `OpenRead` einer Instanz der Klasse `FileInfo` oder der `File`-Klasse erzeugen:

```
System.IO.FileStream fs = System.IO.File.OpenRead(fileName);

...

System.IO.FileInfo fi = new System.IO.FileInfo(fileName);
System.IO.FileStream fs = fi.OpenRead();
```

Alternativ (aber hier nicht unbedingt vorteilhaft) können Sie die Instanz der `FileStream`-Klasse auch selbst erzeugen, wobei Sie am zweiten Argument den Modus zum Öffnen (`FileMode.Open`) und am dritten Argument die Zugriffsart (`FileAccess.Read`) übergeben:

```
System.IO.FileStream fs = new System.IO.FileStream(fileName,
    FileMode.Open, FileAccess.Read)
```

Lesen können Sie die Daten dann über die `Read`-Methode. Dieser Methode übergeben Sie am ersten Argument ein `byte`-Array, das Sie zuvor ausreichend groß dimensionieren müssen. In dieses Array schreibt die Methode die gelesenen Bytes.

Byte-Arrays werden unter .NET vorzugsweise für binäre Daten verwendet. Den meisten Streams können Sie im Konstruktor unter anderem ein solches Array übergeben, um die darin gespeicherten Daten an einen anderen Ort zu übertragen.

Am zweiten Argument übergeben Sie die Position, ab der im Stream gelesen werden soll. In der Praxis übergeben Sie hier eigentlich immer 0 um ab der aktuellen Position zu lesen. Das dritte Argument bestimmt die Anzahl der zu lesenden Bytes. Dieses Argument ist wichtig, wenn Sie, wie weiter unten beschrieben, Daten blockweise einlesen.

Kleinere Dateien können Sie in einem Rutsch lesen, wie es der Programmcode in Listing 204 zeigt. Dazu müssen Sie allerdings wissen, wie groß die einzulesende Datei ist, was Sie über die `Length`-Eigenschaft eines `FileInfo`-Objekts erfahren können. Das Programm liest die Datei `hitchhiker.jpg` ein und speichert diese Base-64-codiert¹ in eine XML-Datei `hitchhiker.xml`. Beide Dateien werden im Ordner der Anwendung erwartet.

Das Beispiel erfordert die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Xml` und `System.Windows.Forms`.

```
// Dateinamen ermitteln
string imageFileName = Path.Combine(Application.StartupPath,
    "hitchhiker.jpg");
string xmlFileName = Path.Combine(Application.StartupPath, "hitchhiker.xml");

// FileStream-Instanz erzeugen
FileInfo fi = new FileInfo(imageFileName);
FileStream fs = fi.OpenRead();

// Byte-Puffer erzeugen und die Datei in diesen einlesen
byte[] buffer = new Byte[fi.Length];
fs.Read(buffer, 0, buffer.Length);

// Die eingelesenen Daten Base-64-codiert in eine XML-Datei schreiben
XmlTextWriter xmlWriter = new XmlTextWriter(xmlFileName, Encoding.UTF8);
xmlWriter.WriteStartDocument(true);
xmlWriter.WriteStartElement("applicationData");
xmlWriter.WriteStartElement("splashImage");
xmlWriter.WriteBase64(buffer, 0, buffer.Length);
xmlWriter.WriteEndElement();
xmlWriter.WriteEndElement();

// XML-Datei schließen
xmlWriter.Close();

// FileStream schließen
fs.Close();
```

Listing 204: Einlesen einer binären Datei in einem Rutsch in eine XML-Datei

1. Eine Base-64-Codierung bewirkt, dass binäre Daten so umgesetzt werden, dass diese einem 7-Bit-ASCII-Code entsprechen und problemlos in Textdateien gespeichert oder über das Internet übertragen werden können.

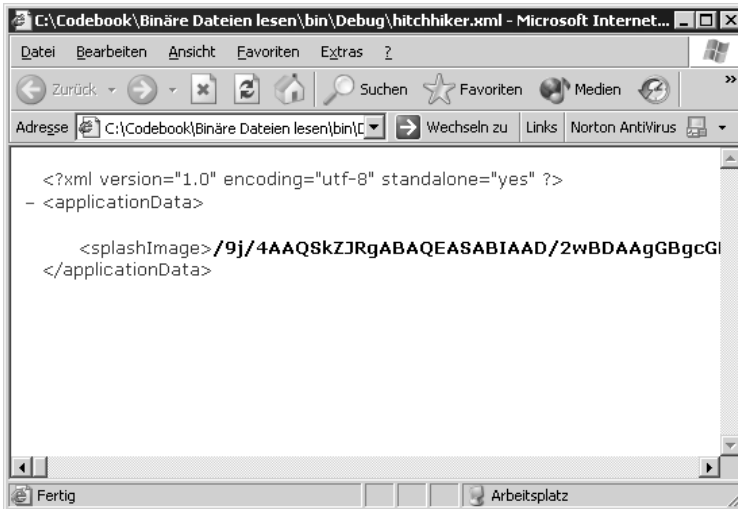


Abbildung 77: Die erzeugte XML-Datei im Internet Explorer

Das Beispiel ist übrigens nicht allzu weit von der Praxis entfernt. Die Speicherung binärer Daten in einer XML-Datei erlaubt es Ihnen z. B., beliebige Konfigurationsdaten (u. a. wie im Beispiel auch Bilder) in einer einzigen Datei zu speichern.

123 Binäre Dateien schreiben

Das Schreiben binärer Daten ist ähnlich einfach wie das Lesen (im vorhergehenden Rezept). Dazu erzeugen Sie ein `FileStream`-Objekt (aus dem Namensraum `System.IO`) und rufen dessen `Write`-Methode auf. Dieser Methode übergeben Sie ein `byte`-Array, eine Startposition für den Stream (normalerweise 0) und die Anzahl der zu schreibenden Bytes. Das `FileStream`-Objekt können Sie über die Methode `OpenWrite` der `FileInfo`-Klasse oder einer Instanz der `File`-Klasse erzeugen. Natürlich ist auch das direkte Erzeugen möglich, wobei Sie im Konstruktor neben dem Dateinamen mindestens den Datei-Modus (`FileAccess.Write`) übergeben müssen. Ideal ist eigentlich die Verwendung der `File`-Klasse, da Sie hier nur den Dateinamen angeben müssen.

Das folgende Beispiel liest die im vorhergehenden Rezept erzeugten XML-Daten ein und erzeugt daraus eine JPEG-Grafikdatei. Da unbekannt ist, wie groß die zu erzeugende Datei wird, werden die Daten blockweise in Stücken von einem KB eingelesen.

Zum Kompilieren dieses Programms müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Text`, `System.Xml` und `System.Windows.Forms` importieren.

```
// Dateinamen ermitteln
string xmlFileName = Path.Combine(Application.StartupPath, "hitchhiker.xml");
string imageFileName = Path.Combine(Application.StartupPath,
    "hitchhiker.jpg");

// FileStream-Instanz zum Schreiben erzeugen
FileStream fs = File.OpenWrite(imageFileName);

// XML-Datei öffnen
XmlTextReader xmlReader = new XmlTextReader(xmlFileName);

// Zum splashImage-Element navigieren (das dem applicationData-Element folgt)
xmlReader.ReadStartElement("applicationData");

// XML-Daten des splashImage-Elements blockweise einlesen
byte [] buffer = new Byte[1024];
int count = 0;
do
{
    count = xmlReader.ReadBase64(buffer, 0, buffer.Length);
    if (count > 0)
    {
        // Eingelesene Daten in den FileStream schreiben
        fs.Write(buffer, 0, count);
    }
} while (count > 0);

// XML-Datei schließen
xmlReader.Close();

// FileStream schließen
fs.Close();
```

Listing 205: Erzeugen einer binären Datei



Abbildung 78: Das aus den XML-Daten erzeugte JPEG-Bild

124 Binäre Dateien Base64-codieren

In vielen Fällen ist es notwendig, binäre Dateien so zu codieren, dass diese problemlos über das Internet versendet werden können. Dazu bietet sich eine Base64-Codierung an, die im Internet weit verbreitet ist. Base64 codiert binäre Daten so, dass diese dem 7-Bit-ASCII-Code entsprechen.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die Methode `ReadAsBase64` in Listing 206 erzeugt aus einer Datei einen Base64-codierten String. Die Datei wird dazu zunächst über eine Instanz der Klasse `FileStream` in ein `byte`-Array eingelesen. Dieses Array übergibt die Methode an die Methode `ToBase64String` der `Convert`-Klasse, die die übergebenen Bytes Base64-codiert.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.IO` importieren.

```
public static string ReadAsBase64(string fileName)
{
    // Datei einlesen
    FileInfo fi = new FileInfo(fileName);
    byte[] buffer = new byte[fi.Length];
    FileStream fs = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    fs.Read(buffer, 0, buffer.Length);
    fs.Close();

    // Das Byte-Array Base64-codiert zurückgeben
    return Convert.ToBase64String(buffer, 0, buffer.Length);
}
```

Listing 206: Methode zum Lesen einer Datei als Base64-codierten String

125 Base64-codierte Strings in Dateien umwandeln

Im Rezept 124 habe ich eine Methode vorgestellt, die eine binäre Datei in einen Base64-codierten String umwandelt. Die Methode `CreateFileFromBase64` in Listing 207 macht genau das Gegenteil: die Umwandlung eines Base64-codierten Strings in ein `Byte`-Array, das dann als Datei gespeichert wird. Zur Umwandlung setzt `CreateFileFromBase64` die Methode `FromBase64String` der `Convert`-Klasse ein.

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` importieren.

```
public static void CreateFileFromBase64(string base64CodedString,
    string fileName)
{
    // String in Byte-Array konvertieren
    byte[] buffer = Convert.FromBase64String(base64CodedString);

    // Datei über einen FileStream schreiben
    FileStream fs = new FileStream(fileName, FileMode.Create,
        FileAccess.Write);
    fs.Write(buffer, 0, buffer.Length);
    fs.Close();
}
```

Listing 207: Methode zum Erzeugen einer Datei aus einem Base64-codierten String

126 (ZIP-)Archive erzeugen

Wenn Sie in einem .NET-Programm Dateien oder Ordner in ZIP-Dateien oder andere Archive packen oder solche Archive entpacken wollen, können Sie nicht auf Klassen des .NET Framework zugreifen. Auch das Windows-API bietet wohl erst ab Windows XP entsprechende Funktionen (ich habe diese allerdings noch nicht gefunden).

Im Internet finden Sie natürlich eine Vielzahl an Bibliotheken, über die Sie mit Archiven arbeiten können. Eine der besten ist *#ziplib* (*SharpZipLib*). Diese in C# geschriebene Bibliothek wurde von *Mike Krüger*, dem Entwickler der bekannten Freeware-Entwicklungsumgebung *SharpDevelop*, auf der Basis existierenden Java-Quellcodes programmiert. Sie finden *#ziplib* an der Adresse www.icsharpcode.net/OpenSource/SharpZipLib.

#ziplib wird unter der GPL-Lizenz (»GNU General Public License«) mit einer Ausnahme veröffentlicht. Kurz gesagt besagt diese Lizenz, dass Entwickler, die unter der GPL-Lizenz entwickeln, immer den Quellcode ihrer Programme mit ausliefern müssen. Andere Programmierer dürfen diesen Quellcode verändern, daraus eigene Programme erzeugen und diese sogar verkaufen. Da dabei immer ein Hinweis auf den Urheber des Programms und die vorgenommenen Veränderungen in das Programm integriert werden müssen, sind die Urheberrechte aller Entwickler geschützt. Ziel dieser Lizenz ist, Betriebssysteme und Programme zu schaffen, deren Quellen frei verfügbar, für jeden zugänglich und nutzbar sind. Auf diese Weise können sich sehr viele Programmierer an der Weiterentwicklung von Programmen und Betriebssystemen beteiligen. Linux, dessen Bestandteile zumeist auch unter der GPL-Lizenz veröffentlicht sind, ist das beste Beispiel für das Funktionieren dieser Idee.

Die Ausnahme von der Standard-Lizenz bei *#ziplib* ist, dass Programme, die diese Bibliothek verwenden, selbst nicht unter der GPL-Lizenz veröffentlicht werden müssen. Dabei wird vorausgesetzt, dass das Programm keine Klassen enthält, die von den *#ziplib*-Klassen abgeleitet werden. So können Sie mit *#ziplib* auch kommerzielle Programme schreiben, die nicht unter die GPL-Lizenz fallen.

#ziplib können Sie über die Assembly *ICSharpCode.SharpZipLib.dll* in ein Programm integrieren, indem Sie diese referenzieren. Die Assembly können Sie über die Adresse www.icsharpcode.net/OpenSource/SharpZipLib/Download.aspx herunterladen, indem Sie die Option COMPILED ASSEMBLY WITH DOCUMENTATION AND INSTALL SCRIPTS DOWNLOAD wählen. Im heruntergeladenen Zip-Archiv, das in der zum Zeitpunkt der Erstellung dieses Buch aktuellen Version 0.5 auch auf der Buch-CD gespeichert ist, finden Sie neben der DLL auch eine (leider nicht detaillierte) Windows-Hilfe-Dokumentation. Auf der Website können Sie zusätzlich auch den Quellcode der Bibliothek herunterladen.

#ziplib ist mit einem »Strong name« vorbereitet auf die Installation im Globalen Assembly Cache (GAC). Die Datei *installGAC.bat* enthält einen Aufruf des Tools *gacutil*, der diese Installation bewirkt. Beachten Sie, dass die Systemvariable *PATH* den Pfad zum Bin-Ordner des .NET Framework-SDK enthalten muss, damit *gacutil* ohne Pfadangabe aufgerufen werden kann. Falls Sie die Assembly im GAC installieren wollen, sollten Sie diese vor der GAC-Installation in einen Ordner kopieren, den Sie für wiederverwendbare .NET-Klassenbibliotheken vorgesehen haben.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Hinweis

Die Installation im GAC bewirkt, dass mehrere Anwendungen dieselbe Assembly gleichzeitig verwenden können (die Assembly muss dann nicht im Anwendungsordner gespeichert sein). Wenn Sie diese Features nicht benötigen, sollten Sie Assemblies nicht im GAC installieren. Wenn Sie Assemblies im GAC installiert haben, sollten Sie diese laut Microsoft immer über deren originalen Speicherort referenzieren (und nicht über den Assembly-Ordner im Windows-Verzeichnis). Die Installation im GAC bewirkt leider auch nicht, dass die Referenzliste in Visual Studio .NET diese Assemblies auflistet. Um dies zu erreichen, beachten Sie den folgenden Tipp.

Tipp

Erzeugen Sie einen neuen Schlüssel mit einem eindeutigen Namen für die neue Assembly (z. B. *SharpZipLib*) in der Registry unter den Schlüsseln *HKEY_LOCAL_MACHINE\Software\Microsoft\ .NETFramework\AssemblyFolders* und *HKEY_CURRENT_USER\Software\Microsoft\ .NETFramework\AssemblyFolders* und setzen Sie den Defaultwert dieser Schlüssel auf den Pfad, der die Assembly speichert. Nach einem Visual Studio .NET-Neustart wird die Assembly in der Liste des Referenz-Hinzufügen-Dialogs aufgeführt.

Um eine Assembly im Referenz-Hinzufügen-Dialog von Visual Studio .NET verfügbar zu haben, müssen Sie die Registry bearbeiten. Dazu starten Sie das Programm *regedit* im Ausführen-Dialog von Windows oder an der Konsole. Bevor Sie aber an die Arbeit gehen: Beachten Sie, dass Sie mit der Registry sehr vorsichtig umgehen müssen und nur Veränderungen vornehmen sollten, von denen Sie wissen, dass diese keinen Schaden anrichten. Durch fehlerhafte Veränderungen in der Registry können Sie das gesamte System beschädigen (so, jetzt bin ich nicht mehr für Ihr eventuell defektes Windows-System verantwortlich ☺).

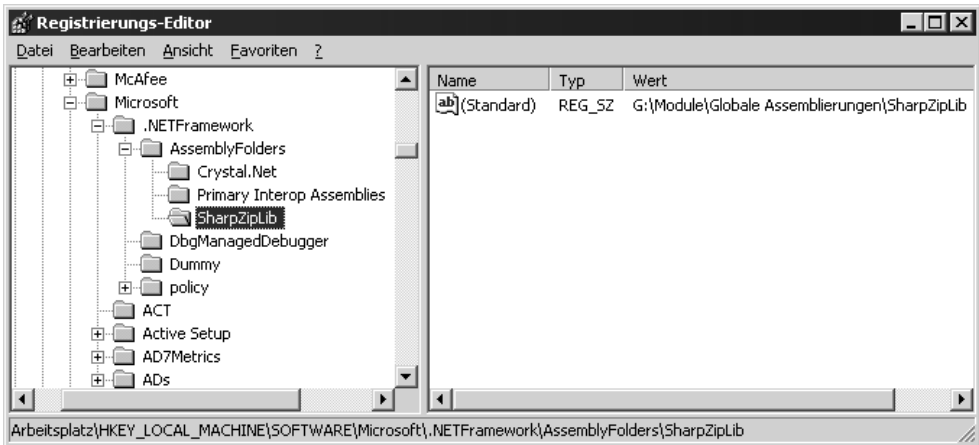


Abbildung 79: Anlegen eines Registry-Schlüssels für das Auflisten der #zip-lib-Assembly in der Visual Studio .NET-Liste

Um das Ganze zu vereinfachen, können Sie auch die `.reg`-Datei, die ich der Komponente beigelegt habe, mit einem Texteditor editieren und im Explorer über einen Doppelklick ausführen, um die entsprechenden Schlüssel anzulegen.

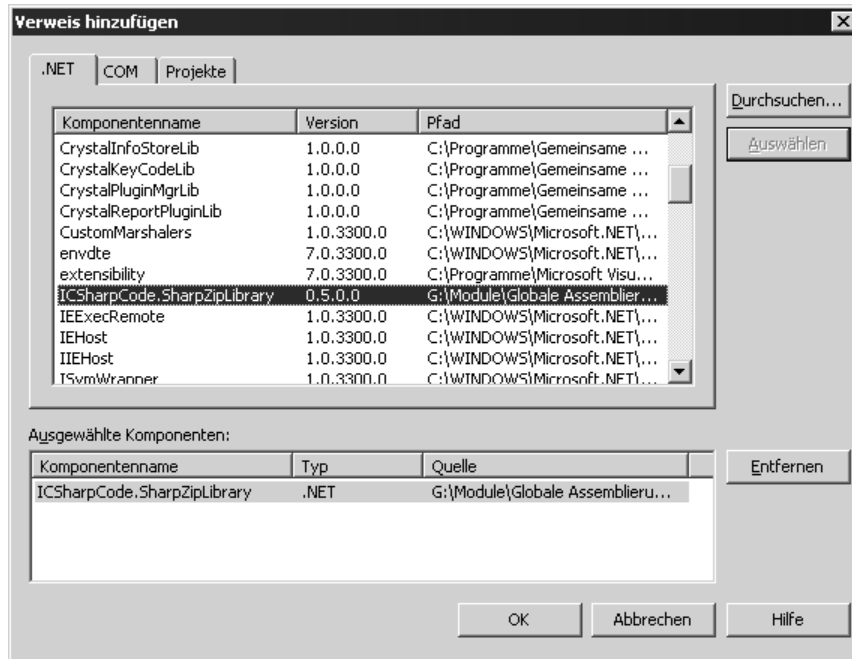


Abbildung 80: Die SharpZipLib-Komponente taucht nach der Registrierung in der Assemblies-Liste des Referenz-Hinzufügen-Dialogs von Visual Studio .NET auf

`#ziplib` ist recht komplex und ermöglicht die Arbeit mit `zip`-, `tar`-, `gzip`- und `bzip2`-Archiven. Ich beschränke mich in diesem Buch auf Zip-Archive. Sie können diese Archive im Dateisystem erzeugen, aus Streams »on the fly« erzeugen, im Dateisystem entpacken, die enthaltenen Dateien über Streams lesen und bearbeiten etc. Ich kann hier nicht die vollständige Funktionalität behandeln und zeige deshalb nur die in meinen Augen wichtigsten Features in diesem und den folgenden Rezepten.

Der in diesem Buch vorgestellte Programmcode basiert auf der Version 0.5, die der Buch-CD beiliegt. Wenn Sie die Komponente aus dem Internet herunterladen, handelt es sich wahrscheinlich um eine neuere Version, die u. U. mit der Version 0.5 nicht mehr kompatibel ist. Es kann also sein, dass Sie den folgenden Programmcode mit der aktuellen Version nur unter einigen Anpassungen verwenden können

Um ein Zip-Archiv aus vorhandenen Dateien zu erzeugen, erzeugen Sie zunächst eine Instanz der Klasse `ZipOutputStream`. Im Konstruktor übergeben Sie ein `Stream`-Objekt, das die

zu erzeugende Zip-Datei repräsentiert. Der `ZipOutputStream` verwendet diesen Stream zur Speicherung der Datei. Über die Methode `setLevel` können Sie die Kompressionsrate mit Werten zwischen 0 und 9 definieren. Eine Kompressionsrate von 0 bewirkt keine Komprimierung, 9 führt zu einer maximalen Komprimierung (was aber auch den Archivierungsvorgang verlangsamt). Die (nicht dokumentierte) Voreinstellung ist 6.

Über die Methode `setComment` können Sie einen Kommentar im Archiv speichern.

Dann erzeugen Sie für jede hinzuzufügende Datei ein `ZipEntry`-Objekt, das einen Datei- oder Ordneintrag in der Zip-Datei repräsentiert. Dieses Objekt initialisieren Sie im Konstruktor mit dem Dateinamen, so wie dieser im Archiv gespeichert werden soll. Hier können Sie auch (relative) Pfadangaben verwenden, um Dateien so zu archivieren, dass diese beim Entpacken in Unterordnern gespeichert werden. In der Eigenschaft `comment` können Sie einen Kommentar zu der archivierten Datei ablegen. Dieses Objekt fügen Sie dem `ZipOutputStream`-Objekt dann über die Methode `putNextEntry` hinzu.

Nun müssen Sie noch die eigentliche Datei in den Stream schreiben. Dazu öffnen Sie die Datei in einen `FileStream` und lesen deren Inhalt über die `Read`-Methode in ein Byte-Array. Dieses Array schreiben Sie dann über die `Write`-Methode in den `ZipOutputStream`. Handelt es sich um eine große Datei, sollten Sie die Datei blockweise, z. B. in Teilen von einem MB aus dem Datei-Stream lesen und in den Zip-Stream schreiben, um den Arbeitsspeicher nicht zu überlasten. Den `FileStream` müssen Sie dann natürlich noch schließen.

Das Erzeugen des `ZipEntry`-Objekts und das Hinzufügen der Datei in den `ZipOutputStream` wiederholen Sie für jede hinzuzufügende Datei. Schließlich rufen Sie noch die `finish`-Methode des `ZipEntry`-Objekts auf um die Zip-Datei zu erzeugen, und schließen den `ZipOutputStream`.

Die Methode `zipFiles` in Listing 208 geht auf diese Weise vor um mehrere in einem String-Array übergebene Dateien in ein Archiv zu schreiben. Übergeben wird zunächst das Array mit den Quelldateinamen, der Name der zu erzeugenden Zip-Datei und die Blockgröße in Byte.

Das Argument `zipLevel` legt die Kompressionsrate fest. Sie können hier einen Wert von 0 (keine Kompression) bis 9 (maximale Kompression) eintragen. Eine höhere Kompression verlangsamt natürlich den Archivierungsvorgang.

Am Argument `includePaths` wird festgelegt, ob der Pfad der Datei mit gespeichert werden soll. In Zip-Archiven können Dateien ohne, mit einer relativen oder einer absoluten Pfadangabe archiviert werden. Die Angabe von Pfaden bringt beim normalen Entpacken eines Archivs den Vorteil, dass die Dateien wieder in einen entsprechenden Ordner entpackt werden. Um möglichst flexibel zu sein, können Sie beim Aufruf von `zipFiles` entscheiden, ob der Pfad jeder Datei mit aufgenommen werden soll. Soll der Pfad mit aufgenommen werden, entfernt `zipFiles` aber den Laufwerkbuchstaben aus der Pfadangabe, da auf dem Zielsystem nicht vorausgesetzt werden kann, dass das entsprechende Laufwerk existiert.

Am letzten Argument können Sie einen Kommentar zur Zip-Datei übergeben, den die meisten Zip-Programme beim Lesen eines Archivs automatisch anzeigen.

Zum Kompilieren dieser Methode müssen Sie die Assembly *ICSharpCode.SharpZipLib.dll* referenzieren und die Namensräume `System`, `System.IO` und `ICSharpCode.SharpZipLib.Zip` importieren.

```
public static void ZipFiles(string[] sourceFileNames, string zipFileName,
    int blockSize, int zipLevel, bool includePaths, string comment)
{
    // Datei-Stream als Basis-Stream erzeugen
    Stream zipFileStream = File.Open(zipFileName, FileMode.CreateNew);

    // ZipOutputStream zum Schreiben der Zip-Datei erzeugen
    ZipOutputStream zipOutputStream = new ZipOutputStream(zipFileStream);

    // Kompressionsrate definieren (0 bis 9)
    zipOutputStream.SetLevel(zipLevel);

    // Kommentar zum Archiv definieren
    zipOutputStream.SetComment(comment);

    // Alle im Array sourceFileNames übergebenen Dateien
    // durchgehen und in das Archiv schreiben
    for (int i = 0; i < sourceFileNames.Length; i++)
    {
        // ZipEntry-Objekt für die neue Datei erzeugen. Der im Konstruktor
        // übergebene Name wird als Dateiname beim Extrahieren verwendet.
        // Sie können hier auch (relative) Pfadangaben mit angeben.
        // Das Programm speichert den Pfad (ohne Laufwerkangabe) mit,
        // wenn das Argument includePaths true ist
        string fileNameForZip;
        FileInfo fi = new FileInfo(sourceFileNames[i]);
        if (includePaths)
        {
            // Dateiname ohne Laufwerksbuchstabe ermitteln
            int pos = fi.FullName.IndexOf('\\');
            if (pos > -1)
                fileNameForZip = fi.FullName.Substring(pos,
                    fi.FullName.Length - pos);
            else
                fileNameForZip = fi.FullName;
        }
        else
        {
            // Nur den Dateinamen speichern
            fileNameForZip = fi.Name;
        }
        ZipEntry zipEntry = new ZipEntry(fileNameForZip);

        // ZipEntry-Objekt dem ZipOutputStream hinzufügen
        zipOutputStream.PutNextEntry(zipEntry);
    }
}
```

Listing 208: Methode zum Erzeugen eines Zip-Archivs über *#ziplib*

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

// Zu archivierende Datei in einem FileStream öffnen
FileStream fileStream = new FileStream(sourceFileNames[i],
    FileMode.Open, FileAccess.Read);

// Quellstream blockweise in ein Byte-Array lesen und in den
// ZipOutputStream schreiben
byte[] buffer = new byte[blockSize];
int bytesWritten = 0;
do
{
    int size = fileStream.Read(buffer, 0, buffer.Length);
    zipOutputStream.Write(buffer, 0, size);
    bytesWritten += size;

} while (bytesWritten < fileStream.Length);

// FileStream schließen
fileStream.Close();
}

// ZipOutputStream abschließen und schließen
zipOutputStream.Finish();
zipOutputStream.Close();
}

```

Listing 208: Methode zum Erzeugen eines Zip-Archivs über #ziplib (Forts.)

Das folgende Beispiel zeigt die Anwendung dieser Methode. Ausnahmen, die beim Erzeugen des Archivs auftreten, werden dabei abgefangen. Das Beispiel benötigt neben den Voraussetzungen der ZipFiles-Methode den Import des Namensraums `System.IO`.

```

// Dateiname der Zip-Datei
string zipFileName = "c:\\demo.zip";

// Dateien, die archiviert werden sollen
string[] sourceFiles = {"c:\\inetpub\\wwwroot\\winxp.gif",
    "c:\\inetpub\\wwwroot\\pagerror.gif",
    "c:\\inetpub\\wwwroot\\warning.gif", "c:\\config.sys"};

try
{
    // Dateien mit einer Blockgröße von 1 MB und einer Kompressionsrate von 9
    // archivieren. Pfade sollen mit eingeschlossen werden
    ZipFiles(sourceFiles, zipFileName, 1048576, 9, true,
        "Demo für die Verwendung von #ziplib");

    Console.WriteLine("Fertig");
}
catch (Exception ex)

```

Listing 209: Anwendung der Methode ZipFiles zum Archivieren von Dateien

```
{
    Console.WriteLine("Fehler beim Archivieren: {0}", ex.Message);
}
```

Listing 209: Anwendung der Methode `ZipFiles` zum Archivieren von Dateien (Forts.)

127 (ZIP-)Archive aus einem Ordner erzeugen

Über `#ziplib` (siehe vorhergehendes Rezept) können Sie auch recht einfach ganze Ordner archivieren. Dazu müssen Sie lediglich den Ordner rekursiv durchgehen und die Dateien aller enthaltenen Unterordner separat archivieren. Um die Ordnerstruktur beim Entpacken beizubehalten, sollten Sie die Datei-Einträge im Archiv mit relativen Pfadangaben speichern. Die im Folgenden beschriebene Methode `ZipFolder` ruft dazu die rekursive Methode `AddFilesFromFolder` auf, die die Dateien eines Ordners archiviert und sich dann selbst für alle Unterordner aufruft.

`ZipFolder` erwartet als Argumente zunächst den Ordnernamen und den Namen der zu erzeugenden Archivdatei. Am dritten Argument übergeben Sie wie im vorhergehenden Rezept eine Blockgröße, die `ZipFolder` zum blockweisen Lesen und Schreiben der Streams verwendet. Mit einer nicht zu großen Blockgröße verhindern Sie Arbeitsspeicherprobleme bei sehr großen Dateien.

Das Argument `zipLevel` definiert die Kompressionsrate mit einem Wert zwischen 0 (keine Kompression) und 9 (maximale Kompression).

Am letzten Argument können Sie dann noch einen Kommentar für die Zip-Datei übergeben.

`ZipFolder` erfordert die Referenzierung der Assembly `ICSharpCode.SharpZipLib.dll` und den Import der Namensräume `System`, `System.IO` und `ICSharpCode.SharpZipLib.Zip`.

```
public static void ZipFolder(string folderName, string zipFileName,
    int blockSize, int zipLevel, string comment)
{
    // Datei-Stream als Basis-Stream erzeugen
    Stream zipFileStream = File.Open(zipFileName, FileMode.CreateNew);

    // ZipOutputStream zum Schreiben der Zip-Datei erzeugen
    ZipOutputStream zipOutputStream = new ZipOutputStream(zipFileStream);

    // Kompressionsrate definieren (0 bis 9)
    zipOutputStream.SetLevel(zipLevel);

    // Kommentar zum Archiv definieren
    zipOutputStream.SetComment(comment);

    // Ordner rekursiv archivieren
    AddFilesFromFolder(folderName, folderName, zipOutputStream, blockSize);
}
```

Listing 210: Methode zum rekursiven Archivieren eines Ordners über `#ziplib`

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    // ZipOutputStream abschließen und schließen
    zipOutputStream.Finish();
    zipOutputStream.Close();
}

/* Rekursive Methode zum Hinzufügen aller Dateien eines Ordners */
private static void AddFilesFromFolder(string baseFolderName,
    string folderName, ZipOutputStream zipOutputStream, int blockSize)
{
    // Alle Dateien des Ordners durchgehen und in das Archiv schreiben
    DirectoryInfo folder = new DirectoryInfo(folderName);
    FileInfo[] files = folder.GetFiles();
    for (int i = 0; i < files.Length; i++)
    {
        // Relativen Pfad des aktuellen Ordners über das Entfernen des
        // Basisordnersnamens ermitteln
        string relativePath = folderName.Replace(baseFolderName, "");
        if (relativePath != null)
        {
            if (relativePath.StartsWith("\\\\"))
                relativePath = relativePath.Remove(0, 1);
            if (relativePath.EndsWith("\\\\") == false)
                relativePath += "\\";
        }

        // ZipEntry-Objekt für die neue Datei mit dem relativen Pfad
        // erzeugen und dem ZipOutputStream hinzufügen
        zipOutputStream.PutNextEntry(
            new ICSharpCode.SharpZipLib.Zip.ZipEntry(
                relativePath + files[i].Name));

        // Zu archivierende Datei in einem FileStream öffnen und über ein
        // Byte-Array blockweise in den ZipOutputStream schreiben
        FileStream fileStream = files[i].OpenRead();
        byte[] buffer = new byte[blockSize];
        int bytesWritten = 0;
        do
        {
            int size = fileStream.Read(buffer, 0, buffer.Length);
            zipOutputStream.Write(buffer, 0, size);
            bytesWritten += size;
        } while (bytesWritten < fileStream.Length);

        // FileStream schließen
        fileStream.Close();
    }

    // Alle Unterordner durchgehen und die Methode rekursiv aufrufen
    DirectoryInfo[] subFolders = folder.GetDirectories();
    for (int i = 0; i < subFolders.Length; i++)

```

Listing 210: Methode zum rekursiven Archivieren eines Ordners über #ziplib (Forts.)


```

    AddFilesFromFolder(baseFolderName, subFolders[i].FullName,
        zipOutputStream, blockSize);
}
}

```

Listing 210: Methode zum rekursiven Archivieren eines Ordners über #ziplib (Forts.)

Listing 211 zeigt eine beispielhafte Anwendung der ZipFolder-Methode.

```
ZipFolder("c:\\inetpub", "c:\\demo.zip", 1048576, 9, "");
```

Listing 211: Archivieren des Ordners c:\\inetpub mit einer Blockgröße von einem MB und einer Komprimierung von 9

Das rekursive Archivieren eines Ordners über *ZipFolder* funktioniert sehr gut (Abbildung 81), ist allerdings wesentlich langsamer als das Archivieren mit einem professionellen Archivier-Programm wie WinZip. Vielleicht können Sie ja noch ein wenig optimieren ...

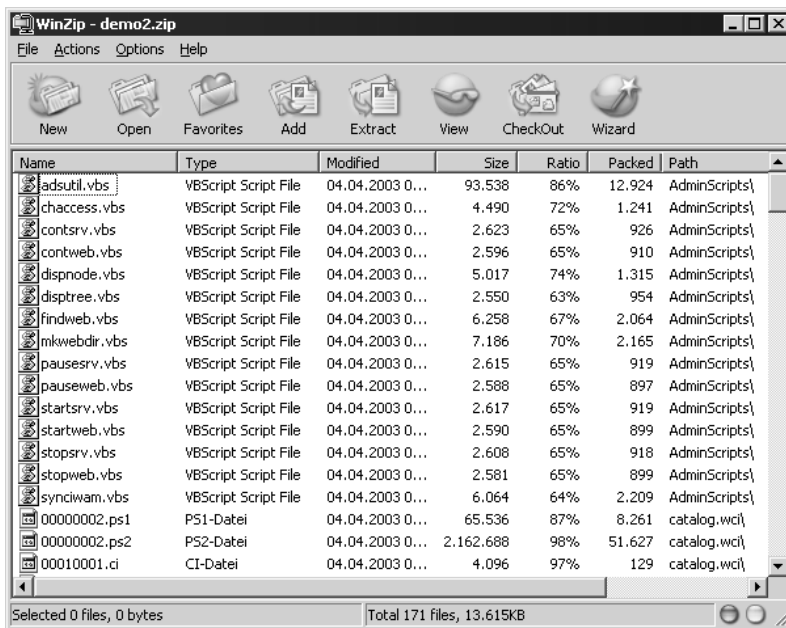


Abbildung 81: Ein mit #ziplib rekursiv archivierter Ordner in Winzip

128 (ZIP-)Archive aus beliebigen Daten erzeugen

Die Komponente *#ziplib* (siehe Rezept 126) arbeitet mit Streams. Deswegen ist es sehr einfach, ein Archiv aus einem Stream oder aus einem Byte-Array zu erzeugen. So können Sie prinzipiell alle Daten in ein Archiv packen.

Das Beispiel in Listing 212 demonstriert dies an Hand einer aus dem Internet geladenen Datei und eines Strings, die in ein Archiv gepackt werden.

Zum Kompilieren des Programms müssen Sie die Assembly *ICSharpCode.SharpZipLib.dll* referenzieren und die Namensräume *System*, *System.IO*, *System.Net*, *System.Text* und *ICSharpCode.SharpZipLib.Zip* importieren.

```
// Dateiname der Zip-Datei
string zipFileName = "c:\\demo3.zip";

// ZipOutputStream erzeugen
ZipOutputStream zipOutputStream = new ZipOutputStream(
    File.Open(zipFileName, FileMode.CreateNew));

// Kompressionsrate definieren (0 bis 9)
zipOutputStream.SetLevel(9);

// PDF-Datei aus dem Internet in ein Byte-Array herunterladen
string pdfUrl = "http://www.juergen-bayer.net/referenzen/CSharp-Referenz.pdf";
WebClient webClient = new WebClient();
byte[] buffer = webClient.DownloadData(pdfUrl);

// ZipEntry-Objekt für die neue Datei erzeugen, dem ZipOutputStream hinzufügen
// und den Byte-Puffer in den Stream schreiben
ZipEntry zipEntry = new ZipEntry("CSharp-Referenz.pdf");
zipOutputStream.PutNextEntry(zipEntry);
zipOutputStream.Write(buffer, 0, buffer.Length);

// String definieren
string demo = "Das ist eine Demo für das Schreiben von Daten in ein Archiv";

// Byte-Array mit der System-Default-Codierung aus dem String erzeugen
// und an das Archiv anhängen
buffer = Encoding.Default.GetBytes(demo);
zipEntry = new ZipEntry("Demo.txt");
zipOutputStream.PutNextEntry(zipEntry);
zipOutputStream.Write(buffer, 0, buffer.Length);

// ZipOutputStream abschließen und schließen
zipOutputStream.Finish();
zipOutputStream.Close();
```

Listing 212: Packen von beliebigen Daten in ein Zip-Archiv

129 Dateiinformatoren aus (ZIP-)Archiven auslesen

Um Informationen über die in einem Zip-Archiv enthaltenen Dateien auszulesen, können Sie (natürlich) die Komponente *#ziplib* verwenden (siehe Rezept 126). Dazu erzeugen Sie zunächst eine *ZipInputStream*-Instanz zum Einlesen der Archivdatei. Wie schon beim Erzeugen eines Archivs, übergeben Sie im Konstruktor einen normalen Stream, der die zu lesende Datei repräsentiert. Dann können Sie über die Methode *GetNextEntry* in einer Schleife das

jeweils nächste ZipEntry-Objekt auslesen. ZipEntry-Objekte speichern die Informationen zu einer im Archiv gespeicherten Datei. Über die Eigenschaft `Name` können Sie den Dateinamen auslesen, die Eigenschaft `Size` verwaltet die unkomprimierte Größe der Datei, aus der Eigenschaft `CompressedSize` können Sie die komprimierte Größe auslesen. ZipEntry-Objekte besitzen noch weitere Eigenschaften, von denen ich die wichtigsten in Listing 213 verwende.

Das Beispiel erfordert die Referenzierung der Assembly `ICSharpCode.SharpZipLib.dll` und den Import der Namensräume `System` und `ICSharpCode.SharpZipLib.Zip`.

```
string zipFileName = "c:\\demo.zip";

// ZipInputStream für die Zip-Datei erzeugen
ZipInputStream zipInputStream = new ZipInputStream(
    File.Open(zipFileName, FileMode.Open, FileAccess.Read));

// Alle im Archiv gespeicherten ZipEntry-Objekte durchgehen
ZipEntry zipEntry;
while ((zipEntry = zipInputStream.GetNextEntry()) != null)
{
    Console.WriteLine("Name : {0}", zipEntry.Name);
    Console.WriteLine("Größe : {0}", zipEntry.Size);
    Console.WriteLine("Komprimierte Größe : {0}", zipEntry.CompressedSize);
    Console.WriteLine("Kompressionsmethode : {0}", zipEntry.CompressionMethod);
    Console.WriteLine("Dateidatum : {0}", zipEntry.DateTime.ToString());
    Console.WriteLine("Kommentar : {0}", zipEntry.Comment);
    Console.WriteLine("CRC : {0}", zipEntry.Crc);
    Console.WriteLine();
}

// ZipInputStream schließen
zipInputStream.Close();
```

Listing 213: Auslesen von Informationen über die in einem Zip-Archiv gespeicherten Dateien

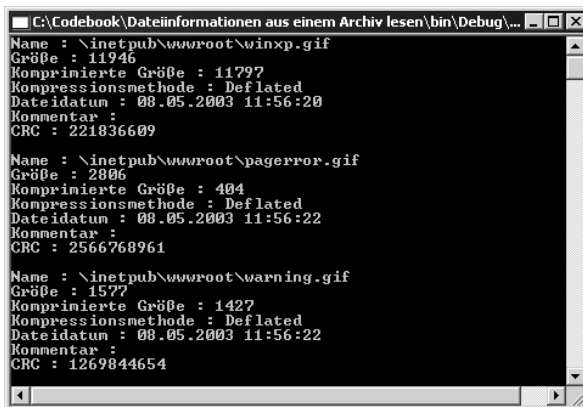


Abbildung 82: Das Beispielprogramm zum Auslesen von Informationen über die in einem Zip-Archiv gespeicherten Dateien

130 (ZIP-)Archive entpacken

Das Entpacken von Zip-Archiven ist über `#ziplib` (siehe Rezept 126) ebenso einfach wie das Verpacken. Wie im vorhergehenden Rezept erzeugen Sie dazu eine Instanz der Klasse `ZipInputStream` und gehen über die `GetNextEntry`-Methode alle `ZipEntry`-Objekte durch. Über diese Objekte ermitteln Sie den Dateinamen der jeweiligen im Archiv gespeicherten Datei und den eventuell angegebenen Unterordner. Ist ein Ordner angegeben, sollten Sie diesen zunächst im Zielordner anlegen. Da der `ZipInputStream` nach dem Lesen eines `ZipEntry`-Objekts auf dem Anfang der Datei steht, können Sie diese danach über die `Read`-Methode des Streams in ein Byte-Array lesen. Dabei brauchen Sie sich natürlich nicht um das Dekomprieren der gepackten Daten zu kümmern.

ACHTUNG

Die `Read`-Methode liest laut Mike Krüger, dem Entwickler der Komponente, nicht immer alle Daten, die noch im Stream gespeichert sind² (was aber in der Version 0.5 nicht dokumentiert ist). Bei meinen Versuchen betraf dies eine Zip-Datei, die eine Text- und eine JPEG-Datei speicherte. Beim Lesen der kompletten Daten eines `ZipEntry`-Objekts in einem Rutsch wurde die zweite Datei nicht komplett eingelesen. Um dieses »Feature« zu vermeiden, müssen Sie die Daten immer in einer Schleife einlesen. `Read` gibt die Anzahl der gelesenen Bytes zurück. Die Schleife muss also ausgeführt werden, bis diese Methode 0 zurückgibt.

Das Lesen in einer Schleife besitzt noch einen anderen Vorteil. Bei sehr großen Dateien können Sie diese so in kleineren Teilen einlesen, um den Arbeitsspeicher nicht zu überlasten. Die einzelnen Byte-Blöcke können Sie dann in ein `FileStream`-Objekt schreiben um eine Datei zu erzeugen.

Die Methode `ExtractToFolder` in Listing 214 extrahiert alle Dateien eines Archivs in einen Ordner. Die Dateien werden dabei in Blöcken einer festzulegenden Größe aus dem Archiv gelesen. Um bereits vorhandene Dateien nicht ohne Warnung zu überschreiben, fragt `ExtractToFolder` in diesem Fall beim Anwender nach, ob er diese überschreiben will. Antwortet der Anwender mit Ja, fragt die Methode einmalig nach, ob alle Dateien überschrieben werden sollen. Diese Überschreib-Logik stammt aus dem Rezept 94. Die Nachfrage kann über das Argument `overwriteWithoutWarning` auch abgeschaltet werden.

`ExtractToFolder` erfordert die Referenzierung der Assemblies `ICSharpCode.SharpZipLib.dll` und `System.Windows.Forms.dll` und den Import der Namensräume `System`, `System.IO`, `System.Windows.Forms` und `ICSharpCode.SharpZipLib.Zip`.

```
/* Eigenschaften, die für das Überschreiben aller Dateien verwendet werden */
private static bool overwriteAllFiles;
private static bool alreadyAskedForOverwriteAllFiles;
```

Listing 214: Methode zum Extrahieren eines Zip-Archivs über `#ziplib`

2. Dies hat mir Mike Krüger per E-Mail bestätigt

```

/* Methode zum Extrahieren eines Zip-Archivs in einen Ordner */
public static void ExtractToFolder(string zipFileName, string folderName,
    int blockSize, bool overwriteWithoutWarning)
{
    // Eigenschaften voreinstellen
    overwriteAllFiles = false;
    alreadyAskedForOverwriteAllFiles = false;

    // ZipInputStream für die Zip-Datei erzeugen
    ZipInputStream zipInputStream = new ZipInputStream(
        File.Open(zipFileName, FileMode.Open, FileAccess.Read));

    // Alle im Archiv gespeicherten ZipEntry-Objekte durchgehen
    ZipEntry zipEntry;
    while ((zipEntry = zipInputStream.GetNextEntry()) != null)
    {
        // Aus dem (relativen) Dateinamen den Unterordner und den
        // Dateinamen extrahieren
        string subFolderName = Path.GetDirectoryName(zipEntry.Name);
        string fileName = Path.GetFileName(zipEntry.Name);

        // Den vollen Ordnernamen des Zielordners ermitteln
        string destFolderName = Path.Combine(folderName, subFolderName);

        // Unterordner erzeugen, falls notwendig
        if (subFolderName != null)
            Directory.CreateDirectory(destFolderName);

        if (fileName != null)
        {
            bool overwriteFile = true;

            if (overwriteWithoutWarning == false &&
                overwriteAllFiles == false)
            {
                // Wenn Dateien nicht ohne Warnung überschrieben werden
                // sollen: Überprüfen, ob bereits eine gleichnamige Datei
                // existiert
                if (File.Exists(Path.Combine(destFolderName, fileName)))
                {
                    // Nachfragen, ob die Datei überschrieben werden soll
                    switch (MessageBox.Show("Die Datei '" + fileName +
                        "' existiert bereits im Ordner '" + destFolderName +
                        "'\r\n\r\nSoll diese Datei überschrieben werden?",
                        Application.ProductName,
                        MessageBoxButtons.YesNoCancel,
                        MessageBoxIcon.Question))
                    {
                        case DialogResult.Yes:
                            overwriteFile = true;
                            break;
                    }
                }
            }
        }
    }
}

```

Listing 214: Methode zum Extrahieren eines Zip-Archivs über #ziplib (Forts.)

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        case DialogResult.No:
            overwriteFile = false;
            break;
        case DialogResult.Cancel:
            // Stream schließen und beenden
            zipInputStream.Close();
            return;
    }

    // Nachfragen, ob alle Dateien überschrieben werden
    // sollen, sofern dies noch nicht geschehen ist
    if (overwriteFile == true &&
        alreadyAskedForOverwriteAllFiles == false)
    {
        switch (MessageBox.Show("Sollen alle vorhandenen " +
            "Dateien automatisch überschrieben werden?",
            Application.ProductName,
            MessageBoxButtons.YesNoCancel,
            MessageBoxIcon.Question))
        {
            case DialogResult.Yes:
                overwriteAllFiles = true;
                break;
            case DialogResult.No:
                overwriteAllFiles = false;
                break;
            case DialogResult.Cancel:
                // Stream schließen und beenden
                zipInputStream.Close();
                return;
        }
        // Definieren, dass nicht noch einmal gefragt wird
        alreadyAskedForOverwriteAllFiles = true;
    }
}

if (overwriteFile)
{
    // FileStream für die Datei erzeugen
    FileStream fileStream =
        File.Create(Path.Combine(destFolderName, fileName));

    // Datei in Blöcken von maximal 1 MB in den Stream schreiben
    // um den Speicher nicht mit großen Dateien zu überlasten
    int size;
    byte[] buffer = new byte[1048576];
    do
    {
        // Den nächsten Datenblock aus dem ZipInputStream lesen
        size = zipInputStream.Read(buffer, 0, buffer.Length);
    }
}

```

Listing 214: Methode zum Extrahieren eines Zip-Archivs über #ziplib (Forts.)

```

        if (size > 0)
            // Wenn Daten gelesen wurden, diese in die Datei
            // schreiben
            fileStream.Write(buffer, 0, size);
    } while (size > 0);

    // FileStream schließen
    fileStream.Close();
}
}

// ZipInputStream schließen
zipInputStream.Close();
}

```

Listing 214: Methode zum Extrahieren eines Zip-Archivs über #ziplib (Forts.)

Die Anwendung dieser Methode ist wieder einmal sehr einfach:

```

string zipFileName = "c:\\demo.zip";
string folderName = "c:\\temp";

// Zip-Archiv mit einer Blockgröße von einem MB mit Überschreib-Nachfrage
// entpacken
try
{
    ExtractToFolder(zipFileName, folderName, 1048576, false);

    Console.WriteLine("Fertig");
}
catch (ZipException ex1)
{
    Console.WriteLine("Fehler bei der Archivoperation: {0}", ex1.Message);
}
catch (IOException ex2)
{
    Console.WriteLine("Fehler bei der Dateioperation: {0}", ex2.Message);
}
catch (Exception ex3)
{
    Console.WriteLine("Allgemeiner Fehler: {0}", ex3.Message);
}

```

Listing 215: Anwendung der Methode zum Extrahieren eines Zip-Archivs

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

131 Den Inhalt von (ZIP-)Archiven lesen

Wenn Sie ein Zip-Archiv nicht in Datei(en) entpacken, sondern lediglich den Inhalt lesen wollen, können Sie dazu einfach ein `ZipInputStream`-Objekt verwenden und den Stream weiter verarbeiten. Das folgende Beispiel zeigt dies an Hand eines Archivs, das eine Text- und eine Grafikdatei enthält. Der Text wird in eine `TextBox` eingelesen, die Grafik in ein `PictureBox`-Steuerelement. Das Formular zeigt Abbildung 83.

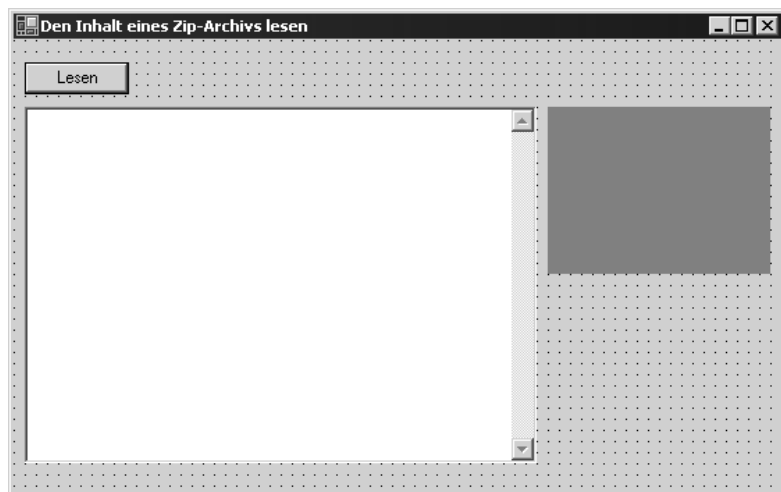


Abbildung 83: Das Beispielformular zum Lesen des Inhalts eines Zip-Archivs

Die Ereignisbehandlungsmethode für das Click-Ereignis des Lesen-Schalters liest die Zip-Datei *demo.zip* ein, die im Ordner der Anwendung erwartet wird. Für jedes darin enthaltene `ZipEntry`-Objekt werden zunächst die Daten in ein `byte-Array` eingelesen. Etwas problematisch ist, dass die `Read`-Methode nicht immer alle noch im Stream gespeicherten Daten einliest (siehe vorhergehendes Rezept). Deshalb muss das Lesen in einer Schleife erfolgen. Da ich ein `byte-Array` benötige, um daraus einen String oder ein `Bitmap`-Objekt zu erzeugen, verwende ich zwei dieser Arrays, eines zum Lesen der einzelnen Datenblöcke und eines, in das ich die jeweils gelesenen Daten hinten anhänge.

Nach dem Lesen überprüft das Programm, ob es sich um eine Text- oder eine Grafikdatei handelt. Für Textdateien wandelt das Programm das eingelesene `Byte-Array` über `Encoding.ASCII.GetString` in einen String um. `Bitmap`-Grafiken können über den Konstruktor der `Bitmap`-Klasse direkt aus einem Stream erzeugt werden. Leider funktioniert das nicht mit einem `ZipInputStream` (in diesem Fall erzeugt die CLR eine Ausnahme vom Typ `ArgumentException` mit der Meldung »Ungültiger Parameter verwendet«). Deshalb erzeuge ich aus dem eingelesenen `byte-Array` einen `MemoryStream` und übergebe diesen an den Konstruktor der `Bitmap`-Klasse.

Das Beispiel benötigt die Referenzierung der Assemblies *System.Windows.Forms.dll* und *ICSharpCode.SharpZipLib.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Drawing`, `System.Windows.Forms` und `ICSharpCode.SharpZipLib.Zip`.


```
private void readButton_Click(object sender, System.EventArgs e)
{
    // Dateiname der Zip-Datei
    string zipFileName = Path.Combine(Application.StartupPath, "demo.zip");

    // Zip-Datei öffnen
    ZipInputStream zipInputStream = new ZipInputStream(
        new FileStream(zipFileName, FileMode.Open, FileAccess.Read));

    // Alle Dateien durchgehen
    ZipEntry zipEntry;
    while ((zipEntry = zipInputStream.GetNextEntry()) != null)
    {
        // Daten in einer Schleife (blockweise) einlesen
        byte[] data = new byte[zipEntry.Size];
        int dataIndex = 0;
        byte[] buffer = new byte[1024];
        int count = 0;
        do
        {
            // Daten lesen
            count = zipInputStream.Read(buffer, 0, buffer.Length);

            // Das eingelesene Array an das data-Array hinten anhängen
            Array.Copy(buffer, 0, data, dataIndex, count);

            // Start-Index für das data-Array hochzählen
            dataIndex += count;
        } while (count > 0);

        // Überprüfen, ob es sich um eine Textdatei handelt
        FileInfo fi = new FileInfo(zipEntry.Name);
        if (fi.Extension == ".txt")
        {
            // Daten in einen Text konvertieren
            string text = Encoding.ASCII.GetString(data);
            this.zipTextBox.Text = text;
        }

        // Überprüfen, ob es sich um eine Bitmap-Grafik handelt
        if (fi.Extension == ".jpg" || fi.Extension == ".jpeg"
            || fi.Extension == ".gif" || fi.Extension == ".bmp")
        {
            // Daten in eine Bitmap-Grafik umwandeln
            Bitmap bitmap = new Bitmap(new MemoryStream(data));
            this.zipPictureBox.Image = bitmap;
        }
    }
}
```

Listing 216: Methode für das Click-Ereignis des Lesen-Schalters der Beispielanwendung zum Lesen eines Zip-Archivs

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// ZipInputStream schließen  
zipInputStream.Close();  
}
```

Listing 216: Methode für das Click-Ereignis des Lesen-Schalters der Beispielanwendung zum Lesen eines Zip-Archivs (Forts.)

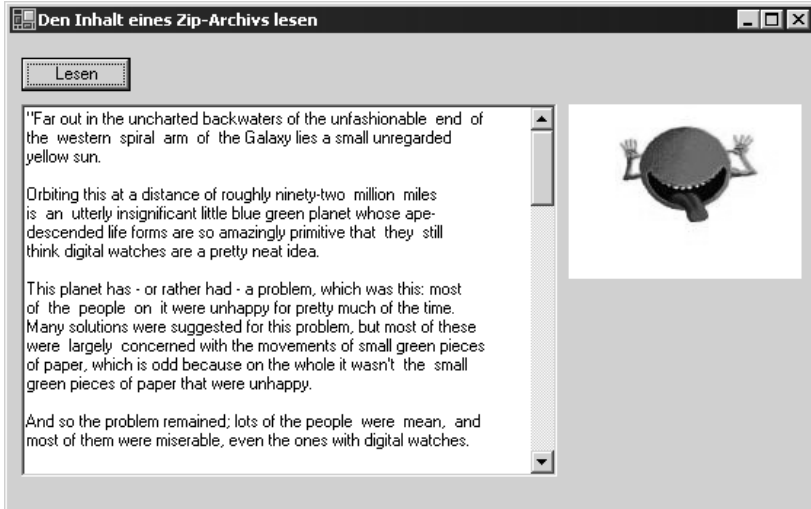


Abbildung 84: Das Programm hat die Daten aus der Zip-Datei eingelesen

132 XML-Dateien über ein `XmlTextReader`-Objekt lesen

Über eine Instanz der Klasse `System.Xml.XmlTextReader` können Sie eine XML-Datei sehr einfach einlesen. Ein `XmlTextReader`-Objekt erlaubt ein einmaliges, schreibgeschütztes Durchgehen der einzelnen XML-Knoten. Da dabei keine Daten im Arbeitsspeicher gehalten werden, benötigt diese Klasse sehr wenig Ressourcen und ermöglicht ein schnelles Lesen. Wenn Sie XML-Daten nur sequenziell lesen und nicht in diesen Daten suchen wollen, ist ein `XmlTextReader`-Objekt die erste Wahl (ansonsten verwenden Sie die DOM-Variante, die ich im nächsten Rezept vorstelle).

Hinweis

Die Verwendung einer `XmlTextReader`-Instanz hat übrigens viel Ähnlichkeit mit SAX (Simple API for XML). Der Unterschied ist, dass Sie bei der Verwendung eines `XmlTextReader`-Objekts die Daten über Methoden explizit auslesen (was als »Pull« bezeichnet wird). SAX hingegen geht ein XML-Dokument intern durch und informiert das Programm über Ereignisse, dass ein neuer XML-Knoten erreicht wurde (»Push«). Wahrscheinlich ist keine dieser Methoden besser als die andere. Auf jeden Fall sind beide ressourcenschonend. SAX wird vom .NET Framework nicht unterstützt. Sie können jedoch spezielle XML-Komponenten wie Microsofts MSXML-Parser einsetzen, wenn Sie SAX verwenden wollen.

Zum Lesen einer XML-Datei über die Pull-Methode müssen Sie zunächst eine Instanz der `XmlTextReader`-Klasse (aus dem Namensraum `System.Xml`) erzeugen. Im (mehrfach überladenen) Konstruktor können Sie den Dateinamen einer XML-Datei oder einen URL angeben, der eine XML-Datei im Internet bezeichnet:

```
xmlReader = new XmlTextReader(xmlFileName);
```

Nun rufen Sie die `Read`-Methode in einer Schleife auf und verarbeiten die XML-Daten im Schleifenkörper:

```
while (xmlReader.Read())
{
    // Die XML-Daten verarbeiten
}
```

`Read` setzt das `XmlTextReader`-Objekt auf den jeweils nächsten Knoten. Die Daten dieses Knoten können Sie dann über Methoden und Eigenschaften des `XmlTextReader`-Objekts lesen.

Bei der Verarbeitung müssen Sie die verschiedenen XML-Knoten unterscheiden. XML kennt einige XML-Knotentypen, wie z. B. einen Start-Tag, die Textdaten eines Elements oder einen End-Tag. Wenn Sie eine XML-Datei gezielt durchgehen wollen, müssen Sie die Struktur der XML-Daten kennen und beim Durchgehen der einzelnen Knoten deren Typ und Name abfragen. Den Typ eines Knotens erhalten Sie über die Eigenschaft `NodeType` als Wert der Aufzählung `XmlNodeType`. Die wichtigsten `XmlNodeType`-Konstanten sind `Element` (Start-Tag), `EndElement` (Ende-Tag) und `Text` (der Text eines Elements). Den Namen des aktuellen Knotens können Sie aus der Eigenschaft `Name` auslesen.

Als Beispiel soll die folgende XML-Datei eingelesen werden, die Personendaten verwaltet:

```
<?xml version="1.0" encoding="utf-8" ?>
<persons>
  <person id="1000">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <type>Alien</type>
  </person>

  <person id="1001">
    <firstname>Ford</firstname>
    <lastname>Prefect</lastname>
    <type>Alien</type>
  </person>

  <person id="1002">
    <firstname>Tricia</firstname>
    <lastname>McMillan</lastname>
    <type>Earthling</type>
  </person>

  <person id="1003">
    <firstname>Arthur</firstname>
    <lastname>Dent</lastname>
    <type>Earthling</type>
  </person>
</persons>
```

Listing 217: Beispiel-XML-Datei mit den Daten von Personen

Das Beispielprogramm soll alle in der XML-Datei gespeicherten Personen in Instanzen einer eigenen Klasse `Person` einlesen, die in einer `ArrayList`-Auflistung verwaltet werden. Ich habe diesen Weg gewählt, um möglichst viel Praxisnähe in das Programm zu integrieren. Die Personen-Auflistung kann dann nach dem Lesen weiterverarbeitet werden. Das Beispiel gibt die eingelesenen Daten allerdings einfach nur an der Konsole aus.

Das Programm geht zum Einlesen der Personen die einzelnen Knoten der XML-Datei durch und überprüft zunächst, ob eine neue Person beginnt. Ist dies der Fall, erzeugt es eine neue Instanz der Klasse `Person`. Über die Methode `GetAttribute` wird dann das Attribut `id` des `person`-Elements eingelesen und im `Person`-Objekt gespeichert.

Für die folgenden Element-Knoten überprüft das Programm, um welches Element es sich handelt. Handelt es sich um einen der Start-Tags der Elemente, die die Personendaten verwalten, wechselt es über die *MoveToContent*-Methode zum nächsten Inhalts-Knoten, der in unserem Fall ein Text-Knoten ist. *MoveToContent* überspringt Kommentare und andere, für die eigentlichen Daten unwichtige Knoten und gibt den Knotentyp zurück, wenn (noch) ein Inhaltsknoten gefunden wurde. Über den Vergleich auf *XmlNodeType.None* wird überprüft, ob eventuell kein Inhaltsknoten mehr gefunden wurde.

Ausnahmen, die beim Öffnen der XML-Datei entstehen können (z.B. wenn die Datei nicht vorhanden oder exklusiv gesperrt ist), fängt das Beispiel ab. Da eine ungültige XML-Datei (z.B. eine mit fehlenden Ende-Tags) erst beim Lesen der Daten erkannt wird, werden auch die entsprechenden Ausnahmen abgefangen um dem Anwender (und dem Programmierer) eine Fehlermeldung ausgeben zu können.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume *System*, *System.Xml*, *System.IO*, *System.Collections* und *System.Windows.Forms* einbinden.

```

/* Klasse zur Speicherung der Daten einer Person */
class Person
{
    public string Id;
    public string FirstName;
    public string LastName;
    public string Type;
}

/* Start-Klasse */
class Start
{
    [STAThread]
    static void Main(string[] args)
    {
        // Der Dateiname
        string xmlFileName = Path.Combine(Application.StartupPath,
            "Persons.xml");

        // XmlTextReader erzeugen
        XmlTextReader xmlReader = null;
        try
        {
            xmlReader = new XmlTextReader(xmlFileName);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Fehler beim Einlesen der XML-Datei '" +
                xmlFileName + "': " + ex.Message, Application.ProductName,
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
    }
}

```

Listing 218: Einlesen einer XML-Datei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    }

    // Auflistung für die Personen erzeugen
    ArrayList persons = new ArrayList();

    // Die einzelnen Knoten einlesen und durchgehen
    Person person = null;
    try
    {
        while (xmlReader.Read())
        {
            // Überprüfen, ob es sich beim aktuellen Knoten um einen
            // Start-Tag handelt
            if (xmlReader.NodeType == XmlNodeType.Element)
            {
                // Überprüfen, ob es sich um ein person-Element handelt
                if (xmlReader.Name == "person")
                {
                    // Neue Person: Neues Person-Objekt erzeugen ...
                    person = new Person();

                    // ... und in der Auflistung ablegen
                    persons.Add(person);

                    // Das Attribut 'id' einlesen
                    person.Id = xmlReader.GetAttribute("id");
                }
                else
                {
                    // Überprüfen, um welches Element es sich handelt
                    if (xmlReader.Name == "firstname")
                    {
                        // Zum Inhalt wechseln und den Text einlesen
                        if (xmlReader.MoveToContent() != XmlNodeType.None)
                            person.FirstName = xmlReader.ReadString();
                    }
                    else if (xmlReader.Name == "lastname")
                    {
                        // Zum Inhalt wechseln und den Text einlesen
                        if (xmlReader.MoveToContent() != XmlNodeType.None)
                            person.LastName = xmlReader.ReadString();
                    }
                    else if (xmlReader.Name == "type")
                    {
                        // Zum Inhalt wechseln und den Text einlesen
                        if (xmlReader.MoveToContent() != XmlNodeType.None)
                            person.Type = xmlReader.ReadString();
                    }
                }
            }
        }
    }

```

Listing 218: Einlesen einer XML-Datei (Forts.)

```
}  
catch (Exception ex)  
{  
    MessageBox.Show("Fehler beim Einlesen der XML-Datei '" +  
        xmlFileName + "': " + ex.Message, Application.ProductName,  
        MessageBoxButtons.OK, MessageBoxIcon.Error);  
    xmlReader.Close();  
    return;  
}  
  
// XmlTextReader schließen  
xmlReader.Close();  
  
// Alle eingelesenen Personen durchgehen und an der Konsole ausgeben  
foreach (Person p in persons)  
{  
    Console.WriteLine("Person {0}", p.Id);  
    Console.WriteLine("Vorname: {0}", p.FirstName);  
    Console.WriteLine("Nachname: {0}", p.LastName);  
    Console.WriteLine("Typ: {0}", p.Type);  
    Console.WriteLine();  
}  
}
```

Listing 218: Einlesen einer XML-Datei (Forts.)

Abbildung 85 zeigt die Konsolenausgabe des Programms, Abbildung 86 die Meldung beim Einlesen einer XML-Datei mit einem fehlenden Ende-Tag für das `firstname`-Element.

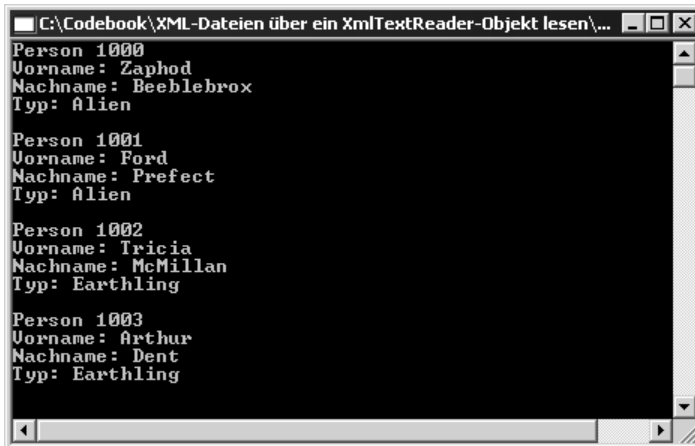


Abbildung 85: Das Programm zum Einlesen einer XML-Datei in Aktion

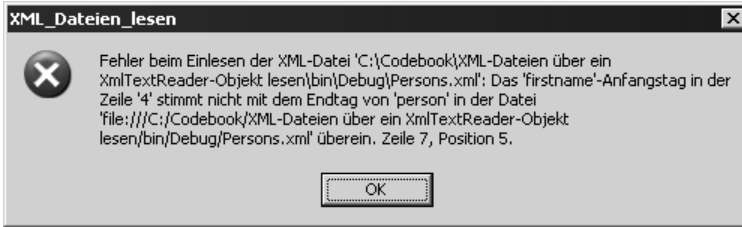


Abbildung 86: Fehlermeldung beim Einlesen einer XML-Datei mit einem fehlenden Ende-Tag für das `firstname`-Element

Hinweis

Entsprechend der XML-Empfehlung des W3C¹ sollten Sie das Lesen einer XML-Datei komplett abbrechen, wenn diese fehlerhaft ist. Nur so genannte wohlgeformte (und damit fehlerfreie) XML-Dateien sollten demnach verarbeitet werden.

133 XML-Dateien beim Einlesen gegen ein Schema (oder DTD) prüfen

XML-Schemata legen die Struktur einer XML-Datei fest. Unter anderem kann in einem Schema der Datentyp von XML-Elementen festgelegt oder definiert sein, dass und wie oft bestimmte Elemente unterhalb anderer Elemente vorkommen müssen bzw. dürfen.

Hinweis

DTD (Document Type Definition) ist eine ältere Sprache zur Definition der Struktur von XML-Dokumenten. XML-Schemata sind die zurzeit aktuelle Variante. Schemata erlauben wesentlich mehr Festlegungen, wie z.B. die Bestimmung des genauen Datentyps von XML-Elementen, was bei DTDs nicht möglich ist. DTDs und Schemata können in XML-Dokumenten eingebettet sein oder in externen Dateien vorliegen. Extern vorliegende Schemata oder DTDs können in einem XML-Dokument über festgelegte XML-Instruktionen mit dem Dokument verbunden werden. Schauen Sie sich die XML-Dateien im Beispiel zum Rezept 134 an, wenn Sie sehen wollen, wie dies in der Praxis aussieht.

Tipp

Hervorragende Informationen zu DTDs finden Sie bei den W3Schools an der Adresse www.w3schools.com/dtd/default.asp, Schemata werden an der Adresse www.w3schools.com/schema/default.asp behandelt.

1. Das World Wide Web Consortium (W3C, www.w3c.org) entwickelt und empfiehlt Technologien und Standards für das Internet, wie z.B. HTML, XML und Cascading Style Sheets

Ein Schema für die Personen-XML-Datei aus dem Rezept 132 könnte z.B. folgendermaßen aussehen:

```
<?xml version="1.0" standalone="yes"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  id="persons">
  <xs:element name="persons">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="firstname" type="xs:string"/>
              <xs:element name="lastname" type="xs:string"/>
              <xs:element name="type" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:int" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 219: Beispiel-Schema für die Personen-XML-Datei

Das Beispiel-Schema legt fest, dass das Wurzel-Element `persons` ein komplexer Typ (mit Unterelementen) ist, der beliebig viele `person`-Elemente enthalten kann. Ein `person`-Element ist ebenfalls als komplexer Typ definiert, das die Elemente `firstname`, `lastname` und `type` genau ein Mal enthalten muss. Für die einzelnen Unterelemente ist jeweils der Datentyp festgelegt. Schließlich wird noch definiert, dass das Element `person` ein Attribut besitzen muss, das `id` heißt und das den (XML-)Typ `int` besitzt.

Um eine XML-Datei beim Einlesen gegen ein Schema zu prüfen, verwenden Sie eine Instanz der Klasse `XmlValidatingReader`. Damit können Sie eine XML-Datei wie mit einer Instanz der `XmlTextReader`-Klasse einlesen (siehe vorhergehendes Rezept). Ein `XmlValidatingReader` ermöglicht aber die Überprüfung der eingelesenen Daten gegen ein DTD (Document Type Definition), gegen ein XML-Schema oder einfach nur auf Wohlgeformtheit der Daten.

Ein `XmlValidatingReader`-Objekt verwendet ein `XmlTextReader`-Objekt als Basis für das Lesen der Daten. Bei der Erzeugung einer `XmlValidatingReader`-Instanz müssen Sie ein `XmlTextReader`-Objekt im Konstruktor übergeben.

Die Eigenschaft `ValidationType` des `XmlValidatingReader`-Objekts bestimmt die Art der Validierung in Form der Werte der `ValidationType`-Aufzählung. Die einzelnen Werte dieser Aufzählung erläutere ich im Rezept 134. Wenn Sie gegen ein XML-Schema prüfen wollen, übergeben Sie hier den Wert `ValidationType.Schema`.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Ist das Schema nicht in der XML-Datei eingebettet, müssen Sie dieses dann noch der `Schemas`-Auflistung hinzufügen. Dazu verwenden Sie deren `Add`-Methode, der Sie eine vorhandene `XmlSchemaCollection`-Instanz, ein vorhandenes `XmlSchema`-Objekt, einen `XmlReader` oder einfach nur den Dateinamen einer Schemadatei übergeben können. In den letzten beiden Fällen können Sie am ersten Argument einen XML-Namensraum-URI² übergeben. Namensräume werden in XML verwendet, um XML-Elemente (die einem bestimmten Namensraum zugeordnet sind) von anderen XML-Elementen (die anderen Namensräumen zugeordnet sind) eindeutig zu unterscheiden. Der Namensraum, den Sie am ersten Argument der Methode `Schemas.Add` übergeben, ist der, der im Attribut `targetNamespace` der Schema-Datei angegeben ist (falls dieses vorhanden ist). Ist ein solcher Ziel-Namensraum angegeben, müssen die Elemente der zu validierenden XML-Datei diesem Namensraum zugeordnet sein. Damit wird sichergestellt, dass ein XML-Schema nur für bestimmte XML-Namensräume verwendet werden kann. Ist dieses Attribut nicht vorhanden, geben Sie einfach `null` an.

Wenn Sie nun eine `XmlValidatingReader`-Instanz zum Lesen verwenden, erzeugt diese beim Lesen eine Ausnahme vom Typ `XmlException`, wenn die eingelesenen Daten nicht wohlgeformt sind oder nicht dem Schema entsprechen.

Hinweis

Im nächsten Rezept stelle ich eine Klasse zum allgemeinen Validieren von XML-Dokumenten vor. Dort beschreibe ich, dass Sie auch dem `Validation`-Ereignis der `XmlValidatingReader`-Instanz eine passende Ereignisbehandlungsmethode zuweisen können. In diesem Fall erzeugt das `XmlValidatingReader`-Objekt bei Fehlern im Dokument keine Ausnahme, sondern ruft die zugewiesene Methode auf. Damit können Sie erreichen, dass das Lesen der XML-Datei nicht beim ersten Fehler abgebrochen wird, und somit alle Fehler eines Dokuments auswerten.

Listing 220 setzt einen `XmlValidatingReader` ein, um die Datei *Persons.xml* (Listing 217), die im Ordner der Anwendung erwartet wird, einzulesen und dabei gegen die Schema-Datei *Persons.xsd* (Listing 219) zu überprüfen. Zur Vereinfachung des Codes werden die eingelesenen Daten lediglich an der Konsole ausgegeben.

Um die Gültigkeit des Dokuments zu überprüfen, erfolgt das Einlesen in einer Ausnahmebehandlung. Damit später eine Aussage über die Gültigkeit getroffen werden kann, setze ich eine Variable `isValid` ein, die zunächst auf `true` und bei einer Ausnahme auf `false` gesetzt wird. Diese Variable wird am Ende des Einlesens abgefragt.

Das Beispiel erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Xml`, `System.Xml.Schema` und `System.Windows.Forms`.

2. Ein URI (Uniform Ressource Identifier) identifiziert Ressourcen (HTML-Dokumente, FTP-Dateien, Mailboxen etc.) im Internet. Die Adresse eines HTML-Dokuments oder die Adresse einer Mailbox sind Beispiele für URIs. URLs (Uniform Resource Locator) sind übrigens eine Unterordnung von URIs, die Dateien referenzieren.

```

// Der Dateiname der XML- und der Schema-Datei
string xmlFileName = Path.Combine(Application.StartupPath,
    "Persons.xml");
string xmlSchemaFileName = Path.Combine(Application.StartupPath,
    "Persons.xsd");

// XmlValidatingReader mit einem neuen XmlReader als Basis erzeugen
XmlValidatingReader xmlValidator = new XmlValidatingReader(
    new XmlTextReader(xmlFileName));

// Validier-Typ auf "Schema" festlegen
xmlValidator.ValidationType = ValidationType.Schema;

// Schema ohne Ziel-Namensraum hinzufügen
try
{
    xmlValidator.Schemas.Add(null, xmlSchemaFileName);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

// Merker für die Gültigkeit des Dokuments initialisieren
bool isValid = true;
try
{
    // Daten einlesen
    while (xmlValidator.Read())
    {
        // Überprüfen, ob der aktuelle Knoten ein Element ist
        if (xmlValidator.NodeType == XmlNodeType.Element)
        {
            // Überprüfen, um welches Element es sich handelt, und
            // die eingelesenen Daten entsprechend ausgeben
            if (xmlValidator.Name == "person")
            {
                Console.WriteLine();
                Console.WriteLine("Person {0}",
                    xmlValidator.GetAttribute("id"));
            }
            else if (xmlValidator.Name == "firstname")
            {
                if (xmlValidator.MoveToContent() != XmlNodeType.None)
                    Console.WriteLine("Vorname: {0}",
                        xmlValidator.ReadString());
            }
            else if (xmlValidator.Name == "lastname")
            {
                if (xmlValidator.MoveToContent() != XmlNodeType.None)
                    Console.WriteLine("Nachname: {0}",

```

Listing 220: Validierendes Einlesen einer XML-Datei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        xmlValidator.ReadString());
    }
    else if (xmlValidator.Name == "type")
    {
        if (xmlValidator.MoveToContent() != XmlNodeType.None)
            Console.WriteLine("Typ: {0}",
                xmlValidator.ReadString());
    }
}
}
}
catch (Exception ex)
{
    // Fehler beim Lesen: Fehler ausgeben und Merker für die
    // Gültigkeit des Dokuments zurücksetzen
    Console.WriteLine(ex.Message);
    isValid = false;
}

// XmlValidatingReader schließen
xmlValidator.Close();

// Ausgeben, ob die Datei gültig ist
Console.WriteLine();
if (isValid)
    Console.WriteLine("Die XML-Datei ist gültig");
else
    Console.WriteLine("Die XML-Datei ist nicht gültig");
}

```

Listing 220: Validierendes Einlesen einer XML-Datei (Forts.)

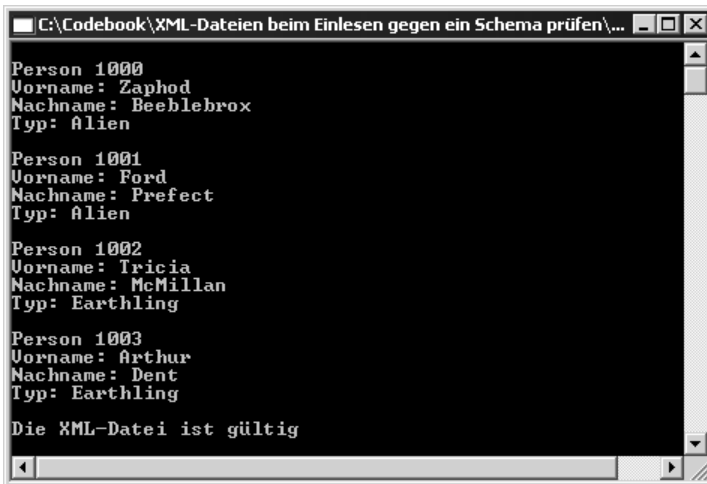


Abbildung 87: Einlesen einer gültigen XML-Datei

Abbildung 88 zeigt die Meldung des Programms für den Fall, dass das `id`-Attribut eines `person`-Elements einen nicht numerischen Wert speichert.

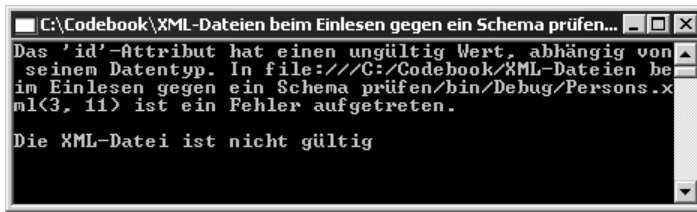


Abbildung 88: Fehlermeldung des Programms beim Einlesen einer `Person.xml`-Datei mit einem nicht numerischen `id`-Attribut in einem `person`-Element

ACHTUNG

Beachten Sie, dass Sie, wenn Sie validierend lesen, zum Lesen der Daten nie das `XmlTextReader`-Objekt verwenden sollten, sondern immer das `XmlValidatingReader`-Objekt. Lesen Sie versehentlich einmal mit dem einen und einmal mit dem anderen Objekt, erhalten Sie Fehlermeldungen, die der eigentlichen Ursache nicht zugeordnet werden können.

134 XML-Dokumente auf Gültigkeit überprüfen

Basierend auf der im vorhergehenden Rezept verwendeten `XmlValidatingReader`-Klasse, die XML-Dateien gegen ein XML-Schema, ein DTD oder einfach nur auf Wohlgeformtheit überprüfen kann, habe ich eine Klasse `XmlValidator` mit den Methoden `ValidateXmlFile` und `ValidateXmlDocument` entwickelt, die eine XML-Datei bzw. ein `XmlDocument`-Objekt (siehe Rezept 135) daraufhin überprüfen, ob diese gültiges XML speichern. Diese Methoden sind folgendermaßen deklariert:

```
public XmlValidatorResult ValidateXmlFile(string xmlFileName,
    ValidationType validationType, string schemaFileName)

public XmlValidatorResult ValidateXmlDocument(XmlDocument xmlDoc,
    ValidationType validationType, string schemaFileName)
```

Am ersten Argument erwarten die Methoden den Dateinamen bzw. ein `XmlDocument`-Objekt. Am zweiten Argument definieren Sie den Validiertyp mit Werten der Aufzählung `ValidationType`. Die folgenden Konstanten sind verfügbar:

- ▶ `None`: Es wird keine Überprüfung gegen ein Schema oder DTD, sondern lediglich auf Wohlgeformtheit vorgenommen,
- ▶ `Auto`: Die Art der Überprüfung (DTD, Schema oder XDR) wird automatisch an Informationen erkannt, die in das XML-Dokument eingebettet sind,
- ▶ `DTD`: Es wird gegen ein eingebettetes oder externes DTD geprüft,

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

- Schema: Es wird gegen ein eingebettetes oder externes Schema geprüft,
- XDR: Es wird gegen ein eingebettetes oder externes XDR-Schema geprüft.

Eine automatische Validierung (mit `ValidationType.Auto`) ist möglich, wenn das XML-Dokument ein internes DTD oder Schema enthält oder auf ein externes verweist. Ansonsten können Sie auch explizit gegen ein externes Schema (kein DTD!³) validieren, auf das im Dokument nicht verwiesen wird, indem Sie als Validiertyp `ValidationType.Schema` angeben und den Dateinamen der Schema-Datei im letzten Argument übergeben. Das ist sogar dann möglich, wenn das Dokument bereits ein DTD oder Schema enthält oder auf ein (anderes) externes verweist.

Am dritten Argument übergeben Sie dann den Namen einer externen Schemadatei, gegen die die Prüfung ausgeführt werden soll.

`ValidateXmlFile` und `ValidateXmlDocument` geben einen Wert der (eigenen) Aufzählung `XmlValidatorResult` zurück:

- `XmlValidatorResult.Valid`: Das Dokument ist in Ordnung,
- `XmlValidatorResult.ErrorsExists`: Das Dokument enthält Fehler,
- `XmlValidatorResult.WarningsExists`: Das Dokument ist prinzipiell in Ordnung, es wurden jedoch Warnungen generiert.

Falls Warnungen generiert wurden oder Fehler aufgetreten sind, können Sie diese aus den Eigenschaften `Errors` bzw. `Warnings` auslesen, die beide einfache `ArrayList`-Auflistungen sind. Damit diese Auflistungen zur Verfügung stehen, werden sie im Konstruktor der `XmlValidator`-Klasse erzeugt.

Die eigentliche Validierung übernimmt die private Methode `ValidateXml`, die ein `XmlTextReader`-Objekt und die Validierinformationen übergeben bekommt. Mit diesem Objekt erzeugt `ValidateXml` ein `XmlValidatingReader`-Objekt, definiert die Validierinformationen und geht dieses sequenziell durch um die Daten zu überprüfen. Dabei habe ich die Tatsache genutzt, dass die `Read`-Methode eines `XmlValidatingReader`-Objekts keine Ausnahme generiert, wenn der Eigenschaft `ValidationEventHandler` ein Delegate vom entsprechenden Typ zugewiesen wurde. Die mit dem Delegate verknüpfte Methode (in `XmlValidator` ist das die Methode `ValidationCallback`) wird in diesem Fall beim Auftreten eines Fehlers oder einer Warnung aufgerufen. Diese Methode hängt die Warnung bzw. den Fehler einfach an die `Errors`- bzw. `Warnings`-Auflistung der `XmlValidator`-Instanz an. Die `ValidateXml`-Methode fragt am Ende ab, ob in diesen Auflistungen Daten gespeichert sind, um zu entscheiden, welcher Wert zurückgegeben wird. Für den Fall, dass ein externes DTD oder Schema oder die XML-Datei nicht gelesen werden kann, erzeugt `ValidateXml` Ausnahmen vom Typ `XmlException`. Damit ist sichergestellt, dass die Validierung nur dann ausgeführt wird, wenn alle verbundenen Dokumente vorhanden bzw. bei den DTD- und Schema-Dokumenten auch gültig sind.

3. DTDs müssen über eine `<!DOCTYPE`-Deklaration in einem XML-Dokument angegeben werden, auch wenn es sich um ein externes DTD handelt.

Die öffentliche Methode `ValidateXmlFile` erzeugt nun einfach ein `XmlTextReader`-Objekt über den übergebenen Dateinamen und ruft damit `ValidateXml` auf.

Die ebenfalls öffentliche Methode `ValidateXmlDocument` ist ein wenig komplexer (und etwas »tricky«). Um aus dem übergebenen `XmlDocument`-Objekt ein `XmlTextReader`-Objekt zu erhalten, erzeugt `ValidateXmlDocument` zunächst mit Hilfe der `Save`-Methode des `XmlDocument`-Objekts einen `MemoryStream`. Dieser Stream wird dann als Basis für das `XmlTextReader`-Objekt verwendet. Dabei ist es wichtig, dass der Stream jeweils auf die Anfangsposition gesetzt wird, da die aktuelle Position nach dem Schreiben und nach dem Erzeugen des `XmlTextReader`-Objekts auf dem Ende des Stream steht und das `XmlTextReader`-Objekt in diesem Fall die Daten nicht lesen könnte (was zu einer Ausnahme mit der nicht gerade aussagefähigen Meldung »Systemfehler« führen würde). `ValidateXmlDocument` ruft deshalb jeweils die `Seek`-Methode des Stream auf um diesen an die Anfangsposition zu setzen. Mit dem erzeugten `XmlTextReader`-Objekt ruft `ValidateXmlDocument` dann nur noch `ValidateXml` auf.

Zum Kompilieren der Klasse `XmlValidator` müssen Sie die Namensräume `System`, `System.IO`, `System.Xml`, `System.Xml.Schema` und `System.Collections` importieren.

```
public class XmlValidator
{
    /* Eigenschaften zum Speichern von Fehlern und Warnungen */
    public ArrayList Errors;
    public ArrayList Warnings;

    /* Aufzählung für den Rückgabotyp */
    public enum XmlValidatorResult
    {
        Valid,           // Das Dokument ist gültig
        ErrorsExists,    // Es sind Fehler aufgetreten
        WarningsExists   // Es sind Warnungen gemeldet worden
    }

    /* Konstruktor */
    public XmlValidator()
    {
        this.Errors = new ArrayList();
        this.Warnings = new ArrayList();
    }

    /* Behandlungsmethode für das Validation-Ereignis */
    private void ValidationCallBack(object sender, ValidationEventArgs args)
    {
        // Fehler bzw. Warnung hinzufügen
        if (args.Severity == XmlSeverityType.Error)
            this.Errors.Add(args.Message);
        else
            this.Warnings.Add(args.Message);
    }
}
```

Listing 221: Klasse `XmlValidator` mit Methoden zum Validieren von XML-Dateien und -Dokumenten

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

/* Private Methode zum Validieren */
private XmlValidatorResult ValidateXml(XmlTextReader reader,
    ValidationType validationType, string schemaFileName)
{
    // Eventuelle alte Fehler löschen
    this.Errors.Clear();

    // XmlValidatingReader mit dem übergebenen XmlReader erzeugen
    XmlValidatingReader xmlValidator = null;
    try
    {
        xmlValidator = new XmlValidatingReader(reader);
    }
    catch (XmlException ex)
    {
        throw new XmlException("Fehler beim Einlesen der XML-Datei: " +
            ex.Message, ex);
    }

    // Validier-Typ festlegen
    xmlValidator.ValidationType = validationType;

    // Delegate mit der Methode für das Validation-Ereignis übergeben
    xmlValidator.ValidationEventHandler +=
        new ValidationEventHandler(this.ValidationCallBack);

    // Das externe Schema hinzufügen, wenn ein Dateiname angegeben wurde
    if (schemaFileName != null)
    {
        try
        {
            xmlValidator.Schemas.Add(null, schemaFileName);
        }
        catch (XmlException ex)
        {
            throw new XmlException("Fehler beim Hinzufügen des externen " +
                "Schemas: " + ex.Message, ex);
        }
    }

    // Daten einlesen und validieren
    try
    {
        while (xmlValidator.Read())
        {
        }
    }
    catch (Exception ex)
    {
        this.Errors.Add(ex.Message);
    }
}

```

Listing 221: Klasse *XmlValidator* mit Methoden zum Validieren von XML-Dateien und -Dokumenten (Forts.)

```

        // Ergebnis zurückgeben
        if (this.Errors.Count == 0 && this.Warnings.Count == 0)
            return XmlValidatorResult.Valid;
        else if (this.Errors.Count == 0)
            return XmlValidatorResult.WarningsExists;
        else
            return XmlValidatorResult.ErrorsExists;
    }

    /* Methode zum Validieren einer XML-Datei */
    public XmlValidatorResult ValidateXmlFile(string xmlFileName,
        ValidationType validationType, string schemaFileName)
    {
        // XmlTextReader erzeugen und damit ValidateXml aufrufen
        return ValidateXml(new XmlTextReader(xmlFileName), validationType,
            schemaFileName);
    }

    /* Methode zum Validieren eines XML-Dokuments */
    public XmlValidatorResult ValidateXmlDocument(XmlDocument xmlDoc,
        ValidationType validationType, string schemaFileName)
    {
        // MemoryStream aus dem Xml-Dokument erzeugen
        MemoryStream xmlStream = new MemoryStream();
        xmlDoc.Save(xmlStream);
        xmlStream.Seek(0, SeekOrigin.Begin);

        // Mit dem MemoryStream eine XmlTextReader-Instanz erzeugen
        XmlTextReader xmlTextReader = new XmlTextReader(xmlStream);
        xmlStream.Seek(0, SeekOrigin.Begin);

        return ValidateXml(xmlTextReader, validationType, schemaFileName);
    }
}

```

Listing 221: Klasse *XmlValidator* mit Methoden zum Validieren von XML-Dateien und -Dokumenten (Forts.)

Das folgende Listing zeigt eine beispielhafte Anwendung der Klasse *XmlValidator*, bei der die XML-Datei *Person.xml* gegen ein externes Schema *Person.xsd* geprüft wird. Das Programm benötigt die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume *System*, *System.IO*, *System.Xml*, *System.Collections* und *System.Windows.Forms*.

```

XmlValidator xmlValidator = new XmlValidator();

string xmlFileName = @"c:\Persons.xml";
string xsdFileName = @"c:\Persons.xsd";

```

Listing 222: Prüfen einer XML-Datei gegen ein externes Schema über die Klasse *XmlValidator*

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

switch (xmlValidator.ValidateXml(xmlFileName, ValidationType.Schema, xsdFileName))
{
    case XmlValidator.XmlValidatorResult.Valid:
        Console.WriteLine("Die XML-Datei ist valide.");
        break;

    case XmlValidator.XmlValidatorResult.WarningsExists:
        Console.WriteLine("Die XML-Datei ist valide, es wurden aber " +
            " Warnungen gemeldet.");

        // Warnungen ausgeben
        for (int i = 0; i < xmlValidator.Warnings.Count; i++)
        {
            Console.WriteLine(xmlValidator.Warnings[i].ToString());
        }
        break;

    case XmlValidator.XmlValidatorResult.ErrorsExists:
        Console.WriteLine("Die XML-Datei ist nicht valide.");

        // Fehler ausgeben
        Console.WriteLine("Die folgenden Fehler wurden gefunden:");
        for (int i = 0; i < xmlValidator.Errors.Count; i++)
        {
            Console.WriteLine();
            Console.WriteLine(xmlValidator.Errors[i].ToString());
        }

        // Überprüfen, ob Warnungen vorhanden sind, und diese ebenfalls ausgeben
        if (xmlValidator.Warnings.Count > 0)
        {
            Console.WriteLine("Die folgenden Warnungen wurden gemeldet:");
            for (int i = 0; i < xmlValidator.Warnings.Count; i++)
            {
                Console.WriteLine();
                Console.WriteLine(xmlValidator.Warnings[i].ToString());
            }
        }
        break;
}

```

Listing 222: Prüfen einer XML-Datei gegen ein externes Schema über die Klasse XmlValidator (Forts.)

Wird die XML-Datei in Listing 223 (die einige Fehler enthält) gegen das Schema in Listing 219 geprüft, resultiert die Ausgabe des Programms in Abbildung 89.

```

<?xml version="1.0" encoding="utf-8"?>
<persons>
    <person id="1000">
        <firstname>Zaphod</firstname>

```

Listing 223: Eine ungültige XML-Datei

```
<lastname>Beeblebrox</lastname>
<!-- Type fehlt -->
</person>
<person id="abc"> <!-- id ist nicht numerisch -->
  <!-- firstname fehlt -->
  <lastname>Prefect</lastname>
  <type>Alien</type>
</person>
<person id="1002">
  <firstname>Tricia</firstname>
  <!-- lastname fehlt -->
  <type>Earthling</type>
</person>
<person id="1003">
  <firstname>Arthur</firstname>
  <lastname>Dent</lastname>
  <type>Earthling</type>
</person>
</persons>
```

Listing 223: Eine ungültige XML-Datei (Forts.)



Abbildung 89: Das Beispielprogramm meldet Fehler in der XML-Datei

135 XML-Dateien über das DOM lesen

Wenn Sie beim Einlesen von XML-Dateien gezielt auf einzelne Elemente zugreifen und dabei vermeiden wollen, das gesamte Dokument sequenziell durchgehen zu müssen, können Sie ein `XmlDocument`-Objekt zum Lesen verwenden. Ein solches Objekt erlaubt über verschiedene Methoden den wahlfreien lesenden und schreibenden Zugriff auf ein XML-Dokument.

Die `XmlDocument`-Klasse implementiert das *DOM* (Document Object Model) des W3C. Das *World Wide Web Consortium* (www.w3c.org) entwickelt und empfiehlt Technologien und Standards für das Internet, wie z.B. HTML, Cascading Style Sheets, XML und eben das DOM.

Das DOM legt einen Satz an Klassen und deren Methoden und Eigenschaften fest, die ein XML-Dokument im Arbeitsspeicher repräsentieren und über die dieses Dokument gelesen und bearbeitet werden kann. Über diese Klassen können Sie ein XML-Dokument sehr flexibel bearbeiten. Das Lesen und Schreiben verbraucht aber im Vergleich zum SAX-ähnlichen Lesen mit einem `XmlTextReader`-Objekt (das ich im Rezept 132 beschrieben habe) und zum Schreiben mit einer `XmlTextWriter`-Instanz wesentlich mehr Arbeitsspeicher, da das XML-Dokument komplett in den Speicher gelesen und dort verarbeitet wird. Das sequenzielle Lesen eines XML-Dokuments ist über das DOM auch langsamer als mit den SAX-ähnlichen Klassen. Dafür können Sie gezielt auf XML-Elemente zugreifen, diese lesen, aktualisieren, löschen und neue Elemente hinzufügen.

Bevor ich in den Rezepten 137 und 140 auf das gezielte Lesen, Erstellen und Verändern eines XML-Dokuments über das DOM eingehe, zeige ich zunächst, wie Sie XML-Daten quasi-sequenziell einlesen.

Die zum Lesen eines kompletten XML-Dokuments oder eines Teilbereichs daraus wichtigste Methode ist `GetElementsByTagName`. Dieser Methode übergeben Sie den Namen eines XML-Tags und erhalten eine `XmlNodeList`-Auflistung mit Referenzen auf `XmlNode`-Objekte zurück, die die XML-Elemente mit dem angegebenen Namen repräsentieren. Für unser XML-Beispiel aus dem Rezept 132 rufe ich in Listing 224 `GetElementsByTagName` für den `person`-Tag auf um alle Elemente zu erhalten, die Personendaten speichern.

Die `XmlNode`-Klasse, deren Instanzen je einen XML-Knoten repräsentieren, ist die wichtigste Klasse des DOM. Zahlreiche Eigenschaften und Methoden erlauben das flexible Arbeiten mit einem Knoten. Die Eigenschaft `ChildNodes` liefert z.B. Referenzen auf alle untergeordneten Knoten in Form einer `XmlNodeList`-Auflistung, die Eigenschaft `FirstChild` repräsentiert den ersten Unterknoten (falls ein solcher existiert). Die `XmlDocument`-Klasse ist selbst von der `XmlNode`-Klasse abgeleitet, sodass ein XML-Dokument auch gleichzeitig ein XML-Knoten ist.

Über die `Attributes`-Eigenschaft eines `XmlNode`-Objekts erhalten Sie Zugriff auf die Attribute eines Knotens. Als Index können Sie neben einem `int`-Wert auch den Namen des Attributs übergeben. Die `SelectSingleNode`-Methode ermöglicht die gezielte Auswahl eines Unterknotens. Dazu übergeben Sie einen XPath-Ausdruck (XPath wird zum Navigieren in einem XML-Dokument verwendet). Im einfachsten Fall entspricht dieser Ausdruck dem Namen des Unterknotens. Im Erfolgsfall liefert diese Methode ein `XmlNode`-Objekt zurück, das den Unterknoten repräsentiert. Wird kein Unterknoten gefunden, erhalten Sie `null` zurück. Für unser Beispiel rufe ich diese Methode unter Übergabe der Tagnamen für die Unterknoten des `person`-Elements auf, um die Eigenschaften einer Person gezielt einzulesen. Den in einem Knoten gespeicherten Text lese ich dann über die Eigenschaft `InnerText` ein.

Das Beispiel aus dem Rezept 132, das die Personen-XML-Datei in eine `ArrayList`-Auflistung von `Person`-Objekten einliest, sieht mit dem DOM programmiert so aus wie in Listing 224.

Das Programm benötigt die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import der Namensräume `System`, `System.IO`, `System.Xml`, `System.Collections` und `System.Windows.Forms`.

```
/* Klasse zur Speicherung der Daten einer Person */
class Person
{
    public string Id;
    public string FirstName;
    public string LastName;
    public string Type;
}

/* Start-Klasse */
class Start
{
    [STAThread]
    static void Main(string[] args)
    {
        // Der Dateiname
        string xmlFileName = Path.Combine(Application.StartupPath,
            "Persons.xml");

        // XmlDocument-Instanz erzeugen
        XmlDocument xmlDoc = new XmlDocument();

        // Auflistung für die Personen erzeugen
        ArrayList persons = new ArrayList();

        // XML-Datei laden
        try
        {
            xmlDoc.Load(xmlFileName);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Fehler beim Einlesen der XML-Datei '" +
                xmlFileName + "': " + ex.Message, Application.ProductName,
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        // Alle person-Elemente einlesen und durchgehen
        XmlNodeList personNodeList = xmlDoc.GetElementsByTagName("person");
        for (int i = 0; i < personNodeList.Count; i++)
        {
            // Neue Person erzeugen und in der Auflistung ablegen
            Person person = new Person();
            persons.Add(person);

            // Das Attribut id einlesen
            XmlAttribute xmlAttribute = personNodeList[i].Attributes["id"];
            if (xmlAttribute != null)
                person.Id = xmlAttribute.InnerText;
        }
    }
}
```

Listing 224: Quasi-sequenzielles Einlesen einer XML-Datei über das DOM

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    // firstname-Element suchen und speichern
    XmlNode xmlNode = personNodeList[i].SelectSingleNode("firstname");
    if (xmlNode != null)
        person.FirstName = xmlNode.InnerText;

    // lastname-Element suchen und speichern
    xmlNode = personNodeList[i].SelectSingleNode("lastname");
    if (xmlNode != null)
        person.LastName = xmlNode.InnerText;

    // type-Element suchen und speichern
    xmlNode = personNodeList[i].SelectSingleNode("type");
    if (xmlNode != null)
        person.Type = xmlNode.InnerText;
}

// Alle eingelesenen Personen durchgehen und an der Konsole ausgeben
foreach (Person p in persons)
{
    Console.WriteLine("Person {0}", p.Id);
    Console.WriteLine("Vorname: {0}", p.FirstName);
    Console.WriteLine("Nachname: {0}", p.LastName);
    Console.WriteLine("Typ: {0}", p.Type);
    Console.WriteLine();
}
}
}

```

Listing 224: Quasi-sequenzielles Einlesen einer XML-Datei über das DOM (Forts.)



Abbildung 90: Die eingelesene XML-Datei

Sie sollten beachten, dass das in diesem Rezept vorgestellte Einlesen über ein `XmlDocument`-Objekt die XML-Daten nicht gegen ein eingebettetes oder externes DTD oder Schema überprüft. Um dies zu erreichen, lesen Sie das Rezept 136.

136 XML-Dateien validierend über das DOM lesen

Im vorhergehenden Rezept wurden die XML-Daten über ein `XmlDocument`-Objekt so eingelesen, dass keine Validierung gegenüber einem eventuell vorhandenen DTD oder Schema erfolgt. Um dies zu erreichen, müssen Sie der `Load`-Methode des `XmlDocument`-Objekts ein `XmlValidatingReader`-Objekt übergeben, das Sie zuvor entsprechend der gewünschten Validierung initialisieren (vergleichen Sie hierzu das Rezept 134). `Load` erzeugt eine Ausnahme vom Typ `XmlException`, wenn die eingelesenen Daten nicht dem Schema bzw. DTD entsprechen.

Das folgende Listing liest die Datei *Persons.xml* (Listing 217 in Rezept 132) über ein `XmlDocument`-Objekt ein. Über eine `XmlValidatingReader`-Instanz wird das Dokument dabei gegen das externe Schema *Persons.xsd* (Listing 219 in Rezept 133) validiert. Aus Vereinfachungsgründen werden die eingelesenen Daten lediglich an der Konsole ausgegeben.

Zum Kompilieren des Programms müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Xml` und `System.Windows.Forms` importieren.

```
// Der Dateiname der XML-Datei und der Schema-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");
string xsdFileName = Path.Combine(Application.StartupPath, "Persons.xsd");

// XmlDocument-Instanz erzeugen
XmlDocument xmlDoc = new XmlDocument();

// XML-Datei laden
try
{
    // XmlValidatingReader zum validierenden Lesen erzeugen und initialisieren
    XmlValidatingReader xmlValidatingReader = new XmlValidatingReader(
        new XmlTextReader(xmlFileName));
    xmlValidatingReader.ValidationType = ValidationType.Schema;
    xmlValidatingReader.Schemas.Add(null, xsdFileName);

    // XmlValidatingReader zum Laden des Dokuments verwenden
    xmlDoc.Load(xmlValidatingReader);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Einlesen der XML-Datei: " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
```

Listing 225: Validierendes Einlesen einer XML-Datei über das DOM

```

        MessageBoxIcon.Error);
    return;
}

// Alle person-Elemente einlesen und durchgehen
XmlNodeList personNodeList = xmlDoc.GetElementsByTagName("person");
for (int i = 0; i < personNodeList.Count; i++)
{
    Console.WriteLine("\r\nPerson");

    // Das Attribut id einlesen
    XmlAttribute xmlAttribute = personNodeList[i].Attributes["id"];
    if (xmlAttribute != null)
        Console.WriteLine("Id: {0}", xmlAttribute.InnerText);

    // firstname-Element suchen
    XmlNode xmlNode = personNodeList[i].SelectSingleNode("firstname");
    if (xmlNode != null)
        Console.WriteLine("Vorname: {0}", xmlNode.InnerText);

    // lastname-Element suchen
    xmlNode = personNodeList[i].SelectSingleNode("lastname");
    if (xmlNode != null)
        Console.WriteLine("Nachname: {0} ", xmlNode.InnerText);

    // type-Element suchen
    xmlNode = personNodeList[i].SelectSingleNode("type");
    if (xmlNode != null)
        Console.WriteLine("Typ: {0}", xmlNode.InnerText);
}

```

Listing 225: Validierendes Einlesen einer XML-Datei über das DOM (Forts.)

Hinweis

In diesem Beispiel wird das Programm abgebrochen, sobald nur ein Fehler auftritt. Sie können jedoch auch über die Zuweisung einer Ereignisbehandlungsmethode an die Eigenschaft `ValidationEventHandler` des `XmlValidatingReader`-Objekts erreichen, dass keine Ausnahme generiert, sondern für jeden Fehler diese Methode aufgerufen wird. Damit können Sie (mit ein wenig Programmierung) auch alle Fehler auswerten. Vergleichen Sie dazu das Rezept 134.

Abbildung 91 zeigt die Meldung des Programms für den Fall, dass in einem `person`-Element im Attribut `id` ein nicht numerischer Wert gespeichert ist. In der Schema-Datei (siehe Rezept 133) ist dieses Attribut als `int`-Wert definiert.



Abbildung 91: Fehlermeldung des Beispielprogramms beim Einlesen einer ungültigen XML-Datei

137 Gezielt über XPath auf Elemente in einem XML-Dokument zugreifen

Das DOM (siehe Rezept 135) ermöglicht in Form der `XmlDocument`-Klasse über die Methoden `SelectNodes` (mehrere Knoten lesen) und `SelectSingleNode` (einen Knoten lesen) über einen XPath-Ausdruck, den Sie diesen Methoden übergeben, gezielt auf einzelne Elemente eines XML-Dokuments zuzugreifen. XPath ist vom W3C als Sprache zur Adressierung von Teilen eines XML-Dokuments vorgesehen. XPath ist recht komplex und erlaubt eine sehr flexible Adressierung in einem XML-Dokument. Die Empfehlung des W3C finden Sie an der Adresse www.w3.org/TR/xpath. Eine sehr gute Einführung zu XPath finden Sie an der Adresse www.w3schools.com/xpath/default.asp.

Suchen in einem XML-Dokument ohne Namensraum-Zuordnung

Um auf ein Element in einem XML-Dokument ohne Zuordnung der Elemente zu einem Namensraum zuzugreifen, können Sie einen einfachen XPath-Ausdruck verwenden. Ein solcher setzt sich zusammen aus hierarchisch geordneten Element-Namen, die jeweils durch einen Schrägstrich voneinander getrennt werden. Ein Schrägstrich am Anfang bezeichnet das Wurzelement. Um auf alle `person`-Elemente der XML-Datei aus dem Rezept 132 zuzugreifen, würde der XPath-Ausdruck folgendermaßen aussehen:

```
/persons/person
```

Zum Einlesen aller `firstname`-Knoten würden Sie den folgenden XPath-Ausdruck verwenden:

```
/persons/person/firstname
```

Das Einlesen der Vornamen aller im Dokument gespeicherten Personen würde also z.B. so aussehen:

```
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(xmlFileName);

XmlNodeList xmlNodeList = xmlDoc.SelectNodes("/persons/person/firstname");
```

Listing 226: Suchen aller XML-Unterknoten mit einem bestimmten Namen

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

if (xmlNodeList != null)
{
    for (int i = 0; i < xmlNodeList.Count; i++)
        Console.WriteLine(xmlNodeList[i].InnerText);
}

```

Listing 226: Suchen aller XML-Unterknoten mit einem bestimmten Namen (Forts.)

Das Programm erfordert den Import der Namensräume `System` und `System.Xml`.

Wenn Sie gezielt auf ein Element mit einem bestimmten Attributwert zugreifen wollen, erweitern Sie den XPath-Ausdruck um einen *Knotentest* mit der Angabe des Attributnamens und des Werts des Attributs in der folgenden Form:

```
[attribute::Name='Wert']
```

Zum Einlesen der Person mit der Id 1001 verwenden Sie dann z.B. den folgenden Programmcode:

```

XmlNode xmlNode =
    xmlDoc.SelectSingleNode("/persons/person[attribute::id='1001']");
if (xmlNode != null)
{
    // Knoten für die Person mit der Id 1001 gefunden
}

```

Listing 227: Suchen eines XML-Knotens über einen Attributwert

Suchen Sie nach Knoten, deren Unterknoten einen bestimmten Wert speichern, erweitern Sie den XPath-Ausdruck um einen Knotentest in der folgenden Form:

```
[Name='Wert']
```

Wenn Sie z.B. alle Personen einlesen wollen, deren Typ *Alien* ist, können Sie den folgenden Programmcode verwenden:

```

xmlNodeList = xmlDoc.SelectNodes("/persons/person[type='Alien']");
if (xmlNodeList != null)
{
    // Passende person-Knoten wurden gefunden: Alle gefundenen Personen durchgehen
    for (int i = 0; i < xmlNodeList.Count; i++)
    {
        // Daten der Person einlesen und verarbeiten
    }
}

```

Listing 228: Suchen mehrerer XML-Knoten über den Wert des Inhalts eines Unterknotens

Das Programm in Listing 229 setzt dieses Wissen um. Es ermittelt zunächst alle in der XML-Datei gespeicherten Vornamen, liest dann die Daten der Person mit der Id 1001 ein und geht schließlich alle Personen durch, deren Typ *Alien* ist.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume *System*, *System.Xml*, *System.IO*, *System.Collections* und *System.Windows.Forms* importieren.

```
// Der Dateiname
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// XmlDocument-Instanz erzeugen
XmlDocument xmlDoc = new XmlDocument();

// XML-Datei laden
try
{
    xmlDoc.Load(xmlFileName);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Einlesen der XML-Datei '" + xmlFileName + "': " +
        ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}

// Einlesen aller Vornamen
XmlNodeList xmlNodeList = xmlDoc.SelectNodes("/persons/person/firstname");
if (xmlNodeList != null)
{
    for (int i = 0; i < xmlNodeList.Count; i++)
        Console.WriteLine(xmlNodeList[i].InnerText);
}

// Die Person mit der Id 1001 suchen
XmlNode xmlNode = xmlDoc.SelectSingleNode("/persons/person[attribute::id='1001']");
if (xmlNode != null)
{
    // Der passende person-Knoten wurde gefunden
    // Das Attribut id einlesen
    XmlAttribute xmlAttribute = xmlNode.Attributes["id"];
    string id = null;
    if (xmlAttribute != null)
        id = xmlAttribute.InnerText;

    // firstname-Element suchen
    XmlNode xmlSubNode = xmlNode.SelectSingleNode("firstname");
    string firstName = null;
    if (xmlSubNode != null)
        firstName = xmlSubNode.InnerText;
```

Listing 229: Gezieltes Zugreifen auf Teile eines XML-Dokuments über XPath

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    // lastname-Element suchen
    xmlSubNode = xmlNode.SelectSingleNode("lastname");
    string lastName = null;
    if (xmlSubNode != null)
        lastName = xmlSubNode.InnerText;

    // type-Element suchen
    string type = null;
    xmlSubNode = xmlNode.SelectSingleNode("type");
    if (xmlSubNode != null)
        type = xmlSubNode.InnerText;

    // Daten der Person ausgeben
    Console.WriteLine();
    Console.WriteLine("Person {0}", id);
    Console.WriteLine("Vorname: {0}", firstName);
    Console.WriteLine("Nachname: {0}", lastName);
    Console.WriteLine("Typ: {0}", type);
}
else
{
    MessageBox.Show("Person mit der Id 1001 nicht gefunden.",
        Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Exclamation);
}

// Personen mit Typ = Alien suchen
xmlNodeList = xmlDoc.SelectNodes("/persons/person[type='Alien']");
if (xmlNodeList != null)
{
    // Passende person-Knoten wurden gefunden: Alle gefundenen Personen
    // durchgehen
    for (int i = 0; i < xmlNodeList.Count; i++)
    {
        // Das Attribut id einlesen
        XmlAttribute xmlAttribute = xmlNodeList[i].Attributes["id"];
        string id = null;
        if (xmlAttribute != null)
            id = xmlAttribute.InnerText;

        // firstname-Element suchen
        XmlNode xmlSubNode =
            xmlNodeList[i].SelectSingleNode("firstname");
        string firstName = null;
        if (xmlSubNode != null)
            firstName = xmlSubNode.InnerText;

        // lastname-Element suchen
        xmlSubNode = xmlNodeList[i].SelectSingleNode("lastname");
        string lastName = null;
        if (xmlSubNode != null)

```

Listing 229: Gezieltes Zugreifen auf Teile eines XML-Dokuments über XPath (Forts.)

```

        lastName = xmlSubNode.InnerText;

        // type-Element suchen
        string type = null;
        xmlSubNode = xmlNodeList[i].SelectSingleNode("type");
        if (xmlSubNode != null)
            type = xmlSubNode.InnerText;

        // Daten der Person ausgeben
        Console.WriteLine();
        Console.WriteLine("Person {0}", id);
        Console.WriteLine("Vorname: {0}", firstName);
        Console.WriteLine("Nachname: {0}", lastName);
        Console.WriteLine("Typ: {0}", type);
    }
}
else
{
    MessageBox.Show("Keine Aliens gefunden.", Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}

```

Listing 229: Gezieltes Zugreifen auf Teile eines XML-Dokuments über XPath (Forts.)

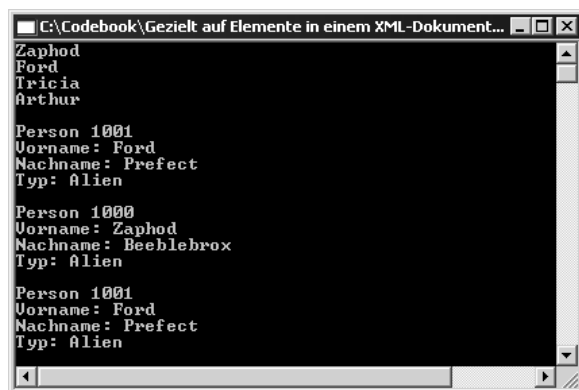


Abbildung 92: Das Beispielprogramm in Aktion

Hinweis

Zum Suchen von Daten in einem XML-Dokument könnten Sie auch noch neben der hier beschriebenen Variante die Methode `GetElementById` verwenden, der Sie die *Id* eines Elements übergeben. Problematisch daran ist nur, dass innerhalb des Schemas oder des DTDs des XML-Dokuments festgelegt sein muss, welche Attribute überhaupt den Typ *Id* besitzen. Dazu reicht es nicht aus, dass Sie ein Attribut einfach nur *id* nennen. Weiterhin ist es problematisch, wenn mehrere Knoten (unterschiedlicher Ebenen) denselben Id-Wert speichern. `GetElementById` gibt dann den ersten Knoten mit der angegebenen Id zurück. Für die Praxis ist die Suche über XPath also wesentlich einfacher und flexibler.

Suchen in XML-Dokumenten mit Namensräumen

Etwas komplizierter wird die Suche, wenn die gesuchten Elemente innerhalb der XML-Datei einem XML-Namensraum zugeordnet sind. Voraussetzung für das Funktionieren der Suche ist zunächst, dass der Namensraum in der XML-Datei einem Präfix⁴ zugeordnet wird, wie es im folgenden Beispiel (in Form des Präfixes *awc* der Fall ist):

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<awc:persons xmlns:awc="http://www.addison-wesley.de/codebook">
...
</awc:persons>
```

Die Verwendung eines Präfixes ist deswegen notwendig, weil der XPath-Standard Default-Namensräume (das ist die Angabe eines XML-Namensraums ohne Präfix) nicht kennt, und die Suche folglich nicht zum Erfolg führt, wenn kein Präfix verwendet wird.

Um in einer XML-Datei mit Namensraum-Zuordnung zu suchen, müssen Sie nach dem Erzeugen und Initialisieren des `XmlDocument`-Objekts ein `XmlNamespaceManager`-Objekt erzeugen. Ein solches Objekt assoziiert Namensräume mit einem anzugebenden Präfix. Im Konstruktor übergeben Sie die `NameTable`-Eigenschaft des `XmlDocument`-Objekts. Diese Eigenschaft verweist auf ein `XmlNameTable`-Objekt, das intern alle Element- und Attributnamen des Dokuments verwaltet. Diese Informationen werden vom `XmlNamespaceManager`-Objekt benötigt.

Nun fügen Sie über die `AddNamespace`-Methode einen oder mehrere Namensräume hinzu und geben für jeden Namensraum am ersten Argument einen Präfix an (der nicht identisch sein muss mit dem im XML-Dokument verwendeten Präfix!).

Wenn Sie dann über `SelectSingleNode` oder `SelectNodes` suchen, übergeben Sie am zweiten Argument das `XmlNamespaceManager`-Objekt. Der XPath-Ausdruck muss dann nur noch um den im `XmlNamespaceManager` definierten Namensraum-Präfix erweitert werden.

Hinweis

Der Grund für die Notwendigkeit eines `XmlNamespaceManager`-Objekts ist, dass ein Namensraum-Präfix innerhalb eines XML-Dokuments nicht eindeutig sein muss. Ein Dokument kann also mehrere unterschiedliche Namensräume demselben Präfix zuordnen:

```
<root>
  <xyz:Element xmlns:xyz="http://www.microsoft.com"/>
  <xyz:Element xmlns:xyz="http://www.msn.com"/>
</root>
```

Eine theoretisch denkbare direkte Auswahl des Namensraums über dessen Präfix im XPath-Ausdruck ist also nicht unbedingt eindeutig und deswegen gar nicht möglich.

4. Ein Präfix symbolisiert einen Namensraum und kann vor den lokalen Namen von Elementen und Attributen gesetzt werden, um diese dem Namensraum zuzuordnen. Sinnvoll sind Präfixe, wenn ein XML-Dokument Elemente aus mehreren Namensräumen besitzt.

Das Beispiel in Listing 231 zeigt eine XPath-Abfrage in einem XML-Dokument mit Namensraum in Form der Suche nach der Person mit der Id 1001. Die Elemente der XML-Datei sind dabei dem Namensraum *http://www.addison-wesley.de/codebook* zugeordnet (Listing 230).

```
<?xml version="1.0" encoding="utf-8" ?>
<awc:persons xmlns:awc="http://www.addison-wesley.de/codebook">
  <person id="1000">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <type>Alien</type>
  </person>

  <person id="1001">
    <firstname>Ford</firstname>
    <lastname>Prefect</lastname>
    <type>Alien</type>
  </person>
</awc:persons>
```

Listing 230: Die (verkürzte) XML-Datei mit Namensraumzuordnung

Das Beispiel erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume *System*, *System.IO*, *System.Xml* und *System.Windows.Forms*.

```
// Der Dateiname
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// XmlDocument-Instanz erzeugen und XML-Datei laden
XmlDocument xmlDoc = new XmlDocument();
try
{
    xmlDoc.Load(xmlFileName);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Einlesen der XML-Datei '" + xmlFileName +
        "': " + ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}

// XmlNamespaceManager erzeugen und mit dem Namensraum initialisieren
XmlNamespaceManager nsManager = new XmlNamespaceManager(xmlDoc.NameTable);
nsManager.AddNamespace("awc", "http://www.addison-wesley.de/codebook");

// Die Person mit der Id 1001 suchen
XmlNode xmlNode = null;
```

Listing 231: XPath-Suche in einem XML-Dokument mit Namensraum-Zuordnung

```

try
{
    // XPath-Ausdruck, der das Element person über den im XmlNamespaceManager-
    // Objekt verwalteten Namensraum-Präfix referenziert
    string xpathQuery = "/awcodebook:persons/person[attribute::id='1001']";

    // XPath-Abfrage mit Übergabe des Namensraum-Managers absetzen
    xmlNode = xmlDoc.SelectSingleNode(xpathQuery, nsManager);
}
catch (Exception ex)
{
    Console.WriteLine("Fehler beim Suchen: " + ex.Message);
}

if (xmlNode != null)
{
    // Der passende person-Knoten wurde gefunden: Das Attribut id einlesen
    XmlAttribute xmlAttribute = xmlNode.Attributes["id"];
    if (xmlAttribute != null)
        Console.WriteLine("Person {0}", xmlAttribute.InnerText);

    // firstname-Element suchen
    XmlNode xmlSubNode = xmlNode.SelectSingleNode("firstname");
    if (xmlSubNode != null)
        Console.WriteLine("Nachname: {0}", xmlSubNode.InnerText);

    // lastname-Element suchen
    xmlSubNode = xmlNode.SelectSingleNode("lastname");
    if (xmlSubNode != null)
        Console.WriteLine("Vorname: {0}", xmlSubNode.InnerText);

    // type-Element suchen
    xmlSubNode = xmlNode.SelectSingleNode("type");
    if (xmlSubNode != null)
        Console.WriteLine("Typ: {0}", xmlSubNode.InnerText);
}
else
{
    MessageBox.Show("Person mit der Id 1001 nicht gefunden.",
        Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Exclamation);
}

```

Listing 231: XPath-Suche in einem XML-Dokument mit Namensraum-Zuordnung (Forts.)



Abbildung 93: Das Programm hat die Person mit der Id 1001 gefunden

138 XML-Dokumente über ein DataSet-Objekt lesen

XML-Dokumente, die eine tabellenähnliche Struktur aufweisen, können Sie auch in ein DataSet-Objekt (aus dem Namensraum `System.Data`) lesen und als `DataTable`-Objekt bearbeiten. Ein solches Vorgehen erleichtert in vielen Fällen die Arbeit mit den XML-Daten, da Sie diese mit den flexiblen Methoden eines `DataTable`-Objekts bearbeiten können. So können Sie die Daten z.B. problemlos gefiltert darstellen, darin suchen, verändern und wieder abspeichern. Natürlich hat ein DataSet auch Nachteile. Zum einen wird das gesamte XML-Dokument in den Arbeitsspeicher gelesen, was bei großen Dokumenten viel Speicher in Anspruch nimmt. Zum anderen werden bei einem solchen Lesen nur die XML-Elemente der ersten und zweiten Hierarchieebene berücksichtigt.

Zum Lesen einer XML-Datei erzeugen Sie ein neues DataSet-Objekt und lesen die Datei über die `ReadXml`-Methode ein. Das DataSet erkennt alle XML-Elemente direkt unterhalb des Wurzelements als Datensatz und fasst gleichnamige Elemente zu Tabellen zusammen. Die Attribute des jeweiligen Elements und die direkt untergeordneten Daten-Elemente werden in Form von Spalten im Datensatz dargestellt. Elemente, die in tieferen Ebenen untergeordnet sind, werden nicht berücksichtigt.

Beim Einlesen der XML-Datei in Listing 232 werden z.B. zwei `DataTable`-Objekte (mit den Namen »*person*« und »*vehicle*«) erzeugt:

```
<?xml version="1.0" encoding="utf-8"?>
<persons
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Persons.xsd">

  <person id="1000">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <vehicleId>1</vehicleId>
  </person>

  <person id="1001">
    <firstname>Tricia</firstname>
    <lastname>McMillan</lastname>
    <vehicleId>2</vehicleId>
```

Listing 232: Beispiel-XML-Datei für das Einlesen über ein DataSet-Objekt

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

</person>

<vehicle id="1">
  <type>Spaceship</type>
</vehicle>

<vehicle id="2">
  <type>Car</type>
</vehicle>
</persons>

```

Listing 232: Beispiel-XML-Datei für das Einlesen über ein DataSet-Objekt (Forts.)

Das Einlesen dieser Datei ist sehr einfach. Dazu erzeugen Sie eine Instanz der Klasse `DataSet` und rufen die `ReadXml`-Methode auf. Am ersten Argument übergeben Sie den Dateinamen der XML-Datei. Im optionalen zweiten Argument können Sie u.a. mit `XmlReadMode.ReadSchema` festlegen, dass ein in der Datei eingebettetes XML-Schema berücksichtigt wird. Ist das Schema in einer separaten Datei gespeichert, können Sie dieses vor dem Lesen der Daten über die Methode `ReadXmlSchema` einlesen. Ein Schema für die Personen-XML-Datei könnte z.B. so aussehen:

```

<?xml version="1.0" standalone="yes"?>
<xs:schema
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" id="persons">
  <xs:element name="persons">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="firstname" type="xs:string"/>
              <xs:element name="lastname" type="xs:string"/>
              <xs:element name="vehicleId" type="xs:int"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:int" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="vehicle">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="type" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:int" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Listing 233: Beispiel-Schema für die Personen-XML-Datei, das das einmalige Vorkommen der person- und der vehicle-Elemente absichert und den Datentyp dieser Elemente festlegt

```

    </xs:complexType>
  </xs:element>
</xs:schema>

```

Listing 233: Beispiel-Schema für die Personen-XML-Datei, das das einmalige Vorkommen der person- und der vehicle-Elemente absichert und den Datentyp dieser Elemente festlegt (Forts.)

Das Schema legt für die person-Elemente fest, dass diese ein Attribut `id` vom Typ `int` und je ein Unterelement `firstname` (`string`), `lastname` (`string`) und `vehicleId` (`int`) besitzen. Für vehicle-Elemente wird festgelegt, dass diese ebenfalls ein `int`-Attribut `id` besitzen und als Unterelement das Element `name` mit dem Typ `string` vorkommen muss.

Ohne Schema werden die Daten zunächst beim Einlesen nicht validiert und alle Daten als String eingelesen. Ein Schema kann u.a. den Datentyp der einzelnen Elemente festlegen, wie es im Beispiel-Schema der Fall ist. Die einzelnen Spalten im `DataSet` besitzen dann den im Schema festgelegten Datentyp.

Den Aufruf der `ReadXmlSchema`- und der `ReadXml`-Methode sollten Sie in eine Ausnahmebehandlung integrieren, da beim Lesen Fehler auftreten können.

Danach können Sie die erzeugten `DataTable`-Objekte bearbeiten wie normale `DataTable`-Objekte. Sie können die einzelnen Zeilen einlesen, neue Zeilen hinzufügen und vorhandene verändern oder löschen. Das Beispiel in Listing 234 liest die einzelnen für die person- und vehicle-Elemente erzeugten Datensätze ein und gibt deren Spaltenwerte an der Konsole aus.

Zum Kompilieren des Programms müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Data` und `System.Windows.Forms` importieren.

```

// Die Dateinamen
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");
string xmlSchemaFileName = Path.Combine(Application.StartupPath,
    "Persons.xsd");

// DataSet erzeugen
DataSet dataSet = new DataSet();

// XML-Schema und -Daten einlesen
try
{
    dataSet.ReadXmlSchema(xmlSchemaFileName);
    dataSet.ReadXml(xmlFileName);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Einlesen der XML-Datei '" + xmlFileName +
        "': " + ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}

```

Listing 234: Lesen einer XML-Datei mit Schema über ein DataSet

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
        return;
    }

    // Die Personen durchgehen
    DataTable dataTable = dataSet.Tables["person"];
    if (dataTable != null)
    {
        for (int i = 0; i < dataTable.Rows.Count; i++)
        {
            Console.WriteLine(dataTable.Rows[i]["id"]);
            Console.WriteLine(dataTable.Rows[i]["firstName"]);
            Console.WriteLine(dataTable.Rows[i]["lastName"]);
            Console.WriteLine(dataTable.Rows[i]["vehicleId"]);
            Console.WriteLine();
        }
    }
    else
    {
        Console.WriteLine("Keine Personen gespeichert");
    }

    // Die Fahrzeuge durchgehen
    dataTable = dataSet.Tables["vehicle"];
    if (dataTable != null)
    {
        for (int i = 0; i < dataTable.Rows.Count; i++)
        {
            Console.WriteLine(dataTable.Rows[i]["id"]);
            Console.WriteLine(dataTable.Rows[i]["type"]);
            Console.WriteLine();
        }
    }
    else
    {
        Console.WriteLine("Keine Fahrzeuge gespeichert");
    }
}
```

Listing 234: Lesen einer XML-Datei mit Schema über ein DataSet (Forts.)



Abbildung 94: Das Beispielprogramm in Aktion

Die Ausnahmen, die die `ReadXml`-Methode erzeugt, wenn das XML-Dokument nicht dem XML-Schema entspricht, sind nicht gerade aussagekräftig. Abbildung 95 zeigt die Meldung, die vom Programm generiert wird, wenn das Attribut `id` eines `person`-Elements einen nicht numerischen Wert speichert. Um diese nicht sehr aussagekräftige Fehlermeldung zu vermeiden und die einzelnen Fehler der XML-Datei genauer anzuzeigen, sollten Sie diese vor dem Lesen oder beim Auftreten einer Ausnahme validieren. Informationen dazu finden Sie im Rezept 134.



Abbildung 95: Fehlermeldung beim Lesen einer `Person.xml`-Datei, bei der das Attribut `id` eines `person`-Elements einen ungültigen Wert speichert

139 XML-Dateien mit einem `XmlTextWriter` erzeugen

Die Verwendung eines `XmlTextWriter`-Objekts ist die effizienteste Methode, XML-Dateien komplett neu zu erstellen. Über ein solches Objekt erzeugen Sie die XML-Datei sequenziell, im Gegensatz zum Schreiben über ein `XmlDocument`-Objekt, das ein beliebiges Einfügen von Knoten erlaubt.

Beim Erzeugen eines `XmlTextWriter`-Objekts können Sie im Konstruktor neben einem vorhandenen `XmlTextWriter`-Objekt ein `Stream`-Objekt, das die zu erzeugende Datei repräsentiert, oder den Namen der zu erzeugenden Datei übergeben. In den letzten beiden Fällen definieren Sie am zweiten Argument die Codierung der Datei in Form einer `Encoding`-Instanz (siehe Rezept 118, Textdateien lesen).

Vor dem Schreiben können Sie festlegen, wie das Dokument formatiert wird. Stellen Sie die Eigenschaft `Formatting` auf `Formatting.Indented`, wenn die XML-Datei mit Zeilenumbrüchen generiert werden soll. In der Eigenschaft `Indentation` können Sie dann die Anzahl der Zeichen für die Einrückung untergeordneter Elemente und in der Eigenschaft `IndentChar` das dazu verwendete Zeichen festlegen.

Dann schreiben Sie die einzelnen Bestandteile des Dokuments über eine Vielzahl an `Write`-Methoden (von denen ich hier nur die wichtigsten beschreibe).

`WriteStartDocument` erzeugt zunächst den Beginn des Dokuments. Über das optionale boolesche Argument können Sie festlegen, welchen Wert das `standalone`-Attribut besitzen soll.

Einzelne Elemente ohne Textdaten erzeugen Sie über `WriteStartElement`. Dabei können Sie (u.a.) den Namen des Elements und optional den Namensraum⁵ übergeben, dem das Element zugeordnet sein soll. Attribute eines Elements können Sie über `WriteAttributeString` schreiben.

Elemente mit Textdaten erzeugen Sie auf eine einfache Weise über die Methode `WriteElementString`. Diese Methode erzeugt das Startelement, den Textinhalt und direkt auch das End-Element. Alternativ können Sie auch zunächst `WriteStartElement` aufrufen, dann den Inhalt über `WriteString` oder über eine andere Methode wie `WriteBase64` (binäre Daten Base64-codiert schreiben) oder `WriteChars` (Byte-Array als String schreiben) erzeugen und das Element schließlich über `WriteEndElement` abschließen. `WriteEndElement` schließt immer das Element ab, das zuletzt über `WriteStartElement` begonnen wurde. Auf diese Weise müssen Sie alle »offenen« Elemente abschließen.

Am Ende der Datei rufen Sie dann noch `WriteEndDocument` auf um das Dokument abzuschließen.

Das Programm in Listing 236 erzeugt eine einfache XML-Datei mit den Daten zweiter Personen. Diese Datei sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<persons xmlns="http://www.addison-wesley.de/codebook/csharp">
  <!--Beispiel zum Erzeugen von XML-Dateien-->
  <person id="1001">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <type>Alien</type>
  </person>
  <person id="1002">
    <firstname>Tricia</firstname>
    <lastname>McMillan</lastname>
    <type>Earthling</type>
  </person>
</persons>
```

Listing 235: Die vom Beispiel erzeugte XML-Datei

Da ein `XmlTextWriter`-Objekt eine bereits vorhandene Datei ohne Warnung überschreibt, überprüft das Beispielprogramm vor dem Schreiben, ob bereits eine gleichnamige Datei existiert, und fragt den Anwender in diesem Fall, ob der die Datei überschreiben will. Ausnahmen, die vorzugsweise beim Öffnen entstehen, werden abgefangen.

5. XML-Namensräume, die üblicherweise (nicht unbedingt existent) Internet-Adressen entsprechen, sichern ab, dass Elemente (über den Namensraum) auch dann eindeutig identifiziert werden können, wenn mehrere XML-Dokumente zu einem Dokument zusammengeführt werden. Wenn Sie ein Element einem Namensraum zuordnen, sind automatisch auch alle untergeordneten Elemente diesem Namensraum zugeordnet.

Das Beispiel erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume *System*, *System.IO*, *System.Text*, *System.Xml* und *System.Windows.Forms*.

```
// Der Dateiname der XML-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "persons.xml");

// Überprüfen, ob die Datei existiert
if (File.Exists(xmlFileName))
{
    if (MessageBox.Show("Die Datei '" + xmlFileName +
        "' existiert bereits.\r\n\r\nWollen Sie diese " +
        "Datei überschreiben?", Application.ProductName,
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
        DialogResult.No)
        return;
}

// XmlTextWriter für eine UTF-8-Codierung erzeugen
XmlTextWriter xmlTextWriter = null;
try
{
    xmlTextWriter = new XmlTextWriter(xmlFileName, Encoding.UTF8);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Öffnen der XML-Datei: " + ex.Message,
        Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

// Formatierung so einstellen, dass einzelne um drei Leerzeichen eingerückte
// Zeilen erzeugt werden
xmlTextWriter.Formatting = Formatting.Indented;
xmlTextWriter.Indentation = 3;
xmlTextWriter.IndentChar = ' ';

// Dokument-Beginn mit standalone-Attribut schreiben
xmlTextWriter.WriteStartDocument(true);

// Startelement mit Namensraum schreiben
xmlTextWriter.WriteStartElement("persons",
    "http://www.addison-wesley.de/codebook/csharp");

// Kommentar schreiben
xmlTextWriter.WriteComment("Beispiel zum Erzeugen von XML-Dateien");

// Element für eine Personen erzeugen
xmlTextWriter.WriteStartElement("person");
```

Listing 236: Schreiben einer XML-Datei über ein XmlTextWriter-Objekt

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// Attribut id schreiben
xmlTextWriter.WriteAttributeString("id", "1001");

// Unterelemente mit String-Daten erzeugen
xmlTextWriter.WriteElementString("firstname", "Zaphod");
xmlTextWriter.WriteElementString("lastname", "Beeblebrox");
xmlTextWriter.WriteElementString("type", "Alien");

// person-Element abschließen
xmlTextWriter.WriteEndElement();

// Element für eine weitere Personen erzeugen
xmlTextWriter.WriteStartElement("person");

// Attribut id schreiben
xmlTextWriter.WriteAttributeString("id", "1002");

// Unterelemente mit String-Daten erzeugen
xmlTextWriter.WriteElementString("firstname", "Tricia");
xmlTextWriter.WriteElementString("lastname", "McMillan");
xmlTextWriter.WriteElementString("type", "Earthling");

// person-Element abschließen
xmlTextWriter.WriteEndElement();

// Dokument abschließen
xmlTextWriter.WriteEndElement();
xmlTextWriter.WriteEndDocument();

// XmlTextWriter schließen
xmlTextWriter.Close();
```

Listing 236: Schreiben einer XML-Datei über ein XmlTextWriter-Objekt (Forts.)

140 XML-Dateien über ein XmlDocument-Objekt erzeugen und ändern

Wie ich bereits im Rezept 135 beschrieben habe, implementiert die `XmlDocument`-Klasse das DOM (Document Object Model) des W3C und repräsentiert ein XML-Dokument im Arbeitsspeicher in Form von hierarchisch angeordneten Objekten.

Die `XmlNode`-Klasse, deren Instanzen je einen XML-Knoten repräsentieren, ist die wichtigste Klasse des DOM. Über zahlreiche Eigenschaften und Methoden können Sie sehr flexibel mit einem Knoten arbeiten. Das `XmlDocument` selbst ist von dieser Klasse abgeleitet.

Über die Methode `AppendChild` können Sie einem Knoten einen neuen Unterknoten hinzufügen. Über `InsertBefore` oder `InsertAfter` können Sie Knoten vor oder hinter anderen Knoten einfügen. Die Auflistungseigenschaft `ChildNodes` ermöglicht den Zugriff auf die einem Knoten untergeordneten Knoten. Über die Eigenschaft `InnerText` können Sie auf den Text eines Elements zugreifen. Die Auflistung `Attributes` ermöglicht den Zugriff auf die Attribute eines Knotens. Ein Knoten besitzt darüber hinaus noch eine Vielzahl weiterer Eigenschaften und

Methoden, die ich hier nicht weiter beschreibe. Einige wichtige Programmieretechniken zur Arbeit mit XML-Dokumenten zeige ich an Hand einer Klasse zur Arbeit mit Kontakten, die in einer XML-Datei gespeichert sind. Diese XML-Datei sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<cbch:contacts
  xmlns:cbch="uuid:7ACED865-0622-48f7-B138-45FED1BB4B27">
  <contact id="1000">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <phone>042-42</phone>
  </contact>
  <contact id="1002">
    <firstname>Tricia</firstname>
    <lastname>McMillan</lastname>
    <phone>01234-12347</phone>
  </contact>
</cbch:contacts>
```

Listing 237: Die XML-Datei zur Verwaltung von Kontakten

Diese Klasse zur Arbeit mit den Kontakten enthält zunächst eine Eigenschaft für das XML-Dokument und den Dateinamen. Da die Kontakte in einem XML-Namensraum verwaltet werden, enthält die Klasse zwei weitere, konstante Eigenschaften, die den XML-Namensraum und dessen Präfix verwalten.

Zum Kompilieren der Klasse müssen Sie die Namensräume `System`, `System.Text` und `System.Xml` importieren.

```
public class ContactHandler
{
    /* Eigenschaften für das Xml-Dokument */
    private XmlDocument xmlDoc;
    private string xmlFileName;

    /* Konstante Eigenschaften für den Namensraum und das
     * Namensraum-Präfix der Kontakt-Elemente */
    private const string xmlNamespace =
        "uuid:7ACED865-0622-48f7-B138-45FED1BB4B27";
    private string xmlNamespacePrefix = "cbch";
}
```

Listing 238: Eigenschaften der Klasse zur Arbeit mit Kontakten in einer XML-Datei

Erzeugen eines neuen oder Laden eines vorhandenen Dokuments im Konstruktor
Im Konstruktor wird der Dateiname übergeben und ein boolescher Wert, der bestimmt, ob die Datei neu erzeugt werden soll. Am dritten Argument kann ein Namensraum übergeben werden, dem die Elemente des Dokuments zugeordnet werden sollen. Das vierte Argument definiert die Codierung einer neu zu erzeugenden Datei.

```
public ContactHandler(string xmlFileName, bool createNew, Encoding encoding)
```

Der Konstruktor legt zunächst den übergebenen Dateinamen für das spätere Speichern des Dokuments in der privaten Eigenschaft `xmlFileName` ab und erzeugt dann ein neues `XmlDocument`-Objekt. Soll die XML-Datei neu geschrieben werden, erzeugt der Konstruktor über die `CreateElement`-Methode des `XmlDocument`-Objekts ein neues XML-Element, das als Wurzel-Element dienen soll. Diesem Element wird neben dem Namen auch der Namensraum und dessen Präfix übergeben, damit alle untergeordneten Elemente diesem zugeordnet werden. Dieses zu diesem Zeitpunkt noch nicht zugeordnete Element wird dann über die `AppendChild`-Methode dem Dokument hinzugefügt. `AppendChild` hängt einen Unterknoten hinten an den Knoten an, für den diese Methode aufgerufen wurde.

Da ein XML-Dokument immer eine XML-Deklaration enthalten muss, wird diese über die `CreateXmlDeclaration`-Methode erzeugt. Dabei muss zurzeit noch am ersten Argument die Version 1.0 übergeben werden, da XML lediglich in dieser Version verfügbar ist. Das zweite Argument bezeichnet die Codierung der Datei. Hier übergibt der Konstruktor einfach den Wert der Eigenschaft `WebName` des übergebenen `Encoding`-Objekts (oder als Voreinstellung »utf-8« wenn kein solches übergeben wurde). Das dritte Argument steht für das `standalone`-Attribut. Das damit erzeugte `XmlDeclaration`-Objekt fügt der Konstruktor über `InsertBefore` dann vor dem Wurzelknoten an.

Für den Fall, dass die XML-Datei nicht neu erzeugt, sondern eingelesen werden soll, ruft der Konstruktor lediglich die `Load`-Methode auf.

```
public ContactHandler(string xmlFileName, bool createNew,
    Encoding encoding)
{
    // Dateiname für das Speichern ablegen
    this.xmlFileName = xmlFileName;

    // XML-Dokument erzeugen bzw. einlesen
    this.xmlDoc = new XmlDocument();
    if (createNew)
    {
        // Wurzelknoten mit Namensraumdeklaration erzeugen und anfügen
        XmlNode rootNode = this.xmlDoc.CreateElement(xmlNamespacePrefix,
            "contacts", xmlNamespace);
        this.xmlDoc.AppendChild(rootNode);

        // XML-Deklaration erzeugen und vor dem Wurzelknoten anfügen
        string encodingName;
        if (encoding != null)
            encodingName = encoding.WebName;
        else
            encodingName = "utf-8";
        XmlDeclaration xmlDeclaration = this.xmlDoc.CreateXmlDeclaration(
            "1.0", encodingName, "yes");
```

Listing 239: Der Konstruktor der Klasse zur Arbeit mit Kontakten in einer XML-Datei

```

        this.xmlDoc.InsertBefore(xmlDeclaration, rootNode);
    }
    else
    {
        // XML-Dokument einlesen
        xmlDoc.Load(xmlFileName);
    }
}

```

Listing 239: Der Konstruktor der Klasse zur Arbeit mit Kontakten in einer XML-Datei (Forts.)

Suchen von Elementen

Da in verschiedenen Methoden der Klasse `contact`-Elemente nach der Id gesucht werden müssen, übernimmt eine private Methode diese Suche über eine XPath-Abfrage. Diese Methode ist etwas komplex, da der Wurzelknoten der XML-Datei in einem Namensraum verwaltet wird (siehe Rezept 137). Die Methode gibt eine `XmlNodeList`-Auflistung mit den gefundenen Knoten zurück:

```

private XmlNodeList FindContact(int id)
{
    XmlNamespaceManager nsManager = new XmlNamespaceManager(
        this.xmlDoc.NameTable);
    nsManager.AddNamespace(xmlNamespacePrefix, xmlNamespace);
    string xpathQuery = "/" + xmlNamespacePrefix +
        ":contacts/contact[attribute::id='" + id + "']";
    return xmlDoc.SelectNodes(xpathQuery, nsManager);
}

```

Listing 240: Private Methode zum Suchen von contact-Elementen nach einer Id

Hinzufügen von Elementen

Die Methode `AddContact` übernimmt das Hinzufügen eines Kontaktes. Zur Vermeidung mehrfacher Einträge überprüft sie zunächst über die `FindContact`-Methode, ob bereits ein `contact`-Element mit der übergebenen Id existiert, und generiert im Fehlerfall eine Ausnahme.

Dann erzeugt `AddContact` über die `CreateElement`-Methode des `XmlDocument`-Objekts eine neue Instanz der `XmlNode`-Klasse, wobei in deren Konstruktor der Name des Elements übergeben wird. Danach wird über die Methode `CreateAttribute` ein neues Attribut erzeugt, das im Anschluss mit der übergebenen Id initialisiert und an die `Attributes`-Auflistung des Kontakt-Knotens angehängt wird.

Für die einzelnen Eigenschaften des Kontakts erzeugt `AddContact` dann jeweils ein neues Element, definiert dessen inneren Text (also den Inhalt des Elements) und hängt den so erzeugten Knoten an den Kontakt-Knoten über dessen `AppendChild`-Methode an. Der Kontakt-Knoten wird schließlich noch an das Dokument-Element (das Wurzel-Element) des `XmlDocument`-Objekts angehängt.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

public void AddContact(int id, string firstName, string lastName,
    string phone)
{
    // Zunächst überprüfen, ob bereits ein Element mit der übergebenen Id
    // existiert
    XmlNodeList contactNodes = this.FindContact(id);
    if (contactNodes.Count > 0)
    {
        // Ausnahme werfen
        throw new XmlException("Kontakt mit der Id '" + id +
            "' existiert bereits", null);
    }

    // Element-Knoten für den Kontakt erzeugen
    XmlNode contactNode = this.xmlDoc.CreateElement("contact");

    // Das Attribut id erzeugen und an den Knoten übergeben
    XmlAttribute idAttribute = this.xmlDoc.CreateAttribute("id");
    idAttribute.Value = id.ToString();
    contactNode.Attributes.Append(idAttribute);

    // Unterknoten für die Kontakteigenschaften erzeugen und dem
    // Kontaktknoten anfügen
    XmlNode firstNameNode = this.xmlDoc.CreateElement("firstname");
    firstNameNode.InnerText = firstName;
    contactNode.AppendChild(firstNameNode);

    XmlNode lastNameNode = this.xmlDoc.CreateElement("lastname");
    lastNameNode.InnerText = lastName;
    contactNode.AppendChild(lastNameNode);

    XmlNode phoneNode = this.xmlDoc.CreateElement("phone");
    phoneNode.InnerText = phone;
    contactNode.AppendChild(phoneNode);

    // Kontakt-Knoten dem Dokument-Element anfügen
    xmlDoc.DocumentElement.AppendChild(contactNode);
}

```

Listing 241: Methode zum Hinzufügen eines Kontakts

Ändern von Elementen

Die Methode `ChangeContact` erlaubt das Ändern der Daten eines Kontakts. Diese Methode sucht über `FindContact` zunächst das `contact`-Element, dessen Attribut `id` den übergebenen `Id`-Wert speichert. Wird genau ein Element gefunden, referenziert `ChangeContact` die jeweiligen Unterelemente und ändert deren Inhalt. Für den (unwahrscheinlichen, aber dennoch möglichen) Fall, dass ein Unterelement nicht gefunden wird, wird dieses einfach neu erzeugt und dem `contact`-Element zugeordnet.

Wird kein oder werden mehrere `contact`-Elemente mit der übergebenen `Id` gefunden, erzeugt `ChangeContact` eine Ausnahme.

```

public void ChangeContact(int id, string firstName, string lastName,
    string phone)
{
    // Knoten mit der übergebenen Id suchen
    XmlNodeList contactNodes = this.FindContact(id);

    // Überprüfen, ob genau ein Knoten gefunden wurde
    if (contactNodes.Count == 1)
    {
        // Knoten verändern
        XmlNode contactNode = contactNodes[0];

        // Unterknoten firstname suchen und dessen Text anpassen
        XmlNode firstNameNode = contactNode.SelectSingleNode("firstname");
        if (firstNameNode == null)
        {
            // firstname-Element nicht gefunden, also neu anlegen
            firstNameNode = this.xmlDoc.CreateElement("firstname");
            this.xmlDoc.AppendChild(firstNameNode);
        }
        firstNameNode.InnerText = firstName;

        // Unterknoten lastname suchen und ändern
        XmlNode lastNameNode = contactNode.SelectSingleNode("lastname");
        if (lastNameNode == null)
        {
            lastNameNode = this.xmlDoc.CreateElement("lastname");
            this.xmlDoc.AppendChild(lastNameNode);
        }
        lastNameNode.InnerText = lastName;

        // Unterknoten phone suchen und ändern
        XmlNode phoneNode = contactNode.SelectSingleNode("phone");
        if (phoneNode == null)
        {
            // phone-Element nicht gefunden, also neu anlegen
            phoneNode = this.xmlDoc.CreateElement("phone");
            this.xmlDoc.AppendChild(phoneNode);
        }
        phoneNode.InnerText = phone;
    }
    else
    {
        // Kein oder mehrere Knoten gefunden: Ausnahme werfen
        if (contactNodes.Count == 0)
            throw new XmlException("Kontakt mit der Id '" + id +
                "' nicht gefunden", null);
        else
            throw new XmlException("Es wurden mehr als ein (" +
                contactNodes.Count + ") Kontakt mit der Id '" +

```

Listing 242: Methode zum Ändern eines Kontakts

Basics

Datum
Zeit

Anwen-
dungen

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

```

        id + "' gefunden", null));
    }
}

```

Listing 242: Methode zum Ändern eines Kontakts (Forts.)

Löschen von Elementen

Die Methode `RemoveContact` löscht einen Kontakt aus dem Dokument. Dazu wird der Kontakt zunächst wieder über `FindContact` gesucht. `RemoveContact` fragt dabei wie `ChangeContact` ab, ob keins oder mehrere Elemente gefunden werden, und generiert in diesem Fall eine Ausnahme. Wird genau ein `contact`-Element gefunden, wird dieses über die `RemoveChild`-Methode des Wurzelknotens (der über die Eigenschaft `DocumentElement` referenziert werden kann) entfernt.

```

public void RemoveContact(int id)
{
    // Knoten mit der übergebenen Id suchen
    XmlNodeList contactNodes = this.FindContact(id);

    // Überprüfen, ob genau ein Element gefunden wurde
    if (contactNodes.Count == 1)
    {
        // Element löschen
        this.xmlDoc.DocumentElement.RemoveChild(contactNodes[0]);
    }
    else
    {
        // Kein oder mehrere Knoten gefunden: Ausnahme werfen
        if (contactNodes.Count == 0)
            throw new XmlException("Kontakt mit der Id '" + id +
                "' nicht gefunden", null);
        else
            throw new XmlException("Es wurden mehr als ein (" +
                contactNodes.Count + ") Kontakt mit der Id '" +
                id + "' gefunden", null);
    }
}

```

Listing 243: Methode zum Entfernen eines Kontakts

Auslesen von Elementen

Nun fehlen nur noch eine Methode zum Auslesen von Kontaktinformationen und eine zum Speichern des Dokuments. Die Methode `ReadContact` liest die Daten des Kontakts mit der übergebenen Id in drei `out`-Argumente. Das `contact`-Element wird dabei wieder über `FindContact` gesucht. Wird genau ein Element gefunden, referenziert die Methode dessen Unter-elemente und liest die Daten in die `out`-Argumente. Wird kein oder werden mehr als ein Element gefunden, generiert `ReadContact` eine Ausnahme.

```

public void ReadContact(int id, out string firstName,
    out string lastName, out string phone)
{
    // Knoten mit der übergebenen Id suchen
    XmlNodeList contactNodes = this.FindContact(id);

    // Überprüfen, ob genau ein Element gefunden wurde
    if (contactNodes.Count == 1)
    {
        // Unterelemente referenzieren und die Daten zurückgeben
        XmlNode firstNameNode =
            contactNodes[0].SelectSingleNode("firstname");
        if (firstNameNode != null)
            firstName = firstNameNode.InnerText;
        else
            firstName = null;

        XmlNode lastNameNode =
            contactNodes[0].SelectSingleNode("lastname");
        if (lastNameNode != null)
            lastName = lastNameNode.InnerText;
        else
            lastName = null;

        XmlNode phoneNode = contactNodes[0].SelectSingleNode("phone");
        if (phoneNode != null)
            phone = phoneNode.InnerText;
        else
            phone = null;
    }
    else
    {
        // Kein oder mehrere Knoten gefunden: Ausnahme werfen
        if (contactNodes.Count == 0)
            throw new XmlException("Kontakt mit der Id '" + id +
                "' nicht gefunden", null);
        else
            throw new XmlException("Es wurden mehr als ein (" +
                contactNodes.Count + ") Kontakt mit der Id '" +
                id + "' gefunden", null);
    }
}

```

Listing 244: Methode zum Lesen eines Kontakts

Speichern des Dokuments

Schließlich bleibt nur noch die Methode zum Speichern des Dokuments übrig, die die Save-Methode des XmlDocument-Objekts aufruft.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
public void Save()
{
    xmlDoc.Save(xmlFileName);
}
}
```

Listing 245: Methode zum Speichern des Dokuments

Beim Speichern berücksichtigt das `XmlDocument`-Objekt übrigens automatisch die über die XML-Deklaration definierte Codierung.

Eine Beispielanwendung

Fertig ist die Klasse `ContactHandler`. Eine Beispielanwendung zeigt Listing 246. Zum Kompilieren dieser Anwendung müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Text` und `System.Windows.Forms` importieren.

```
// Der Dateiname der XML-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "contacts.xml");

// Neues XML-Dokument erzeugen
ContactHandler ch = new ContactHandler(xmlFileName, true, Encoding.UTF8);

// Kontakt hinzufügen
ch.AddContact(1000, "Zap", "BBrox", "01234-12345");
ch.AddContact(1001, "Ford", "Prefect", "01234-12346");

// Dokument speichern
ch.Save();

// Dokument einlesen
ch = new ContactHandler(xmlFileName, false, null);

// Kontakt hinzufügen
ch.AddContact(1002, "Tricia", "McMillan", "01234-12347");

// Kontakt ändern
ch.ChangeContact(1000, "Zaphod", "Beeblebrox", "042-42");

// Kontakt löschen
ch.RemoveContact(1001);

// Kontakt auslesen
string firstName, lastName, phone;
ch.ReadContact(1000, out firstName, out lastName, out phone);
Console.WriteLine("Kontakt 1000:");
Console.WriteLine(firstName);
Console.WriteLine(lastName);
```

Listing 246: Beispielanwendung der Klasse `ContactHandler` zur Verwaltung eines XML-Dokuments über das DOM


```
Console.WriteLine(phone);  
  
// Dokument speichern  
ch.Save();
```

Listing 246: Beispielanwendung der Klasse `ContactHandler` zur Verwaltung eines XML-Dokuments über das DOM (Forts.)

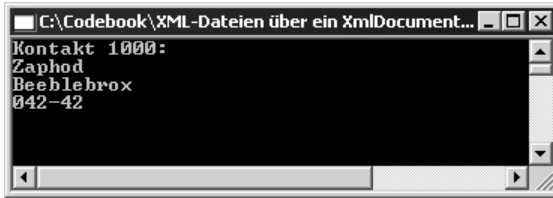


Abbildung 96: Das Beispielprogramm in Aktion

141 XML-Dateien über ein DataSet-Objekt erzeugen

Ein `DataSet`-Objekt (aus dem Namensraum `System.Data`) erlaubt das sehr einfache Erzeugen und Verändern von XML-Dokumenten, die eine tabellenähnliche (auf zwei Ebenen beschränkte) Struktur aufweisen. Zum Erzeugen müssen Sie lediglich das `DataSet` dynamisch im Speicher erzeugen, mit Daten füllen und schließlich die `WriteXml`-Methode aufrufen um die XML-Datei zu schreiben.

Um den Namen des Wurzelements festzulegen, können Sie nach der Erzeugung des `DataSet` diesen der Eigenschaft `DataSetName` zuweisen. Den zu verwendenden Namensraum und den Namensraum-Präfix können Sie über die Eigenschaften `Namespace` und `Prefix` festlegen.

Der Programmcode in Listing 247 erzeugt eine der im Rezept 132 eingesetzten XML-Datei ähnliche Datei zur Speicherung von Personendaten. Nach der Erzeugung des `DataSet`-Objekts fügt das Programm diesem ein `DataTable`-Objekt an und definiert die einzelnen Spalten der Tabelle. Über die `Rows`-Auflistung werden dann einzelne Personen-Datensätze hinzugefügt. Beim Schreiben der XML-Datei übergibt das Programm am zweiten Argument der `WriteXml`-Methode den Wert `XmlWriteMode.WriteSchema`, um dem Dokument ein eingebettetes Schema hinzuzufügen. Dabei entstehende Ausnahmen werden abgefangen.

Um zu verhindern, dass eine bereits existierende Datei ohne Warnung überschrieben wird, fragt das Programm zu Beginn ab, ob eine solche existiert, und lässt den Anwender entscheiden, ob die Datei überschrieben werden soll.

Das Beispiel erfordert die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Xml` und `System.Windows.Forms`.

```
// Der Dateiname der XML-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// Überprüfen, ob die Datei existiert
if (File.Exists(xmlFileName))
{
    if (MessageBox.Show("Die Datei '" + xmlFileName +
        "' existiert bereits.\r\n\r\nWollen Sie diese Datei überschreiben?",
        Application.ProductName, MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.No)
        return;
}

// DataSet erzeugen und initialisieren
DataSet dataSet = new DataSet();
dataSet.DataSetName = "persons";
dataSet.Namespace = "http://www.addison-wesley.de/codebook/persons";
dataSet.Prefix = "per";

// Tabelle erzeugen
DataTable dataTable = dataSet.Tables.Add("person");

// Spalten definieren
dataTable.Columns.Add("id", typeof(int));
dataTable.Columns.Add("firstname", typeof(string));
dataTable.Columns.Add("lastname", typeof(string));
dataTable.Columns.Add("type", typeof(string));

// Zeilen anfügen
dataTable.Rows.Add(new object[] {1000, "Zaphod", "Beeblebrox", "Alien"});
dataTable.Rows.Add(new object[] {1001, "Ford", "Prefect", "Alien"});
dataTable.Rows.Add(new object[] {1002, "Tricia", "McMillan", "Earthling"});
dataTable.Rows.Add(new object[] {1003, "Arthur", "Dent", "Earthling"});

// XML-Datei mit Schema schreiben
try
{
    dataSet.WriteXml(xmlFileName, XmlWriteMode.WriteSchema);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Schreiben der XML-Datei: " + ex.Message,
        Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}
```

Listing 247: Erzeugen eines XML-Dokuments über ein DataSet-Objekt

142 XML-Dateien über ein DataSet ändern

Das Verändern eines XML-Dokuments, das eine tabellenähnliche Struktur (mit Elementen in zwei Ebenen) aufweist, ist über ein `DataSet`-Objekt sehr einfach, da Sie dazu die Methoden und Eigenschaften der `DataSet`- und der `DataTable`-Klasse verwenden können. Das folgende Listing zeigt dies am Beispiel der Personen-XML-Datei, die im Rezept 142 erstellt wird. In diesem Beispiel werden eine Person hinzugefügt, die Person mit der Id 1003 gelöscht und die Daten der Person mit der Id 1001 geändert.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Data`, `System.IO` und `System.Windows.Forms` importieren.

```
// Der Dateiname der XML-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// DataSet erzeugen und XML-Datei mit Schema einlesen
DataSet dataSet = new DataSet();
try
{
    dataSet.ReadXml(xmlFileName, XmlReadMode.ReadSchema);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Lesen der XML-Datei: " + ex.Message,
        Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

// DataTable referenzieren
DataTable personTable = dataSet.Tables["person"];

if (personTable == null)
{
    MessageBox.Show("Das XML-Dokument enthält keine person-Elemente",
        Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

// Zeile anfügen
personTable.Rows.Add(new object[] {1004, "Marvin", "", "Robot"});

// Zeile suchen und löschen
DataRow[] rows = personTable.Select("id = 1003");
for (int i = 0; i < rows.Length; i++)
    rows[i].Delete();

// Zeile suchen und aktualisieren
rows = personTable.Select("id = 1001");
```

Listing 248: Verändern einer XML-Datei über ein `DataSet`-Objekt

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

if (rows.Length == 1)
{
    rows[0]["type"] = "Traveller";
}
else
{
    if (rows.Length == 0)
        Console.WriteLine("Person mit der Id 1001 nicht gefunden");
    else
        Console.WriteLine("Mehrere Personen mit der Id 1001 gefunden");
}

// XML-Datei mit Schema (unter einem anderen Namen ) schreiben
try
{
    xmlFileName = xmlFileName.Replace("Persons", "Persons_New");
    dataSet.WriteXml(xmlFileName, XmlWriteMode.WriteSchema);
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Schreiben der XML-Datei: " + ex.Message,
        Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

```

Listing 248: Verändern einer XML-Datei über ein DataSet-Objekt (Forts.)

143 Binäre Daten in einer XML-Datei speichern

In XML-Dokumenten können Sie nicht nur Textdaten speichern, sondern auch binäre. Das Prinzip dazu ist ganz einfach: Sie müssen die binären Daten lediglich über das Base64-Verfahren in einen Text umwandeln, der im 7-Bit-ASCII-Zeichensatz codiert ist. Dazu können Sie beim Schreiben über ein `XmlTextWriter`-Objekt einfach die Methode `WriteBase64` verwenden. Verwenden Sie ein `XmlDocument`-Objekt, müssen Sie die Daten über `Convert.ToBase64String` selbst umwandeln.

Das Beispiel in Listing 249 erzeugt ein neues XML-Dokument mit einem `person`-Element, in dem neben dem Vor- und dem Nachnamen auch ein Bild der Person gespeichert wird.

Das Programm erfordert die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import der Namensräume `System`, `System.IO`, `System.Xml` und `System.Windows.Forms`.

```

// Der Dateiname
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// Neues XML-Dokument erzeugen
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml("<?xml version=\"1.0\" encoding=\"utf-8\" \" +
    \"standalone=\"yes\"?><persons></persons>");

```

Listing 249: Speichern von binären Daten in einer XML-Datei

```
// Ein person-Element erzeugen
XmlNode personNode = xmlDoc.CreateElement("person");
xmlDoc.DocumentElement.AppendChild(personNode);

// Vor- und Nachname schreiben
XmlNode subNode = xmlDoc.CreateElement("firstname");
subNode.InnerText = "Zaphod";
personNode.AppendChild(subNode);
subNode = xmlDoc.CreateElement("lastname");
subNode.InnerText = "Beeblebrox";
personNode.AppendChild(subNode);

// Das Bild einlesen
string imageFileName = Path.Combine(Application.StartupPath, "Zaphod.jpg");
FileInfo fi = new FileInfo(imageFileName);
byte[] buffer = new byte[fi.Length];
FileStream fs = new FileStream(imageFileName, FileMode.Open, FileAccess.Read);
fs.Read(buffer, 0, buffer.Length);

// Das Bild Base64-codieren ...
string codedFile = Convert.ToBase64String(buffer, 0, buffer.Length);

// ... und im XML-Dokument speichern
subNode = xmlDoc.CreateElement("image");
subNode.InnerText = codedFile;
personNode.AppendChild(subNode);

// XML-Datei speichern
xmlDoc.Save(xmlFileName);
```

Listing 249: Speichern von binären Daten in einer XML-Datei (Forts.)

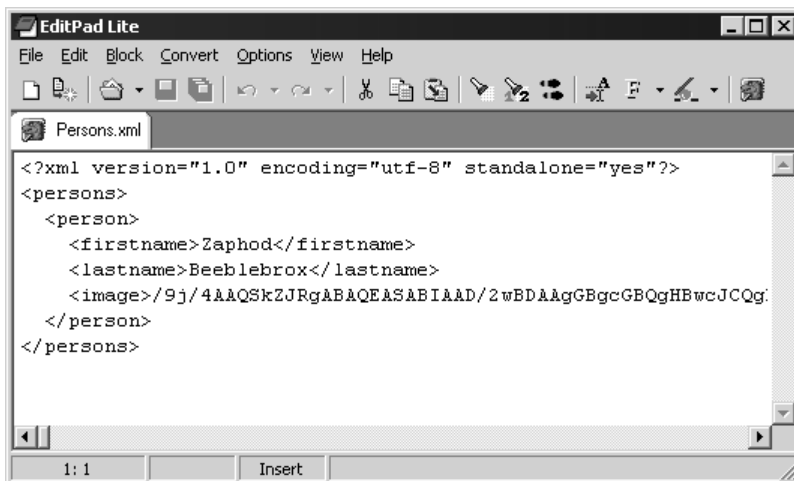


Abbildung 97: Die erzeugte XML-Datei in einem Editor

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

144 Base64-codierte Bilder (binäre Daten) aus XML-Dateien auslesen

Binäre Daten, die im Base64-Format in einer XML-Datei gespeichert sind, können Sie über die Methode `FromBase64String` der `Convert`-Klasse in ein `byte-Array` konvertieren. Dieses Array können Sie dann über ein `Stream`-Objekt weiterverarbeiten (z.B. über einen `FileStream` in eine Datei schreiben).

Das (Windows.Forms-)Beispiel in Listing 250 liest das Bild aus der im vorhergehenden Rezept erzeugten XML-Datei ein und erzeugt damit ein `MemoryStream`-Objekt. Dieses Objekt wird dem Konstruktor der `Bitmap`-Klasse übergeben um ein `Bitmap`-Objekt zu erzeugen. Die Referenz auf das `Bitmap`-Objekt wird dann in die `Image`-Eigenschaft eines `PictureBox`-Steuerelements geschrieben.

Zum Kompilieren des Beispiels müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.Xml`, `System.IO` und `System.Windows.Forms` importieren.

```
private void ReadImage_Click(object sender, System.EventArgs e)
{
    // XML-Datei einlesen
    string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(xmlFileName);
    XmlNode personNode = xmlDoc.SelectSingleNode(
        "persons/person[attribute::id='1000']");
    XmlNode imageNode = personNode.SelectSingleNode("image");

    // Base64-codierten String in Bitmap umwandeln
    byte[] buffer = Convert.FromBase64String(imageNode.InnerText);
    Bitmap bitmap = new Bitmap(new MemoryStream(buffer, 0, buffer.Length));

    pictureBox1.Image = bitmap;
}
```

Listing 250: Auslesen eines in einer XML-Datei gespeicherten Bildes



Abbildung 98: Das Programm hat das Bild aus der XML-Datei eingelesen

145 XML-Dokumente über XSL-Dokumente transformieren

XML-Dokumente müssen recht häufig in eine andere Form transformiert werden (z.B. in XHTML). Üblicherweise wird dazu XSL (Extensible Stylesheet Language) verwendet. XSL ist eine Sprache mit einer Vielzahl an speziellen Instruktionen, über die Sie die Daten eines XML-Dokuments sehr flexibel auslesen, transformieren und in ein Ergebnis-Dokument speichern können. Ein einfaches XSL-Dokument, das die Personen-XML-Datei aus dem Rezept 132 in eine XHTML-Datei transformiert, sieht z.B. so aus:

```
<?xml version="1.0"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">

<!-- Transformation nach HTML -->

<body>
<h1>Personen</h1>
<p>

<xsl:for-each select = 'persons/person'>
  <xsl:sort select='lastname' order='ascending' />
  <xsl:value-of select='lastname' />,
  <xsl:value-of select='firstname' />:
  <xsl:value-of select='type' />
<br/>

</xsl:for-each>
</p>

</body>

</html>
```

Listing 251: Ein XSL-Dokument zur Transformation der Personen-XML-Datei nach XHTML

Das (etwas unschöne) aus den transformierten XML-Daten erzeugte HTML-Dokument sieht dann folgendermaßen aus:

```
<html>
  <body>
    <h1>Personen</h1>
    <p>Beeblebrox,
Zaphod:
Alien<br>Dent,
Arthur:
Earthling<br>McMillan,
Tricia:
```

Listing 252: Die transformierte Personen-XML-Datei

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

Earthling<br>Prefect,
Ford:
Alien<br></p>
</body>
</html>

```

Listing 252: Die transformierte Personen-XML-Datei (Forts.)

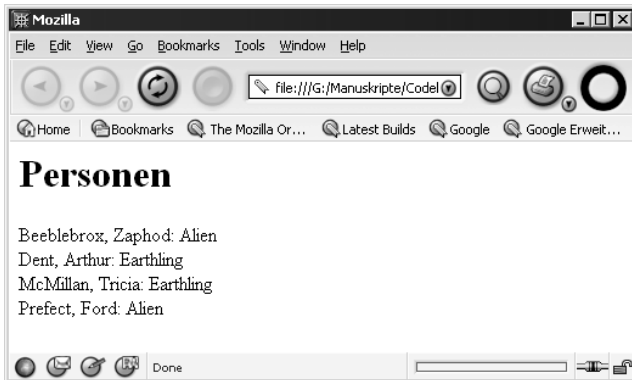


Abbildung 99: Die erzeugte HTML-Datei in einem Webbrowser

Eine solche Transformation können Sie über eine Instanz der Klasse `XslTransform` aus dem Namensraum `System.Xml.Xsl` vornehmen.

Die Methode `TransformXmlFile` in Listing 253 erzeugt aus einem XML-Dokument über ein XSL-Dokument eine Ergebnisdatei. Die entsprechenden Dateinamen werden dieser Methode übergeben. `TransformXmlFile` erzeugt zunächst eine Instanz der `XslTransform`-Klasse und lädt die XSL-Datei über deren `Load`-Methode. Die Transformation wird dann über die Methode `Transform` ausgeführt.

`TransformXmlFile` erfordert den Import der Namensräume `System`, `System.Xml` und `System.Xml.Xsl`.

```

public static void TransformXmlFile(string xmlSourceFileName,
    string xslFileName, string xmlDestFileName)
{
    // Instanz der Klasse XslTransform erzeugen
    XslTransform xslTransform = new XslTransform();

    // XSL-Datei laden
    xslTransform.Load(xslFileName);

    // XML-Datei transformieren
    xslTransform.Transform(xmlSourceFileName, xmlDestFileName);
}

```

Listing 253: Methode zur Transformation einer XML-Datei

146 Den Namen des Computers ermitteln

Den Namen des Computers, auf dem ein .NET-Programm ausgeführt wird, können Sie über die Eigenschaft `MachineName` der `Environment`-Klasse aus dem Namensraum `System` auslesen:

```
string machineName = System.Environment.MachineName;
```

147 Systempfade herausfinden

Der Pfad zum Windows-System-Ordner steht Ihnen über die Eigenschaft `SystemDirectory` der Klasse `System.Environment` zur Verfügung. Eine Möglichkeit, das Windows-Verzeichnis selbst auszulesen, fehlt, wahrscheinlich deshalb, weil die `Environment`-Klasse betriebssystemübergreifend funktionieren soll. Den Pfad zum Windows-Verzeichnis können Sie aber recht einfach über die API-Funktion `GetWindowsDirectory` auslesen. Diese Funktion erwartet am ersten Argument normalerweise einen Zeiger auf einen ausreichend groß dimensionierten (C-)String. In C# können Sie an diesem Argument ein `StringBuilder`-Objekt übergeben, das Sie zuvor mit der maximal möglichen Anzahl Zeichen (bei Pfaden zurzeit 260) initialisieren müssen. Das Beispielprogramm initialisiert die Größe auf ein Zeichen mehr, da die API-Funktion den String mit einem 0-Zeichen abschließt. Die Größe dieses Puffers übergeben Sie `GetWindowsDirectory` am zweiten Argument.

Über die Methode `GetFolderPath` der `Environment`-Klasse können Sie den Pfad zu weiteren Systemordnern ermitteln. Diese Methode erwartet als Argument einen Wert der Aufzählung `Environment.SpecialFolder` (Tabelle 10).

Konstante	Bedeutung
<code>ApplicationData</code>	Pfad zum Verzeichnis für programmspezifische Daten für Benutzer, deren Profil in einem Netzwerk-Ordner gespeichert und beim Einloggen auf den lokalen Computer kopiert wird
<code>CommonApplicationData</code>	Pfad zum Verzeichnis für programmspezifische Daten für alle Benutzer
<code>CommonProgramFiles</code>	Pfad zum Verzeichnis für Komponenten, die von mehreren Anwendungen verwendet werden
<code>Cookies</code>	Pfad zum Verzeichnis, in dem Cookies gespeichert werden
<code>DesktopDirectory</code>	Pfad zum Desktop-Verzeichnis des aktuellen Benutzers
<code>Favorites</code>	Pfad zum Favoriten-Verzeichnis des aktuellen Benutzers
<code>History</code>	Pfad zum Internet-Verlaufs-Verzeichnis des aktuellen Benutzers

Tabelle 10: Die Werte der Aufzählung `SpecialFolder`

Konstante	Bedeutung
InternetCache	Pfad zum Verzeichnis für temporäre Internetdateien
LocalApplicationData	Pfad zum Verzeichnis für programmspezifische Daten für Benutzer, deren Profil lokal verwaltet wird
Personal	Pfad zum Verzeichnis für die eigenen Dateien des aktuellen Benutzers
ProgramFiles	Pfad zum Verzeichnis, in dem per Voreinstellung alle Programme installiert werden
Programs	Pfad zum Verzeichnis, in dem die Programmgruppen des aktuellen Benutzers verwaltet werden
Recent	Pfad zum Verzeichnis, in dem Verweise auf die zuletzt geöffneten Dokumente für den aktuellen Benutzer verwaltet werden
SendTo	Pfad zum Verzeichnis, in dem die Verweise verwaltet werden, die im Senden-An-Kontextmenüeintrag erscheinen
StartMenu	Pfad zum Verzeichnis, in dem die Startmenüeinträge des aktuellen Benutzers verwaltet werden
Startup	Pfad zum Autostart-Verzeichnis
System	Pfad zum Windows-System32-Verzeichnis
Templates	Pfad zum Verzeichnis, in dem Dokumentenvorlagen verwaltet werden

Tabelle 10: Die Werte der Aufzählung *SpecialFolder* (Forts.)

Listing 254 liest einige Systempfade und gibt diese an der Konsole aus. Das Programm benötigt den Import der Namensräume `System`, `System.Text` und `System.Runtime.InteropServices`.

```

/* Deklaration der API-Funktion GetWindowsDirectory */
[DllImport("kernel32.dll")]
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,
    uint uSize);

...

// Windows-Systempfad auslesen
string windowsSystemPath = System.Environment.SystemDirectory;

// Windows-Pfad auslesen
const int MAX_PATH = 260;
StringBuilder buffer = new StringBuilder(MAX_PATH + 1);
string windowsPath = null;
if (GetWindowsDirectory(buffer, MAX_PATH + 1) > 0)
    windowsPath = buffer.ToString();

// Spezielle Systemordner ermitteln
string programPath = Environment.GetFolderPath(
    Environment.SpecialFolder.ProgramFiles);

```

Listing 254: Auslesen einiger Windows-Systempfade

```

string desktopPath = Environment.GetFolderPath(
    Environment.SpecialFolder.DesktopDirectory);
string applicationDataPath = Environment.GetFolderPath(
    Environment.SpecialFolder.ApplicationData);
string recentPath = Environment.GetFolderPath(
    Environment.SpecialFolder.Recent);

Console.WriteLine("Windows-Pfad: {0}", windowsPath);
Console.WriteLine("Windows-System-Pfad: {0}", windowsSystemPath);
Console.WriteLine("Programmordner: {0}", programPath);
Console.WriteLine("Desktop: {0}", desktopPath);
Console.WriteLine("Anwendungsdaten: {0}", applicationDataPath);
Console.WriteLine("Zuletzt geöffnete Dokumente: {0}", recentPath);

```

Listing 254: Auslesen einiger Windows-Systempfade (Forts.)

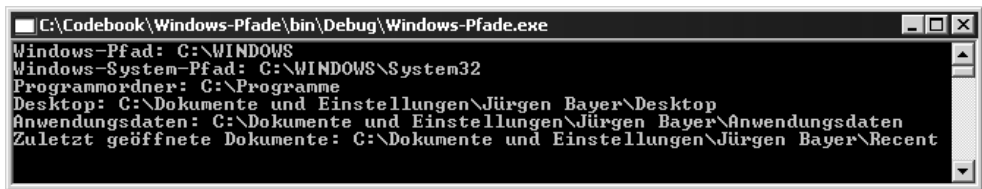


Abbildung 100: Das Programm zum Auslesen einiger Systempfade

148 Windows-Version auslesen

Die Version des aktuellen Betriebssystems können Sie über das `Version`-Objekt auslesen, das `System.Environment.OSVersion.Version` referenziert. Die einzelnen Teile der Version stehen über die Eigenschaften `Major`, `Minor`, `Revision` und `Build` zur Verfügung. Die Eigenschaft `Platform` liefert einen String mit der Kurzbezeichnung des Betriebssystems. Für die NT-Linie von Microsoft gibt diese Eigenschaft den String »Win32NT« zurück.

```

using System;

...

Console.WriteLine("Version:");
Console.WriteLine("Major: {0}", Environment.OSVersion.Version.Major);
Console.WriteLine("Minor: {0}", Environment.OSVersion.Version.Minor);
Console.WriteLine("Revision: {0}", Environment.OSVersion.Version.Revision);
Console.WriteLine("Build: {0}", Environment.OSVersion.Version.Build);
Console.WriteLine("Plattform: {0}", Environment.OSVersion.Platform);

```

Listing 255: Ermitteln der Windows-Version

Dummerweise ist es über diese Klasse nicht allzu einfach, die genaue Version des Betriebssystems herauszufinden. Über die Versionsnummer können Sie zwar herausfinden, auf welcher Windows-Version das Programm ausgeführt wird. Windows 2000 besitzt z.B. die Versions-



Abbildung 101: Ermitteln der Windows-Version auf einem Windows-XP-Rechner

nummer 5.0.x.x, Windows XP die Versionsnummer 5.1.x.x. Im nächsten Rezept finden Sie eine Lösung zur Ermittlung der Windows-Hauptversion.

149 Windows-Hauptversion ermitteln

Die Methode `GetWindowsMainVersion` in Listing 256 ermittelt die Windows-Hauptversion basierend auf den Angaben, die Microsoft an der Adresse msdn.microsoft.com/library/en-us/sysinfo/base/osversioninfo_str.asp veröffentlicht. Um die Version in einer flexibel verarbeitbaren Form zurückzugeben, habe ich zunächst eine Aufzählung `WindowsMainVersion` implementiert, die die möglichen Windows-Versionen als Konstanten beinhaltet. Die Konstante `Unknown` steht dafür, dass die Version nicht ermittelt werden konnte (weil es sich z.B. um eine neuere Version handelt).

Zum Kompilieren des Quellcodes müssen Sie den Namensraum `System` importieren.

```

/* Aufzählung für die Windows-Hauptversion */
public enum WindowsMainVersion
{
    WindowsNT35,
    Windows95,
    Windows98,
    WindowsMe,
    WindowsNT4,
    Windows2000,
    WindowsXP,
    WindowsServer2003,
    Unknown
}

/* Methode zum Ermitteln der Windows-Hauptversion */
public static WindowsMainVersion GetWindowsMainVersion()
{
    switch (Environment.OSVersion.Version.Major)
    {
        case 3:
            return WindowsMainVersion.WindowsNT35;
        case 4:
            if (Environment.OSVersion.Version.Minor == 0)

```

Listing 256: Methode zur Ermittlung der Windows-Hauptversion

```

        if (Environment.OSVersion.Platform ==
            PlatformID.Win32NT)
            return WindowsMainVersion.WindowsNT4 ;
        else
            return WindowsMainVersion.Windows95;
    else if (Environment.OSVersion.Version.Minor == 10)
        return WindowsMainVersion.Windows98;
    else if (Environment.OSVersion.Version.Minor == 90)
        return WindowsMainVersion.WindowsMe;
    else
        return WindowsMainVersion.Unknown;
case 5:
    if (Environment.OSVersion.Version.Minor == 0)
        return WindowsMainVersion.Windows2000;
    else if (Environment.OSVersion.Version.Minor == 1)
        return WindowsMainVersion.WindowsXP;
    else if (Environment.OSVersion.Version.Minor == 2)
        return WindowsMainVersion.WindowsServer2003;
    else
        return WindowsMainVersion.Unknown;
default:
    return WindowsMainVersion.Unknown;
}
}

```

Listing 256: Methode zur Ermittlung der Windows-Hauptversion (Forts.)

150 Die Prozessorgeschwindigkeit ermitteln

Die Geschwindigkeit des (ersten) Prozessors eines Computers können Sie über WMI ermitteln. Dazu erzeugen Sie eine Instanz der Klasse `System.Management.ManagementObject`, der Sie im Konstruktor den String »Win32_Processor.DeviceID='CPU0'« übergeben um eine Instanz der WMI-Klasse `Win32_Processor` für den ersten Prozessor des Systems zu erzeugen. Über den Indexer können Sie dann die Prozessorgeschwindigkeit auslesen, indem Sie diesem als Index den String »CurrentClockSpeed« übergeben. Den zurückgegebenen `object`-Wert sollten Sie dann noch in einen `uint`-Wert konvertieren.

Die Methode `GetProcessorSpeed` in Listing 257 macht genau das. Für den Fall, dass die Prozessorgeschwindigkeit nicht verfügbar ist, gibt `GetProcessorSpeed` 0 zurück. Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Management.dll` referenzieren und die Namensräume `System` und `System.Management` einbinden.

```

public static uint GetProcessorSpeed()
{
    // Win32_Processor-Instanz für den ersten Prozessor erzeugen und die
    // Geschwindigkeit abfragen
    ManagementObject mo = new ManagementObject(
        "Win32_Processor.DeviceID='CPU0'");

```

Listing 257: Methode zur Ermittlung der Prozessorgeschwindigkeit

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

uint currentClockSpeed = 0;
try
{
    currentClockSpeed = (uint)(mo["CurrentClockSpeed"]);
}
catch {}

// Speicher des WMI-Objekts freigeben um den Arbeitsspeicher
// möglichst schnell zu entlasten
mo.Dispose();

return currentClockSpeed;
}

```

Listing 257: Methode zur Ermittlung der Prozessorgeschwindigkeit (Forts.)

Das Programm funktioniert nur dann, wenn WMI auf dem Computer, auf dem es ausgeführt wird, installiert sein. Außerdem muss das aktuelle Benutzerkonto über ausreichende WMI- und Systemrechte verfügt. Falls dies nicht der Fall ist, können Sie vor der Ausführung dieser Methode das Programm mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 211 beschreibe.

151 Informationen über den Arbeitsspeicher auslesen

Informationen über den Arbeitsspeicher des Systems erhalten Sie relativ einfach über WMI. Dazu fragen Sie über eine Instanz der Klasse `System.Management.ManagementObject` ein Objekt der WMI-Klasse `Win32_OperatingSystem` ab. Über verschiedene Eigenschaften dieser Klasse erhalten Sie (u.a.) Informationen zum Arbeitsspeicher.

Die Methode `GetRamInformations` in Listing 258 liest diese Informationen aus dem lokalen System in eine Instanz der Klasse `RamInformations` aus und gibt eine Referenz darauf zurück. `GetRamInformations` erzeugt dazu eine Auflistung der WMI-Objekte, die die auf einem System installierten Betriebssysteme repräsentieren, und fragt das erste dieser Objekte ab (das das aktuelle System repräsentiert).

Zum Kompilieren dieses Quellcodes benötigt das Programm eine Referenz auf die Assembly `System.Management.dll`. Außerdem müssen Sie die Namensräume `System` und `System.Management` einbinden. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```

/* Klasse zur Speicherung von Arbeitsspeicher-Informationen */
public class RamInformations
{
    public ulong FreePhysicalMemory = 0;
    public ulong FreeSpaceInPagingFiles = 0;
    public ulong FreeVirtualMemory = 0;
    public ulong MaxProcessMemorySize = 0;
}

```

Listing 258: Methode zur Ermittlung von Informationen zum Arbeitsspeicher

```

    public ulong TotalVirtualMemorySize = 0;
    public ulong TotalVisibleMemorySize = 0;
}

/* Methode zur Ermittlung von Informationen zum Arbeitsspeicher */
public static RamInformations GetRamInformations()
{
    // Instanz der Klasse RamInformations erzeugen
    RamInformations ri = new RamInformations();

    // Speicher-Infos über WMI abfragen
    ManagementClass mc = new ManagementClass("Win32_OperatingSystem");
    ManagementObjectCollection moc = mc.GetInstances();
    ManagementObjectCollection.ManagementObjectEnumerator
        mocEnumerator = moc.GetEnumerator();
    mocEnumerator.MoveNext();
    ManagementObject mo = (ManagementObject)mocEnumerator.Current;

    try{ri.FreePhysicalMemory = (ulong)mo["FreePhysicalMemory"];}
    catch {}
    try{ri.FreeSpaceInPagingFiles = (ulong)mo["FreeSpaceInPagingFiles"];}
    catch {}
    try{ri.FreeVirtualMemory = (ulong)mo["FreeVirtualMemory"];}
    catch {}
    try{ri.MaxProcessMemorySize = (ulong)mo["MaxProcessMemorySize"];}
    catch {}
    try
    {ri.TotalVirtualMemorySize = (ulong)mo["TotalVirtualMemorySize"];}
    catch {}
    try{ri.TotalVisibleMemorySize = (ulong)mo["TotalVisibleMemorySize"];}
    catch {}

    // Speicher der WMI-Objekte freigeben um den Arbeitsspeicher möglichst
    // schnell zu entlasten
    mo.Dispose();
    moc.Dispose();
    mc.Dispose();

    // Informationen zurückgeben
    return ri;
}

```

Listing 258: Methode zur Ermittlung von Informationen zum Arbeitsspeicher (Forts.)

Voraussetzung für die fehlerfreie Ausführung dieser Methode ist, dass auf dem Zielsystem WMI installiert ist und dass das Benutzerkonto, das aktuell verwendet wird, über ausreichende WMI- und Systemrechte verfügt. Falls das interaktive Konto nicht über ausreichend Rechte verfügt, können Sie vor der Ausführung dieser Methode das Programm mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 211 beschreibe.

Listing 258 zeigt eine Anwendung dieser Methode, bei der die ermittelten Daten an der Konsole ausgegeben werden.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
RamInformation ri = GetRamInformations();

Console.WriteLine("FreePhysicalMemory: {0}", ri.FreePhysicalMemory);
Console.WriteLine("FreeSpaceInPagingFiles: {0}", ri.FreeSpaceInPagingFiles);
Console.WriteLine("FreeVirtualMemory: {0}", ri.FreeVirtualMemory);
Console.WriteLine("MaxProcessMemorySize: {0}", ri.MaxProcessMemorySize);
Console.WriteLine("TotalVirtualMemorySize: {0}", ri.TotalVirtualMemorySize);
Console.WriteLine("TotalVisibleMemorySize: {0}", ri.TotalVisibleMemorySize);
```

Listing 259: Anwendung der Methode zur Ermittlung von Informationen zum Arbeitsspeicher



Abbildung 102: Das Beispielprogramm zum Auslesen von Arbeitsspeicher-Informationen in Aktion

152 Die Version des aktuellen Service-Packs ermitteln

Über die WMI-Klasse `Win32_OperatingSystem` können Sie Informationen zum aktuell auf dem System installierten Service Pack auslesen. Dazu stehen Ihnen die Eigenschaften `ServicePackMajorVersion` und `ServicePackMinorVersion` zur Verfügung, die beides `ushort`-Werte sind.

Die Methode `GetServicePackVersion` ermittelt eine Instanz der Klasse `Win32_OperatingSystem` für das aktive Betriebssystem, liest diese Eigenschaften aus und gibt eine Instanz der Klasse `OSVersion` mit Informationen zu der Version zurück. Ich verzichte auf eine nähere Beschreibung dieser Methode, da die Vorgehensweise sich prinzipiell nicht von der des Rezepts 151 unterscheidet.

Zum Kompilieren der Methode müssen Sie die Assembly `System.Management.dll` referenzieren und die Namensräume `System` und `System.Management` einbinden. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
/* Klasse für die Rückgabe der Version */
public class OSVersion
{
    public ushort Major;
    public ushort Minor;
}
```

Listing 260: Methode zur Ermittlung der Version des aktuellen Service Packs

```

/* Methode zum Ermitteln der Version des aktuellen Service Pack */
public static OSVersion GetServicePackVersion()
{
    // Version-Instanz für die Rückgabe erzeugen
    OSVersion version = new OSVersion();

    // Win32_OperatingSystem-Instanz für das aktive Betriebssystem ermitteln
    ManagementClass mc = new ManagementClass("Win32_OperatingSystem");
    ManagementObjectCollection moc = mc.GetInstances();
    ManagementObjectCollection.ManagementObjectEnumerator
        mocEnumerator = moc.GetEnumerator();
    mocEnumerator.MoveNext();
    ManagementObject mo = (ManagementObject)mocEnumerator.Current;

    // Die Version abfragen und speichern
    try{version.Major =(ushort)mo["ServicePackMajorVersion"];}
    catch {}
    try{version.Minor =(ushort)mo["ServicePackMinorVersion"];}
    catch {}

    // WMI-Objekte freigeben um den Speicher möglichst schnell zu entlasten
    mo.Dispose();
    moc.Dispose();
    mc.Dispose();

    // Ergebnis zurückgeben
    return version;
}

```

Listing 260: Methode zur Ermittlung der Version des aktuellen Service Packs (Forts.)

Wie bereits in den vorhergehenden Rezepten benötigt das Programm eine Referenz auf die Assembly *System.Management.dll*. Damit es fehlerfrei ausgeführt werden kann, muss auf dem Zielsystem WMI installiert sein. Außerdem muss das Benutzerkonto, das aktuell verwendet wird, über ausreichende WMI- und Systemrechte verfügt. Falls das aktuelle Benutzerkonto nicht über ausreichend Rechte verfügt, können Sie vor der Ausführung der *GetServicePackVersion*-Methode das Programm mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 211 beschreibe.

153 Installierte Programme auflisten

Das Auflisten aller Programme, die auf einem System installiert sind, ist leider nicht allzu einfach. Über die WMI-Klasse *Win32_Product* erhalten Sie lediglich Informationen zu Programmen, die über den Windows-Installer installiert wurden. Viele Installations-Tools arbeiten aber unabhängig vom Windows-Installer. Die mit diesen Tools installierten Anwendungen werden von der Klasse *Win32_Product* nicht berücksichtigt.

Natürlich können Sie nicht jedes installierte Programm finden. Problematisch werden z.B. .NET-Programme, die durch einfaches Kopieren auf ein System übertragen wurden. Sinnvoll

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateien

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

wäre aber wenigstens das Auflisten der Programme, die in Windows als deinstallierbar registriert sind und in der Software-Liste der Systemsteuerung auftauchen.

Um diese Programme zu ermitteln, können Sie die Registry abfragen. Die deinstallierbaren Programme finden Sie im Schlüssel `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall`. Zum Zugriff auf die Registry können Sie die WMI-Klasse `StdRegProv` aus dem Namensraum `root\default` verwenden. Die Verwendung dieser Klasse würde den Vorteil bringen, auch die auf entfernten Systemen installierten Programme auflisten zu können. Abgesehen davon, dass die `StdRegProv`-Klasse in Windows 95, 98 und Me (laut der Dokumentation) nicht zur Verfügung steht, ist deren Anwendung auch nicht allzu einfach. Problematisch ist z.B. die (mir nicht gelungene) By-Reference-Übergabe eines Arrays als COM-Variant-Typ beim Aufruf der WMI-Methode `EnumKey`. Also habe ich eine auf dem lokalen System funktionierende Lösung über den direkten Zugriff auf die Registry über eine `RegistryKey`-Instanz (siehe Rezept 65) implementiert.

Die Methode `EnumInstalledPrograms` in Listing 261 erzeugt deshalb ein solches Objekt für den Schlüssel, der die Informationen zu der deinstallierbaren Software enthält. Über die Methode `GetSubKeyNames` ermittelt `EnumInstalledPrograms` die Namen der in diesem Schlüssel enthaltenen Unterschlüssel, geht diese durch und öffnet jeden Unterschlüssel über `OpenSubKey`. Um den Namen des Programms zu ermitteln, liest `EnumInstalledPrograms` zunächst den Wert `DisplayName` aus, der bei echten Programmen in der Regel den anzuzeigenden Programmnamen enthält (bei installierten Komponenten, die keine Programme darstellen, existiert dieser Wert nicht). Ist `DisplayName` nicht vorhanden, versucht `EnumInstalledPrograms` noch den eventuell vorhandenen Wert `QuietDisplayName` auszulesen. Kann aus einem dieser Werte ein Name ausgelesen werden, wird das als installiertes Programm gewertet und der Name der Ergebnis-Auflistung hinzugefügt.

Die Methode erfordert die Referenzierung der Assembly `System.Management.dll` und die Einbindung der Namensräume `System`, `System.Management`, `System.Collections.Specialized` und `Microsoft.Win32`. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
public static StringCollection EnumInstalledPrograms()
{
    // StringCollection für die Rückgabe erzeugen
    StringCollection installedPrograms = new StringCollection();

    // RegistryKey-Instanz für den Schlüssel HKEY_LOCAL_MACHINE\Software\
    // Microsoft\Windows\CurrentVersion\Uninstall erzeugen
    string keyPath = @"Software\Microsoft\Windows\CurrentVersion\Uninstall";
    RegistryKey regKey = Registry.LocalMachine.OpenSubKey(keyPath);
    if (regKey != null)
    {
        // Alle Unterschlüssel durchgehen
        string[] subKeyNames = regKey.GetSubKeyNames();
        for (int i = 0; i < subKeyNames.Length; i++)
        {
```

Listing 261: Methode zur Auflistung der installierten (und deinstallierbaren) Anwendungen

```

        // Unterschlüssel öffnen und den Wert DisplayName oder
        // QuietDisplayName auslesen
        RegistryKey subKey = regKey.OpenSubKey(subKeyNames[i]);
        string programName = (string)subKey.GetValue("DisplayName");
        if (programName == null)
            programName = (string)subKey.GetValue("QuietDisplayName");

        // Wenn ein Programmname ermittelt werden konnte, diesen an die
        // String-Auflistung anfügen
        if (programName != null)
            installedPrograms.Add(programName);
    }
}
else
{
    // Schlüssel nicht gefunden: Ausnahme generieren
    throw new Exception("Registry-Schlüssel " +
        Registry.LocalMachine.Name + "\\\" + keyPath + " nicht gefunden");
}

return installedPrograms;
}

```

Listing 261: Methode zur Auflistung der installierten (und deinstallierbaren) Anwendungen (Forts.)

Das folgende Listing zeigt eine Beispielanwendung:

```

try
{
    StringCollection installedPrograms = EnumInstalledPrograms();
    foreach (string programName in installedPrograms)
        Console.WriteLine(programName);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 262: Anwendung der Methode zur Auflistung der installierten Anwendungen

154 Informationen zu den parallelen Schnittstellen des Systems auslesen

Informationen über die parallelen Schnittstellen eines Systems erhalten Sie über die WMI-Klasse `Win32_ParallelPort`. Listing 263 ermittelt über diese Klasse die wichtigsten Informationen zu den parallelen Schnittstellen des lokalen Computers. Um diese speichern zu können, sind zunächst zwei Klassen und eine Aufzählung implementiert. Die Klasse `ParallelPortCapabilities` verwaltet die Eigenschaften einer parallelen Schnittstelle und wird als Typ der Eigenschaft `Capabilities` in der Klasse `ParallelPort` eingesetzt. Die Aufzählung `ParallelPortStatusInfo` wird zur Speicherung des Status eines Ports verwendet. Instan-

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges



Abbildung 103: Auflisten der installierten Programme

zen der Klasse `ParallelPort` speichern dann später die wichtigsten Daten einer parallelen Schnittstelle.

Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Management.dll` referenzieren und die Namensräume `System`, `System.Management` und `System.Collections` einbinden.

```

/* Klasse für die Fähigkeiten eines parallelen Ports */
public class ParallelPortCapabilities
{
    public bool XTATCompatible;
    public bool PS2Compatible;
    public bool ECP;
    public bool EPP;
    public bool PC98;
    public bool PC98Hireso;
    public bool PCH98;
    public bool Other;
}

/* Aufzählung für den Status eines parallelen Ports */
public enum ParallelPortStatusInfo: ushort
{
    NotDefined = 0,
    Other = 1,
    Unknown = 2,
    Enabled = 3,
    Disabled = 4,
    NotApplicable = 5
}

```

Listing 263: Klassen und eine Aufzählung zur Speicherung der wichtigsten Eigenschaften eines parallelen Ports

```
/* Klasse mit den wichtigsten Eigenschaften eines parallelen Ports */
public class ParallelPort
{
    public string DeviceId;
    public ParallelPortCapabilities Capabilities =
        new ParallelPortCapabilities();
    public bool DMA Support;
    public ParallelPortStatusInfo StatusInfo;
}
```

Listing 263: Klassen und eine Aufzählung zur Speicherung der wichtigsten Eigenschaften eines parallelen Ports (Forts.)

Um die ermittelten Ports auflisten zu können, implementiert die Klasse `ParallelPorts` eine typsichere Collection, die von der Klasse `CollectionBase` abgeleitet wird. Dabei werden nur die wichtigsten Methoden implementiert.

```
public class ParallelPorts: CollectionBase
{
    /* Implementierung des Indizierers */
    public ParallelPort this[int index]
    {
        get {return (ParallelPort)base.InnerList[index];}
        set {base.InnerList[index] = value;}
    }

    /* Implementierung der Add-Methode */
    public void Add(ParallelPort port)
    {
        base.InnerList.Add(port);
    }
}
```

Listing 264: Auflistung für `ParallelPort`-Instanzen

Die Methode `EnumParallelPorts` erzeugt für die WMI-Klasse `Win32_ParallelPort` eine Auflistung, geht diese durch und speichert die eingelesenen Daten in Instanzen der Klasse `ParallelPort`. Das Einlesen erfolgt in einer Ausnahmebehandlung um Fehler bei Eigenschaften zu vermeiden, die null speichern. Die Eigenschaft `StatusInfo`, die ja eine Aufzählung vom Typ `ParallelPortStatusInfo` ist, kann direkt konvertiert werden, da die Werte dieser Aufzählung mit den `StatusInfo`-Werten korrespondieren.

Das Auslesen der Fähigkeiten eines Ports ist etwas komplexer, da diese in einem `uint`-Array verwaltet werden. `EnumParallelPorts` geht dieses Array durch, vergleicht die gespeicherten Werte mit den dokumentierten Konstanten und belegt die entsprechende Eigenschaft der `Capabilities`-Eigenschaft des `ParallelPort`-Objekts mit `true`, wenn eine Fähigkeit gesetzt ist.

Die so definierten Ports werden dann der anfangs erzeugten `ParallelPorts`-Auflistung hinzugefügt, die am Ende der Methode zurückgegeben wird.

```

public static ParallelPorts EnumParallelPorts()
{
    // ParallelPorts-Instanz für die Rückgabe erzeugen
    ParallelPorts ports = new ParallelPorts();

    // WMI-Auflistung der Objekte der Win32_ParallelPort-Klasse erzeugen
    ManagementClass mc = new ManagementClass("Win32_ParallelPort");
    ManagementObjectCollection moc = mc.GetInstances();

    // Die einzelnen Objekte durchgehen
    foreach (ManagementBaseObject mbo in moc)
    {
        // ParallelPort-Instanz erzeugen und initialisieren
        ParallelPort port = new ParallelPort();
        try{port.DeviceId = (string)mbo["DeviceId"];}
        catch {}
        try{port.DMASupport = (bool)mbo["DMASupport"];}
        catch {}
        try
        {port.StatusInfo = (ParallelPortStatusInfo)mbo["StatusInfo"];}
        catch {}

        // Fähigkeiten aus dem Capabilities-Array auslesen
        try
        {
            uint[] caps = (uint[])mbo["Capabilities"];
            for (int i = 0; i < caps.Length; i++)
            {
                switch (caps[i])
                {
                    case 2: // Other
                        port.Capabilities.Other = true;
                        break;
                    case 3: // XT/AT Compatible
                        port.Capabilities.XTATCompatible = true;
                        break;
                    case 4: // PS/2 Compatible
                        port.Capabilities.PS2Compatible = true;
                        break;
                    case 5: // ECP
                        port.Capabilities.ECP = true;
                        break;
                    case 6: // EPP
                        port.Capabilities.EPP = true;
                        break;
                    case 7: // PC-98
                        port.Capabilities.PC98 = true;
                        break;
                    case 8: // PC-98-Hireso
                        port.Capabilities.PC98Hireso = true;
                        break;
                }
            }
        }
    }
}

```

Listing 265: Methode zum Auflisten der parallelen Ports des Systems

```

        case 9: // PC-H98
            port.Capabilities.PCH98 = true;
            break;
    }
}
}
catch {}

// ParallelPort-Instanz an die Auflistung anfügen
ports.Add(port);
}

// Speicher der WMI-Objekte freigeben um den Arbeitsspeicher möglichst
// schnell zu entlasten
moc.Dispose();
mc.Dispose();

// Ergebnis zurückgeben
return ports;
}

```

Listing 265: Methode zum Auflisten der parallelen Ports des Systems (Forts.)

Hinweis

Die Abfrage der parallelen Ports ergab auf meinem System (Windows XP Professional) für den Status, den (vorhandenen) DMA-Support und die Fähigkeiten immer den Wert null. Im Internet habe ich dazu leider nichts gefunden. Na ja, wenigstens die Eigenschaft `DeviceId` war immer vorhanden ...

Das folgende Listing zeigt eine Anwendung der Methode `EnumParallelPorts`:

```

/* Auflisten der parallelen Ports des Systems */
ParallelPorts ports = EnumParallelPorts();

foreach(ParallelPort port in ports)
{
    Console.WriteLine("DeviceId: {0}", port.DeviceId);
    Console.WriteLine("DMASupport: {0}", port.DMASupport);
    Console.WriteLine("StatusInfo: {0}", port.StatusInfo);
    Console.WriteLine("Capabilities:");
    Console.WriteLine("ECP: {0}", port.Capabilities.ECP);
    Console.WriteLine("EPP: {0}", port.Capabilities.EPP);
    Console.WriteLine("PC98: {0}", port.Capabilities.PC98);
    Console.WriteLine("PC98Hireso: {0}", port.Capabilities.PC98Hireso);
    Console.WriteLine("PCH98: {0}", port.Capabilities.PCH98);
    Console.WriteLine("PS2Compatible: {0}", port.Capabilities.PS2Compatible);
    Console.WriteLine("Other: {0}", port.Capabilities.Other);
    Console.WriteLine();
}

```

Listing 266: Beispielhafte Anwendung der Methode zum Auflisten der parallelen Ports des Systems



Abbildung 104: Das Beispielprogramm zur Ermittlung der wichtigsten Informationen zu den parallelen Schnittstellen des Systems

155 Informationen zu den seriellen Schnittstellen des Systems auslesen

Ähnlich dem Rezept 154 können Sie über die WMI-Klasse `Win32_SerialPort` (eingeschränkte) Informationen zu den seriellen Schnittstellen des lokalen Systems auslesen. Die Methode `EnumSerialPorts` Listing 267 arbeitet deswegen auf nahezu dieselbe Weise wie die Methode `EnumParallelPorts` aus dem genannten Rezept.

`EnumSerialPorts` gibt eine Auflistung des Typs `SerialPorts` mit Instanzen der Klasse `SerialPort` zurück. Die Klasse `SerialPort` enthält die wichtigsten Eigenschaften einer seriellen Schnittstelle.

`EnumSerialPorts` benötigt eine Referenz auf die Assembly `System.Management.dll` und den Import der Namensräume `System`, `System.Management` und `System.Collections`. Auf dem ausführenden Rechner muss WMI installiert sein.

```

/* Klasse für die Fähigkeiten eines seriellen Ports */
public class SerialPortCapabilities
{
    public bool    Unknown;
    public bool    CompatibleXTAT;
    public bool    Compatible16450;
    public bool    Compatible16550;
    public bool    Compatible16550A;
    public bool    Compatible8251;
    public bool    Compatible8251FIFO;
    public bool    Other;
}

/* Aufzählung für den Status eines seriellen Ports */
public enum SerialPortStatusInfo: ushort
{
    NotDefined = 0,
    Other = 1,
}

```

Listing 267: Methode zum Auflisten der seriellen Schnittstellen eines Systems


```

        Unknown = 2,
        Enabled = 3,
        Disabled = 4,
        NotApplicable = 5
    }

    /* Klasse mit den wichtigsten Eigenschaften eines seriellen Ports */
    public class SerialPort
    {
        public string DeviceId;
        public uint MaxBaudRate;
        public bool Supports16BitMode;
        public bool SupportsDTRDSR;
        public bool SupportsElapsedTimeouts;
        public bool SupportsIntTimeouts;
        public bool SupportsParityCheck;
        public bool SupportsRLSD;
        public bool SupportsRTSCTS;
        public bool SupportsSpecialCharacters;
        public bool SupportsXOnXOff;
        public bool SupportsXOnXOffSet;
        public SerialPortCapabilities Capabilities =
            new SerialPortCapabilities();
        public SerialPortStatusInfo StatusInfo;
    }

    /* Auflistung für SerialPort-Objekte */
    public class SerialPorts: CollectionBase
    {
        /* Implementierung des Indizierers */
        public SerialPort this[int index]
        {
            get {return (SerialPort)base.InnerList[index];}
            set {base.InnerList[index] = value;}
        }

        /* Implementierung der Add-Methode */
        public void Add(SerialPort port)
        {
            base.InnerList.Add(port);
        }
    }

    /* Methode zum Auflisten der seriellen Ports des Systems */
    public static SerialPorts EnumSerialPorts()
    {
        // SerialPorts-Instanz für die Rückgabe erzeugen
        SerialPorts ports = new SerialPorts();

        // WMI-Auflistung der Objekte der Win32_SerialPort erzeugen
        ManagementClass mc = new ManagementClass("Win32_SerialPort");
    }

```

Listing 267: Methode zum Auflisten der seriellen Schnittstellen eines Systems (Forts.)

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

ManagementObjectCollection moc = mc.GetInstances();

// Die einzelnen Objekte durchgehen
foreach (ManagementBaseObject mbo in moc)
{
    // SerialPort-Instanz erzeugen und initialisieren
    SerialPort port = new SerialPort();
    try{port.DeviceId = (string)mbo["DeviceId"];}
    catch {}
    try{port.MaxBaudRate = (uint)mbo["MaxBaudRate"];}
    catch {}
    try{port.Supports16BitMode = (bool)mbo["Supports16BitMode"];}
    catch {}
    try{port.SupportsDTRDSR = (bool)mbo["SupportsDTRDSR"];}
    catch {}
    try{port.SupportsElapsedTimeouts =
        (bool)mbo["SupportsElapsedTimeouts"];}
    catch {}
    try{port.SupportsIntTimeouts = (bool)mbo["SupportsIntTimeouts"];}
    catch {}
    try{port.SupportsParityCheck = (bool)mbo["SupportsParityCheck"];}
    catch {}
    try{port.SupportsRLSD = (bool)mbo["SupportsRLSD"];}
    catch {}
    try{port.SupportsRTSCTS = (bool)mbo["SupportsRTSCTS"];}
    catch {}
    try{port.SupportsSpecialCharacters =
        (bool)mbo["SupportsSpecialCharacters"];}
    catch {}
    try{port.SupportsXOnXOff = (bool)mbo["SupportsXOnXOff"];}
    catch {}
    try{port.SupportsXOnXOffSet = (bool)mbo["SupportsXOnXOffSet"];}
    catch {}
    try{port.StatusInfo = (SerialPortStatusInfo)mbo["StatusInfo"];}
    catch {}
    // Fähigkeiten aus dem Capabilities-Array auslesen
    try
    {
        uint[] caps = (uint[])mbo["Capabilities"];
        for (int i = 0; i < caps.Length; i++)
        {
            switch (caps[i])
            {
                case 1: // Other
                    port.Capabilities.Other = true;
                    break;
                case 2: // Unknown
                    port.Capabilities.Unknown = true;
                    break;
                case 3: // XT/AT Compatible
                    port.Capabilities.CompatibleXTAT = true;

```

Listing 267: Methode zum Auflisten der seriellen Schnittstellen eines Systems (Forts.)

```

        break;
    case 4:    // 16450 Compatible
        port.Capabilities.Compatible16450 = true;
        break;
    case 5:    // 16550 Compatible
        port.Capabilities.Compatible16550 = true;
        break;
    case 6:    // 16550A Compatible
        port.Capabilities.Compatible16550A = true;
        break;
    case 160:  // 8251 Compatible
        port.Capabilities.Compatible8251 = true;
        break;
    case 161:  // 8251FIFO Compatible
        port.Capabilities.Compatible8251FIFO = true;
        break;
    }
}
}
catch {}

// SerialPort-Instanz an die Auflistung anfügen
ports.Add(port);
}

// Speicher des WMI-Objekts freigeben um den Arbeitsspeicher möglichst
// schnell zu entlasten
moc.Dispose();
mc.Dispose();

// Ergebnis zurückgeben
return ports;
}

```

Listing 267: Methode zum Auflisten der seriellen Schnittstellen eines Systems (Forts.)

Hinweis

Ähnlich der Abfrage der parallelen Ports ergaben die Fähigkeiten (Capabilities) auf meinem System (Windows XP Professional) immer den Wert `null`. Anders als bei der Klasse `Win32_ParallelPort` war aber die Eigenschaft `StatusInfo` verfügbar.

156 Ermitteln der Soundkarten des Systems

Über die WMI-Klasse `Win32_SoundDevice` können Sie Informationen zu den installierten Soundkarten auslesen. Leider sind die meisten (von der Basisklasse `CIM_LogicalDevice` geerbten) Eigenschaften dieser Klasse scheinbar nicht implementiert und geben `null` zurück. Aber eigentlich interessiert uns ja nur, ob überhaupt eine Soundkarte installiert ist und wenn ja, welche.

Die Methode `EnumSoundcards` in Listing 268 liest deshalb nur die Geräte-Id und den Produkt-namen der installierten Soundkarten in eine Instanz der Klasse `Hashtable`. Ich beschreibe die Funktionsweise dieser Methode nicht näher, da sie im Prinzip nicht anders vorgeht als die Methoden in den vorhergehenden Rezepten.

Zum Kompilieren dieser Methode benötigen Sie eine Referenz auf die Assembly `System.Management.dll`. Außerdem müssen Sie die Namensräume `System`, `System.Management` und `System.Collections` importieren.

```
public static Hashtable EnumSoundcards()
{
    // Hashtable für die Rückgabe erzeugen
    Hashtable soundcards = new Hashtable();

    // Abfragen der installierten Soundkarten
    ManagementObjectSearcher searcher = new ManagementObjectSearcher(
        "SELECT * FROM Win32_SoundDevice");
    ManagementObjectCollection moc = searcher.Get();
    foreach (ManagementBaseObject mbo in moc)
    {
        // DeviceId (als Schlüssel) und ProductName (als Wert)
        // an die Hashtable-Instanz anfügen
        soundcards.Add(mbo["DeviceId"], mbo["ProductName"]);
    }

    // ManagementObjectSearcher freigeben, um den Speicher möglichst schnell zu
    // entlasten
    searcher.Dispose();

    // StringCollection zurückgeben
    return soundcards;
}
```

Listing 268: Methode zum Auflisten der wichtigsten Informationen zu den installierten Soundkarten

Wenn Sie nun lediglich ermitteln wollen, ob eine Soundkarte installiert ist, können Sie das folgende Programm einsetzen:

```
Hashtable soundcards = EnumSoundcards();
if (soundcards.Count == 0)
{
    MessageBox.Show("Auf Ihrem System wurde keine Soundkarte gefunden",
        Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Exclamation);
}
```

Listing 269: Einfache Abfrage danach, ob auf dem System eine Soundkarte installiert ist

157 Windows herunterfahren und neu starten

Über die Methoden `Reboot` und `Shutdown` einer Instanz der WMI-Klasse `Win32_OperatingSystem` können Sie ein System neu starten oder herunterfahren. Alternativ könnten Sie dazu auch die API-Funktion `ExitWindowsEx` einsetzen. Das dabei notwendige Erlangen des `Reboot`-Privilegs ist aber recht kompliziert.

Die Methode `ShutdownSystem` in Listing 270 besitzt die Argumente `hostName`, `remoteAuthUser` und `remoteAuthPassword`, um auch das Herunterfahren und Booten eines entfernten Systems zu ermöglichen. Mit diesen Informationen erzeugt die Methode zunächst ein `ManagementScope`-Objekt. Da das Herunterfahren und Neu-Starten des Systems das Privileg `SE_SHUTDOWN_NAME` (Herunterfahren des Systems) erfordert, setzt `ShutdownSystem` zusätzlich die Eigenschaft `EnablePrivileges` dieses Objekts auf `true`. Damit wird laut der Dokumentation erreicht, dass die Privilegien des aktuellen Benutzerkontos für WMI aktiviert werden und somit zur Verfügung stehen. Für das Herunterfahren und Neu-Booten eines entfernten Systems ist das Setzen dieser Eigenschaft allerdings nicht notwendig, wenn Sie das `ManagementScope`-Objekt mit Authentifizierungsinformationen versehen.

Das `ManagementScope`-Objekt wird bei der Erzeugung einer `ManagementClass`-Instanz übergeben, womit die dort eingestellte Konfiguration beim folgenden Erzeugen der `ManagementObjectCollection`-Auflistung für die verfügbaren Betriebssysteme übernommen wird. Über den Enumerator dieser Auflistung ermittelt `ShutdownSystem` dann eine Referenz auf ein `ManagementObject`-Objekt des aktiven Betriebssystems und ruft über die `Invoke`-Methode die `Reboot`- oder `Shutdown`-Methode des `Win32_OperatingSystem`-Objekts auf.

Die Methode `ShutdownSystem` benötigt eine Referenz auf die Assembly `System.Management.dll` und den Import der Namensräume `System` und `System.Management`. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
public static void ShutdownSystem(string hostName, string remoteAuthUser,
    string remoteAuthPassword, bool reboot)
{
    // Den Hostnamen auf localhost setzen, wenn keiner übergeben wurde
    if (hostName == null || hostName == "")
        hostName = "localhost";

    // ManagementScope-Objekt erzeugen, damit die EnablePrivileges-Eigenschaft
    // gesetzt und die Authentifizierungs-Informationen übergeben werden
    // können
    ManagementScope scope = new ManagementScope(@"\\" + hostName +
        "\\root\\cimv2");
    scope.Options.EnablePrivileges = true;
    scope.Options.Username = remoteAuthUser;
    scope.Options.Password = remoteAuthPassword;

    // Win32_OperatingSystem-Instanz für das aktive Betriebssystem ermitteln
    ManagementClass mc = new ManagementClass(scope, new ManagementPath(
```

Listing 270: Methode zum Herunterfahren oder Rebooten des Systems

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        "Win32_OperatingSystem"), new ObjectGetOptions());
ManagementObjectCollection moc = mc.GetInstances();
ManagementObjectCollection.ManagementObjectEnumerator
    mocEnumerator = moc.GetEnumerator();
mocEnumerator.MoveNext();
ManagementObject mo = (ManagementObject)mocEnumerator.Current;

// Rechner über die Win32Shutdown-Methode neu starten oder herunterfahren
if (reboot)
    mo.InvokeMethod("Reboot", null);
else
    mo.InvokeMethod("Shutdown", null);

// WMI-Objekte freigeben um den Speicher möglichst schnell zu entlasten
mo.Dispose();
moc.Dispose();
mc.Dispose();
}

```

Listing 270: Methode zum Herunterfahren oder Rebooten des Systems (Forts.)

Zur erfolgreichen Ausführung der Methode `ShutdownSystem` muss natürlich zunächst WMI auf dem Zielsystem installiert sein. Zudem muss das Benutzerkonto, das für die WMI-Authentifizierung verwendet wird, über das Privileg `SE_SHUTDOWN_NAME` verfügen. Dieses Privileg vergeben Sie über die lokalen Sicherheitseinstellungen (SYSTEMSTEUERUNG / VERWALTUNG / LOKALE SICHERHEITSRICHTLINIE / SICHERHEITSEINSTELLUNGEN / LOKALE RICHTLINIEN / ZUWEISEN VON BENUTZERRECHTEN), wo dieses Privileg als `HERUNTERFAHREN DES SYSTEMS` bezeichnet wird.

Hinweis

Leider funktionierte `ShutdownSystem` auf meinen Systemen (Windows XP Professional und Windows 2000 Professional) nicht für den lokalen Rechner. Obwohl das aktuelle Benutzerkonto auf meinem System über das Privileg `SE_SHUTDOWN_NAME` verfügte, erhielt ich beim Testen dieser Methode immer den Fehler »Recht wurde aufgehoben« beim Aufruf der `Reboot-` oder `Shutdown-Methode`. Da das Herunterfahren und Neu-Starten eines entfernten Systems problemlos möglich war und dieser Fehler auf dem lokalen System auch mit einem Quellcode auftrat, den ich einem Beispiel aus dem Internet entnommen hatte, gehe ich davon aus, dass die Methode `ShutdownSystem` prinzipiell fehlerfrei und das Problem irgendwo anders zu suchen ist. Unter Windows 98 (SE) traten übrigens keine Probleme auf: Der lokale Rechner wurde wie angefordert heruntergefahren bzw. neu gebootet.

158 Die verfügbaren Leistungsindikatoren auslesen

Windows stellt ab Windows NT mit Leistungsindikatoren (Performance Counter) eine Möglichkeit zur Verfügung, die Leistung von Teilen des Systems zu überwachen. Mit vordefinierten Leistungsindikatoren können Sie z.B. den aktuellen Speicherverbrauch oder die CPU-Auslastung überprüfen. Unter Windows 2000 und XP können Sie die verfügbaren Leis-

tungsindikatoren (u.a.) über den Systemmonitor (der in XP nur noch unter der Überschrift *Leistung* erscheint) überwachen. Diesen erreichen Sie über die Systemsteuerung (SYSTEMSTEUERUNG / VERWALTUNG / LEISTUNG bzw. SYSTEMSTEUERUNG / VERWALTUNG / SYSTEMMONITOR).

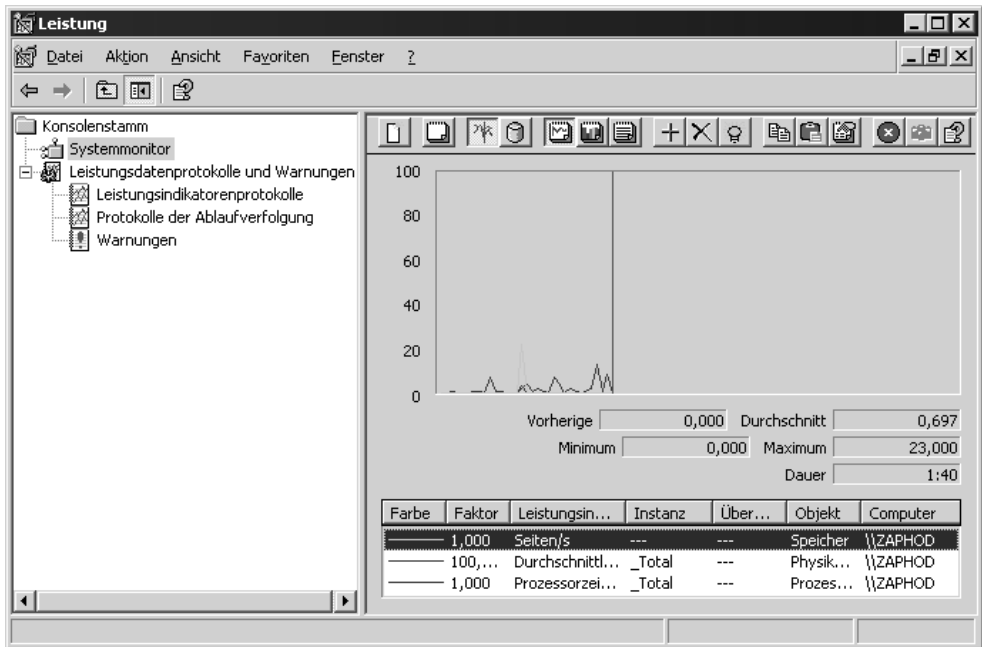


Abbildung 105: Der Systemmonitor von Windows XP

Über das Kontextmenü des Diagramms können Sie Leistungsindikatoren hinzufügen. Leistungsindikatoren sind in Kategorien eingeteilt. Die Kategorie *Speicher* enthält z.B. u.a. die Indikatoren *Verfügbare Bytes*, *Verfügbare KB*, *Verfügbare MB* und *Wechselfehler/s* (Abbildung 105).

Der Systemmonitor öffnet die angegebenen Leistungsindikatoren und stellt diese so ein, dass sie in einem angegebenen Zeitintervall aktualisiert werden. Geöffnete Leistungsindikatoren werden lediglich im Arbeitsspeicher verwaltet und von Programmen wie dem Systemmonitor abgefragt. Diese Abfrage von Leistungsindikatoren können Sie auch in Ihre Programme integrieren, wie ich es ab dem Rezept 159 zeige.

Zunächst geht es aber darum, die verfügbaren Leistungsindikatoren aufzulisten. Zum Zugriff auf diese stellt das .NET Framework im Namensraum `System.Diagnostics` die Klassen `PerformanceCounterCategory` und `PerformanceCounter` zur Verfügung. Eine `PerformanceCounterCategory`-Instanz repräsentiert eine Leistungsindikatorenkategorie, eine `PerformanceCounter`-Instanz einen einzelnen Leistungsindikator. Über diese Klassen können Sie eine Liste der auf einem System verfügbaren Leistungsindikatoren erzeugen.

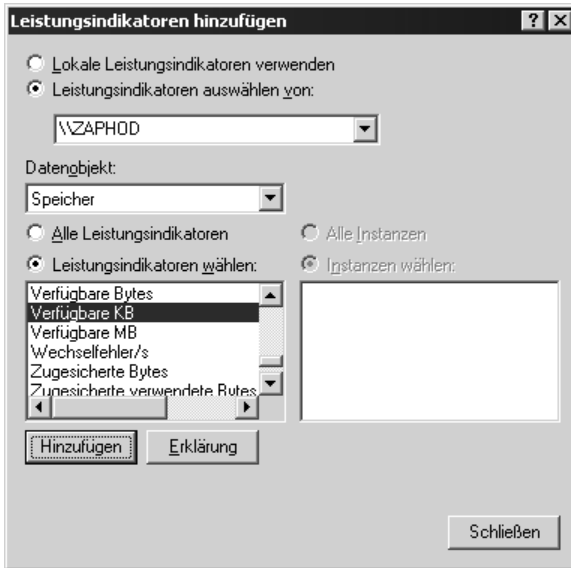


Abbildung 106: Hinzufügen einzelner Leistungsindikatoren

Hinweis

Beachten Sie, dass Leistungsindikatoren erst ab Windows NT zur Verfügung stehen. Unter Windows 98 und Me erhalten Sie bei der Verwendung der Klassen `PerformanceCounterCategory` und `PerformanceCounter` den Fehler »Dieses Feature erfordert Windows NT«.

Die Methode `WritePerformanceCounterNames` im folgenden Listing schreibt die verfügbaren Leistungsindikatoren in eine Textdatei, deren Dateiname übergeben wird. Sie erzeugt dazu zunächst eine `StreamWriter`-Instanz, ermittelt über die statische Methode `GetCategories` der `PerformanceCounterCategory`-Klasse alle verfügbaren Kategorien und geht diese durch. Nachdem der Name der Kategorie in den `StreamWriter` geschrieben wurde, liest `WritePerformanceCounterNames` über die Methode `GetInstanceNames` die Namen von eventuell vorhandenen »Instanzen« aus. Dazu sollten Sie wissen, dass Leistungsindikatoren ohne eine weitere Unterteilung in der Kategorie gespeichert sein oder in mehreren benannten Instanzen innerhalb der Kategorie vorkommen können. Verständlich wird dies durch den Vergleich zweier Kategorien. Die Kategorie *TCP* beinhaltet z.B. nur einen Satz an Leistungsindikatoren und somit keine Instanz. In der Kategorie *Prozess* werden aber einzelne Sätze der Leistungsindikatoren dieser Kategorie für jeden einzelnen Prozess und für die Gesamtheit aller Prozesse verwaltet. Diese Kategorie enthält also mehrere Instanzen der Leistungsindikatoren.

Dummerweise erzeugt die zum Auslesen der Leistungsindikatoren verwendete Methode `GetCounters` eines `PerformanceCounterCategory`-Objekts eine Ausnahme, wenn Sie diese Methode ohne Argument aufrufen und die Kategorie Instanzen beinhaltet. Für diesen Fall müssen Sie dieser Methode den Namen der Instanz übergeben und können dann die einzelnen Leistungsindikatoren dieser Instanz auslesen.

Zum Kompilieren der Methode müssen Sie die Namensräume `System`, `System.Diagnostics`, `System.Text` und `System.IO` importieren.

Da `WritePerformanceCounterNames` lediglich die Namen der Leistungsindikatoren einer Kategorie auflisten soll, wird überprüft, ob Instanzen bestehen. Ist dies der Fall, geht `WritePerformanceCounterNames` die Namen der Instanzen durch und schreibt diese in die Datei.

```
public static void WritePerformanceCounterNames(string fileName)
{
    // Datei zum Schreiben öffnen
    StreamWriter sr = new StreamWriter(fileName, false, Encoding.Default);

    // Ermitteln und Durchgehen der Leistungsindikatoren-Kategorien
    PerformanceCounterCategory[] pccList =
        PerformanceCounterCategory.GetCategories();
    foreach (PerformanceCounterCategory pcc in pccList)
    {
        sr.WriteLine("Kategorie: {0}", pcc.CategoryName);

        try
        {
            // Ermitteln der Namen der eventuell vorhandenen Instanzen
            string[] pcInstanceNames = pcc.GetInstanceNames();

            PerformanceCounter[] pcList = null;
            if (pcInstanceNames.Length > 0)
            {
                // Es existieren Instanzen der Kategorie: Zunächst die Namen der
                // Instanzen ausgeben
                sr.WriteLine("Instanzen von {0}:", pcc.CategoryName);
                for (int i = 0; i < pcInstanceNames.Length; i++)
                    sr.WriteLine("    " + pcInstanceNames[i]);
            }
        }
    }
}
```

Listing 271: Abfragen der verfügbaren Leistungsindikatoren – Teil 1

Für den Fall, dass Instanzen vorhanden sind, sollen die Indikatoren einer der Instanzen ausgelesen werden. Dabei entstand beim Testen aber das Problem, dass beim Aufruf der `GetCounters`-Methode für bestimmte Instanzen eine Ausnahme mit der Meldung erzeugt wurde, dass diese nicht vorhanden wäre (obwohl der Name ja von `GetInstanceNames` zurückgegeben wurde). Deshalb habe ich das Auslesen der Leistungsindikatoren in eine Schleife integriert, die alle Instanznamen durchgeht und für jede Instanz versucht, deren Indikatoren auszulesen. Für den Fall, dass alle Instanznamen zu Ausnahmen führen, speichert `WritePerformanceCounterNames` den zuletzt aufgetretenen Fehler in einer Variable, gibt diesen in die Datei aus und ruft `continue` auf, damit die äußere Schleife mit der nächsten Kategorie weitermacht.

```

// Abfragen der Leistungsindikatoren einer der Instanzen
bool ok = false;
string lastError = null;
foreach(string instanceName in pcInstanceNames)
{
    try
    {
        pcList =
            pcc.GetCounters(instanceName);
        ok = true;
        break;
    }
    catch (Exception ex)
    {
        lastError = ex.Message;
    }
}
if (ok == false)
{
    // Fehler ausgeben
    sr.WriteLine("Kann Kategorie {0} nicht abfragen, da " +
        "beim Abfragen der Instanzen Fehler aufgetreten sind. " +
        "Letzter Fehler: {1}", pcc.CategoryName, lastError);

    // Mit der nächsten Kategorie weitermachen
    sr.WriteLine();
    continue;
}
}

```

Listing 272: Abfragen der verfügbaren Leistungsindikatoren – Teil 2

Sind keine Instanzen vorhanden, ruft `WritePerformanceCounterNames` die `GetCounters`-Methode ohne Argument auf:

```

else
{
    pcList = pcc.GetCounters();
}

```

Listing 273: Abfragen der verfügbaren Leistungsindikatoren – Teil 3

Schließlich geht `WritePerformanceCounterNames` die ermittelten Leistungsindikatoren nur noch durch und schreibt deren Name, Beschreibung und den Typ, der ein Wert der Aufzählung `PerformanceCounterType` ist (siehe Tabelle 11), in die Datei:

```

        sr.WriteLine("Leistungsindikatoren von {0}:", pcc.CategoryName);
        foreach (PerformanceCounter pc in pcList)
        {
            sr.WriteLine("  {0}: {1} ({2})", pc.CounterName, pc.CounterHelp,
                pc.CounterType);
        }
    }
    catch (Exception ex)
    {
        sr.WriteLine("Kann Kategorie nicht abfragen: {0}", ex.Message);
    }

    sr.WriteLine();
}

// StreamWriter schließen
sr.Close();
}

```

Listing 274: Abfragen der verfügbaren Leistungsindikatoren – Teil 4

Die äußere Ausnahmebehandlung habe ich übrigens deswegen integriert, da einige Leistungsindikatorenkategorien mit Instanzen scheinbar keine Daten speichern und bei der Abfrage zu der Ausnahme »Es können keine genauen Daten zurückgegeben werden, da die <Kategorienname>-Kategorie keine Instanzeninformationen enthält« führen.

Abbildung 107 zeigt das Ergebnis einer Abfrage der Leistungsindikatoren meines Systems in einem Editor.

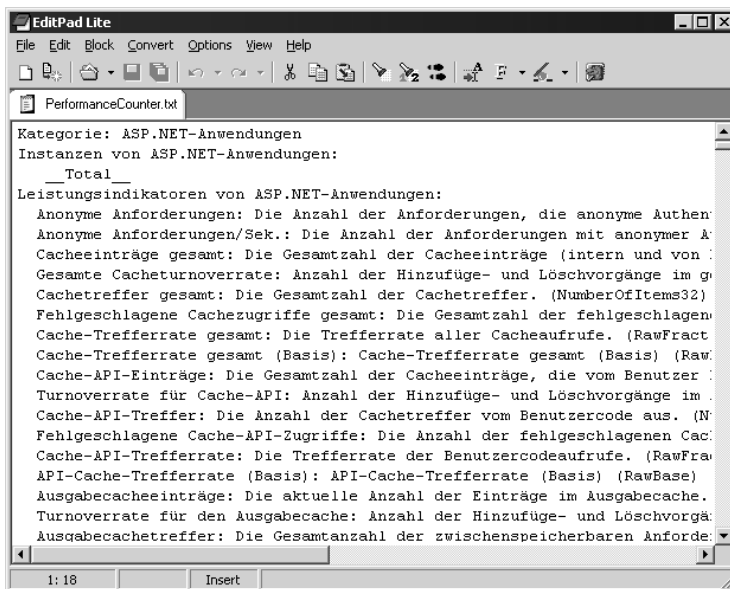


Abbildung 107: Das Ergebnis der WritePerformanceCounterNames-Methode in einem Texteditor

Der Typ eines Leistungsindikators sagt übrigens aus, welche Art von Informationen der Leistungsindikator speichert (siehe Tabelle 11).

Leistungsindikatortyp	Beschreibung
AverageCount64	Ein AverageCount64-Indikator gibt die durchschnittliche Anzahl von Aktionen oder Elementen innerhalb eines Messintervalls an. Ein Messintervall reicht vom letzten bis zum aktuellen Lesen des Wertes. Innerhalb dieses Intervalls kann der Indikatorwert auch mehrfach aktualisiert werden. Ein AverageCount64-Indikator berechnet den Durchschnitt aller Aktualisierungen im Messintervall. Ein Beispiel für einen solchen Indikator ist der Indikator <i>Mittlere Bytes/Übertragung</i> in der Kategorie <i>Datenträger</i> .
AverageTimer32	Dieser Typ beschreibt einen Leistungsindikator, der die für die Ausführung aller in einem Messintervall ausgeführten bestimmten Vorgänge benötigte durchschnittliche Zeit verwaltet. Der Indikator wird mit einer Zeitangabe in Form von Systemticks (Basisfrequenz des Motherboards) aktualisiert. Bei einer Messung (also beim Lesen des Indikatorwerts) liefert er den Durchschnitt der seit der letzten Messung geschriebenen Werte in Sekunden.
ElapsedTime	In einem Leistungsindikator dieses Typs wird eine seit einem Startzeitpunkt verstrichene Zeit in Sekunden gespeichert.
NumberOfItems32 NumberOfItems64	In einem solchen Leistungsindikator werden einfach nur Elemente oder Vorgänge gezählt. Ein Beispiel dafür ist der Indikator <i>Aufträge in Warteschlange</i> der Kategorie <i>Druckerwarteschlange</i> . <i>NumberOfItems32</i> definiert dabei einen 32-Bit-Integerwert, <i>NumberOfItems64</i> einen 64-Bit-Integerwert.
RateOfCountsPerSecond32 RateOfCountsPerSecond64	Ein Leistungsindikator dieses Typs speichert die Anzahl von Vorgängen pro Sekunde als 32- oder 64-Bit-Integerzahl.
Timer100Ns	Ein Leistungsindikator dieses Typs speichert einen prozentualen Anteil der Aktivierungsdauer einer Komponente im Vergleich zu der Gesamtdauer eines Messintervalls. Die Zeit wird dabei in Einheiten von 100 Nanosekunden gemessen. Der Indikator <i>Prozessorzeit (%)</i> der <i>Prozess</i> -Kategorie ist ein Beispiel dafür.

Tabelle 11: Die wichtigen Typen der Leistungsindikatoren

Die Leistungsindikatoren, die einen Durchschnitt oder eine Differenz berechnen, sind vielleicht etwas schwer zu verstehen (bei meinen ersten Versuchen dachte ich jedenfalls, diese berechnen den gesamten Durchschnitt seit dem ersten Setzen eines Werts). Diese Indikatoren berechnen den Durchschnitt bzw. die Differenz der Indikator-Rohwerte immer zwischen

zwei Messungen (also zwischen dem letzten und dem aktuellen Lesen des Indikatorwerts). Wenn Sie einen `AverageCount64`-Indikator z.B. einmal pro Sekunde lesen, und dieser wird vom System innerhalb dieser Sekunde mit den Werten 1, 5 und 6 aktualisiert, ergibt das Lesen des Indikators den Wert 4 ($(1 + 5 + 6) / 3$). Nach dem Lesen wird der Indikator-Rohwert immer auf 0 gesetzt, weshalb beim nächsten Lesen wieder der Durchschnitt des dann aktuellen Messintervalls angezeigt wird.

159 Den Arbeitsspeicher überwachen

Die Überwachung des Arbeitsspeichers kann in einigen Situationen, bei denen eine (speicherintensive) Anwendung Probleme (z.B. Abstürze) verursacht, die nicht genauer lokalisiert werden können, ein wichtiges Mittel zur Problemlösung darstellen. Eine Speicherüberwachung können Sie natürlich über den Windows-Systemmonitor vornehmen. Viele Anwender (bei Ihren Kunden z.B.) sind damit aber überfordert. Deshalb ist es häufig sinnvoll, die Speicherüberwachung in das eigene Programm zu integrieren.

Die Standard-Windows-Leistungsindikatorenkategorie (siehe Rezept 158) *Speicher (Memory)* enthält einige Leistungsindikatoren zur Überwachung des Arbeitsspeichers. Die wichtigsten zeigt Tabelle 12.

Leistungsindikator	Beschreibung	Typ
Auslagerungsseiten: Residente Bytes (Pool Paged Resident Bytes)	die aktuelle Größe des Auslagerungsspeichers in Bytes	<code>NumberOfItems64</code>
Verfügbare Bytes (Available Bytes)	für Prozesse verfügbarer physikalischer Speicher in Bytes	<code>NumberOfItems64</code>
Zugesicherte Bytes (Committed Bytes)	Zugesicherter virtueller Speicher in Bytes. Zugesicherter Speicher ist physikalischer Speicher, für den in Auslagerungsdateien Speicherplatz reserviert wurde.	<code>NumberOfItems64</code>
Zusagegrenze (Commit Limit)	virtueller Speicher in Bytes, der zugesichert werden kann, ohne die Auslagerungsdatei(en) zu vergrößern	<code>NumberOfItems64</code>

Tabelle 12: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie

Zur Überwachung des Arbeitsspeichers können Sie ein `PerformanceCounter`-Objekt verwenden, in dessen Konstruktor Sie den Namen der Leistungsindikatorenkategorie und den Namen des Leistungsindikators übergeben. Im optionalen dritten Argument können Sie mit `true` bestimmen, dass es auch erlaubt ist, in den Leistungsindikator zu schreiben, was allerdings nur bei benutzerdefinierten Leistungsindikatoren (siehe Rezept 162) Sinn macht.

Die Namen der Leistungsindikatorenkategorien und der Leistungsindikatoren können in der auf dem System aktuellen Sprache angegeben werden. Ein auf einem solchen System entwickeltes Programm läuft dann aber nicht mehr auf einem Computer mit einer anderen eingestellten Systemsprache. Laut meinen Versuchen und einem News-group-Beitrag können Sie die Namen aber scheinbar immer auch in der englischen Sprache angeben. Damit sollte das Programm dann auf allen Systemen ausgeführt werden können. Um die englischen Texte zu erhalten, können Sie das Programm aus dem vorhergehenden Rezept vor dem Lesen der verfügbaren Indikatoren auf die englische Sprache umschalten:

```
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en");
```

Um den aktuellen Wert des Indikators abzufragen, rufen Sie die Methode `NextValue` auf, die einen `float`-Wert zurückgibt. Diesen Aufruf können Sie manuell (z.B. über einen Schalter) ausführen lassen oder in eine Timer-Methode oder einen Thread integrieren, um den Speicher regelmäßig und andauernd zu überwachen.

Listing 275 zeigt die Verwendung eines Threads zur Überwachung des Arbeitsspeichers in einer Konsolenanwendung. Das Programm startet in `Main` einen Thread mit der Methode `GetRAMUsage`. Diese Methode erzeugt zunächst ein `PerformanceCounter`-Objekt, liest in einer (Thread-typischen) Endlosschleife den aktuellen Wert des Leistungsindikators aus und gibt diesen an der Konsole aus. Beim Unterschreiten eines Minimalwerts (1 MB) wird zusätzlich eine Alarmmeldung ausgegeben. `GetRAMUsage` hält den Thread dann über `Thread.Sleep` für eine Sekunde an, um eine sekundliche Überprüfung des Arbeitsspeichers zu erreichen.

Die `Main`-Methode wartet, während der Thread läuft, so lange, bis der Anwender die Return-Taste betätigt, und beendet den Thread schließlich.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.Threading` und `System.Diagnostics` einbinden. Das Programm kann nicht unter Windows 98 und Me ausgeführt werden, da Leistungsindikatoren auf diesen Systemen nicht zur Verfügung stehen.

```
/* Thread-Methode für die Überwachung des Arbeitsspeichers */
public static void GetRAMUsage()
{
    // PerformanceCounter-Instanz für den Leistungsindikator
    // 'Verfügbare Bytes' der Kategorie 'Speicher' erzeugen (in englischer
    // Sprache um das Programm auch auf Systemen ausführen zu können, die eine
    // andere Sprache als Deutsch verwenden)
    PerformanceCounter pc = new PerformanceCounter("Memory",
        "Available Bytes");
    do
    {
        // den nächsten Wert auslesen und ausgeben
        float byteCount = pc.NextValue();
```

Listing 275: Überwachen der freien Bytes im physikalischen Arbeitsspeicher über einen Thread

```

        Console.WriteLine("{0:0} Bytes verfügbar", byteCount);

        // Alarmmeldung ausgeben, wenn der Minimalwert unterschritten wird
        if (byteCount < 1048576)
        {
            Console.WriteLine("ALARM: Der verfügbare Speicher " +
                               "ist unter 1 MB gesunken");
        }

        // Thread eine Sekunde anhalten
        Thread.Sleep(1000);
    } while (true);
}

/* Main-Methode */
[STAThread]
static void Main(string[] args)
{
    Console.WriteLine("Beenden mit Return");

    // Thread starten
    Thread ramUsageThread = new Thread(new ThreadStart(GetRAMUsage));
    ramUsageThread.IsBackground = true;
    ramUsageThread.Start();

    // Auf Return warten
    Console.ReadLine();

    // Thread beenden
    ramUsageThread.Abort();
}

```

Listing 275: Überwachen der freien Bytes im physikalischen Arbeitsspeicher über einen Thread (Forts.)

Abbildung 108 zeigt das Programm in Aktion.



Abbildung 108: Das Programm zur Überwachung der freien Bytes im physikalischen Arbeitsspeicher in Aktion

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

160 Speicherauslastung und CPU-Belastung des aktuellen Prozesses überwachen

Bei speicher- oder prozessorintensiven Programmen ist es häufig zu Test- oder Debugging-zwecken sinnvoll, den vom Programm belegten Arbeitsspeicher oder die Prozessorauslastung in spezifischen Situationen (z.B. beim Aufruf einer rechenintensiven Methode) zu überwachen. Dazu können Sie die Leistungsindikatoren der Leistungsindikatorenkategorie *Prozess* verwenden, die ein standardmäßiger Bestandteil von Windows ist. Grundlegende Informationen zu Leistungsindikatoren finden Sie im Rezept 158. Tabelle 13 stellt die wichtigsten Leistungsindikatoren der Kategorie *Prozess* (*Process*) dar.

Leistungsindikator	Beschreibung	Typ
Arbeitsseiten (Working Set)	Die aktuelle Speicherbelegung für die Arbeitsseiten des Prozesses in Bytes. Aus Gründen der Performance wird Arbeitsspeicher immer auf so genannten Seiten verwaltet, die eine bestimmte Größe besitzen. Das Auslagern von zurzeit unbenutztem Speicher in die Auslagerungsdatei und das Wieder-Zurücklesen ist wesentlich effizienter, wenn an Stelle von einzelnen Bytes ganze Seiten geschrieben und gelesen werden können. Die Arbeitsseiten eines Prozessors beinhalten den kompletten vom Prozess belegten (virtuellen) Speicher (das ist der private und der mit anderen Prozessen geteilte Speicher). Die einem Prozess zugeordneten Arbeitsseiten können von Windows je nach dem freien Platz im virtuellen Speicher auch mehr Speicher beinhalten, als der Prozess benötigt.	NumberOfItems64
Auslagerungsdatei (Bytes) (Page File Bytes)	die aktuelle Anzahl der Bytes, die ein Prozess in der Auslagerungsdatei verwendet	NumberOfItems64
Private Bytes (Private Bytes)	die aktuelle Größe des Speichers, der vom Prozess reserviert wurde und nicht mit anderen Prozessen geteilt werden kann	NumberOfItems64
Prozessorzeit (%) (% Processor Time)	der Prozentanteil der verstrichenen Zeit, in der alle Threads eines Prozesses den Prozessor verwendet haben	Timer100Ns
Threadanzahl (Thread Count)	die Anzahl der in diesem Prozess aktiven Threads	NumberOfItems32
Verstrichene Zeit (Elapsed Time)	die gesamte Zeitdauer, in Sekunden, in der dieser Prozess ausgeführt wurde	ElapsedTime

Tabelle 13: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie

Leistungsindikator	Beschreibung	Typ
Virtuelle Größe (Virtual Bytes)	die aktuelle Größe des virtuellen Adressraums in Bytes, den der Prozess verwendet	NumberOfItems64

Tabelle 13: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie (Forts.)

Die für uns aussagekräftigsten Indikatoren sind *Prozessorzeit (%)* und *Arbeitsseiten*. Dabei handelt es sich um die Werte, die auch im Windows-Taskmanager für jeden Prozess angezeigt werden. Der Indikator *Arbeitsseiten* steht für die Gesamt-Speicherausnutzung des Prozesses.

Das Programm in Listing 276 liest diese beiden Indikatoren und zusätzlich die Anzahl der aktiven Thread aus, um eine Aussage über die Systembelastung des aktuellen Prozesses machen zu können. Dazu werden zunächst drei `PerformanceCounter`-Instanzen für die Leistungsindikatoren erzeugt. Die Namen der Kategorie und der Indikatoren werden in englischer Sprache übergeben, um zu erreichen, dass das Programm auch auf Systemen ausgeführt werden kann, die eine andere Sprache als Deutsch verwenden.

Die Kategorie *Prozess* enthält immer mehrere Instanzen, nämlich eine Instanz pro ausgeführtem Prozess, die den Namen des Prozesses trägt, und eine Instanz mit dem Namen *_Total*, die Summen-Informationen für alle Prozesse liefert. Da das Programm die Daten des aktuellen Prozesses abfragen soll, liest es den Namen des aktuellen Prozesses aus der Eigenschaft `ProcessName` der `Process`-Instanz aus, die die Methode `Process.GetCurrentProcess` zurückgibt, und übergibt diesen Namen an die Eigenschaft `InstanceName` der erzeugten `PerformanceCounter`-Instanzen. Schließlich fragt das Programm den aktuellen Wert jeder dieser Instanzen ab und gibt diesen aus.

Zum Kompilieren des Programms müssen Sie die Namensräume `System` und `System.Diagnostics` importieren. Die Ausführung ist unter Windows 98 und Me nicht möglich, da Leistungsindikatoren erst ab Windows NT zur Verfügung stehen.

```
// PerformanceCounter-Instanzen für die zu überwachenden
// Leistungsindikatoren der Kategorie 'Prozess' erzeugen
PerformanceCounter pc1 = new PerformanceCounter("Process",
"% Processor Time");
PerformanceCounter pc2 = new PerformanceCounter("Process", "Working Set");
PerformanceCounter pc3 = new PerformanceCounter("Process", "Thread Count");

// Instanzname auf den Namen des aktuellen Prozesses festlegen
pc1.InstanceName = Process.GetCurrentProcess().ProcessName;
pc2.InstanceName = Process.GetCurrentProcess().ProcessName;
pc3.InstanceName = Process.GetCurrentProcess().ProcessName;

// Die jeweils nächsten Werte auslesen und ausgeben
Console.WriteLine("Prozessorzeit: {0:0} %", pc1.NextValue());
Console.WriteLine("Speicherauslastung: {0:0} KB", pc2.NextValue() / 1024);
Console.WriteLine("Threadanzahl: {0:0}", pc3.NextValue());
```

Listing 276: Anzeigen der vom aktuellen Prozess verwendeten Prozessorzeit, der Speicherauslastung und der Anzahl der Threads



Abbildung 109: Das Beispielprogramm in Aktion

In einer realen Anwendung ist es wohl in der Regel sinnvoll, die Abfrage der Speicherauslastung vor, während und nach dem Aufruf von speicherintensiven Methoden auszuführen. Die Ermittlung der Prozessorauslastung würde hier aber keinen Sinn machen, da der aktuelle Thread beim Aufruf der `NextValue`-Methode nur diese ausführt und die damit ermittelte Prozessorauslastung keine Aussagekraft besitzt. Die Prozessorauslastung sollten Sie immer in einem separaten Thread ermitteln. Dort können Sie die ermittelten Daten z.B. in ein (Trace-) Protokoll (siehe nächstes Rezept) schreiben.

Vor und nach dem Aufruf prozessorintensiver Methoden können Sie diesen Aufruf ebenfalls im Protokoll dokumentieren und erhalten somit eine Zuordnung der Methodenaufrufe zu der CPU-Belastung. Im folgenden Rezept finden Sie eine Klasse, die diese Arbeit erleichtert.

161 Speicherauslastung und CPU-Belastung des aktuellen Prozesses in ein Trace-Protokoll schreiben

Das vorhergehende Rezept beschreibt, wie Sie die interessantesten Leistungsindikatoren der Leistungsindikatorenkategorie *Prozess* abfragen um Performance-Informationen für den Prozess einer Anwendung zu erhalten. Diese Informationen werden in der Praxis häufig in ein Trace-Protokoll (»Spurverfolgungs-Protokoll«) geschrieben. Dieses Protokoll kann dann auf verschiedene Weisen ausgegeben (z.B. in eine Datei geschrieben) werden. Ein Beispiel dafür finden Sie in Listing 283, bei der Anwendung der im Folgenden beschriebenen Klasse.

Um das Tracen der Speicher- und der Prozessorauslastung zu erleichtern, habe ich die Klasse `PerformanceTracer` entwickelt. Über die Methode `TraceRamUsage` können Sie den aktuellen Speicherverbrauch der Anwendung in das Trace-Protokoll schreiben. Die Methode `StartCPUUsageTraceThread` startet einen Thread, der in dem in Millisekunden übergebenen Intervall die CPU-Belastung in das Trace-Protokoll schreibt. Die Methode `StopCPUUsageTraceThread` beendet diesen Thread.

Um die Klasse kompilieren zu können, müssen Sie die Namensräume `System`, `System.Threading` und `System.Diagnostics` einbinden. Das Programm kann nicht unter Windows 98 und Me ausgeführt werden, da Leistungsindikatoren auf diesen Systemen nicht zur Verfügung stehen.

Die Klasse enthält zunächst private Eigenschaften für die `ProcessCounter`-Instanzen, für die Übergabe des Abfrageintervalls an den CPU-Auslastungs-Thread und für den Thread selbst:

```
public class PerformanceTracer
{
    /* Private Eigenschaften für die PerformanceCounter */
    private PerformanceCounter ramCounter;
    private PerformanceCounter cpuCounter;

    /* Private Eigenschaft zur Übergabe des Abfrageintervalls der CPU-Auslastung
     * an den Thread */
    private int cpuUsageReadInterval;

    /* Private Eigenschaft für den Thread */
    private Thread cpuUsageThread = null;
```

Listing 277: Klasse Methoden zum Tracen der Speicher- und der CPU-Auslastung – Teil 1

Im Konstruktor der Klasse werden die `PerformanceCounter`-Instanzen für die Leistungsindikatoren *Working Set* und *% Processor Time* der *Process*-Kategorie erzeugt und mit dem Namen des aktuellen Prozesses initialisiert. Ich verwende hier die englischen Namen um zu erreichen, dass das Programm auch auf fremdsprachigen Systemen ausgeführt werden kann:

```
public PerformanceTracer()
{
    this.ramCounter = new PerformanceCounter("Process", "Working Set");
    this.ramCounter.InstanceName = Process.GetCurrentProcess().ProcessName;
    this.cpuCounter = new PerformanceCounter("Process", "% Processor Time");
    this.cpuCounter.InstanceName = Process.GetCurrentProcess().ProcessName;
}
```

Listing 278: Der Konstruktor der Klasse zum Tracen der Speicher- und der CPU-Auslastung

Die Methode `TraceRamUsage` ermittelt den aktuellen Speicherverbrauch über die `PerformanceCounter`-Instanz `ramCounter` und schreibt diesen mit der übergebenen Info in das `Trace`-Protokoll:

```
public void TraceRamUsage(string info)
{
    Trace.WriteLine(String.Format("Speicherauslastung bei {0}: " +
        "{1:0.00} KB ", info, this.ramCounter.NextValue() / 1024));
}
```

Listing 279: Methode zum Tracen der Speicherauslastung

Da die Überprüfung der aktuellen CPU-Auslastung in einem Thread erfolgen soll, enthält die Klasse die Methode `TraceCPUUsage`, die diese Arbeit erledigt. Das Auslesen der CPU-Auslastung erfolgt in einer Endlosschleife (die durch das Beenden des Threads abgebrochen werden kann). Über das `lock`-Schlüsselwort wird zunächst der Zugriff auf die `Trace`-Klasse für andere Threads gesperrt, damit keine Probleme bei einem gleichzeitigen Zugriff entstehen. Danach liest `TraceCPUUsage` über das private `PerformanceCounter`-Objekt `cpuCounter` die aktu-

Basics

Datum
ZeitAnwen-
dungenDatei-
systeme

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

elle Prozessorauslastung aus und schreibt diese in das Trace-Protokoll. Da das Lesen in Intervallen erfolgen soll, hält die Methode den Thread dann noch für das angegebene Intervall an:

```
private void TraceCPUUsage()
{
    do
    {
        lock(typeof(Trace))
        {
            Trace.WriteLine("CPU-Auslastung " +
                DateTime.Now.ToLongTimeString() + ": " +
                this.cpuCounter.NextValue() + "%");
        }

        Thread.Sleep(this.cpuUsageReadInterval);
    } while (true);
}
```

Listing 280: Methode für den Thread zum Tracen der CPU-Auslastung

Die Methode `StartCPUUsageTraceThread` schreibt zunächst das übergebene Intervall in die private Eigenschaft `cpuUsageReadInterval`, erzeugt dann einen neuen Thread für die `TraceCPUUsage`-Methode und startet diesen:

```
public void StartCPUUsageTraceThread(int readInterval)
{
    this.cpuUsageReadInterval = readInterval;

    this.cpuUsageThread = new Thread(new ThreadStart(this.TraceCPUUsage));
    cpuUsageThread.Start();
}
```

Listing 281: Methode zum Starten des Thread, der die CPU-Auslastung verfolgt

Die Methode `StopCPUUsageTraceThread` ruft die `Abort`-Methode des Thread auf, um diesen zu stoppen:

```
public void StopCPUUsageTraceThread()
{
    this.cpuUsageThread.Abort();
}
}
```

Listing 282: Methoden zum Stoppen des Thread, der die CPU-Auslastung verfolgt

Listing 283 zeigt eine beispielhafte Anwendung dieser Klasse in einer Konsolenanwendung. Die Methode `Stress` erzeugt in einer Schleife mit 100.000 Durchläufen einzelne `DateTime`-Objekte und gibt den Schleifenzähler an der Konsole aus. Die Schleife produziert eine hohe

CPU-Auslastung, das Instanzieren der `DateTime`-Objekte führt zu einer erhöhten Speicherauslastung, da der Garbage Collector diese frühestens nach der Ausführung der Schleife freigibt. Beim Ein- und Austritt schreibt `Stress` den aktuellen Speicherverbrauch über die `TraceRamUsage`-Methode in das Trace-Protokoll.

Die `Main`-Methode initialisiert zunächst das Trace-Protokoll mit einem standardmäßigen Trace-Listener, der die erhaltenen Daten in eine Textdatei schreibt. Dann erzeugt `Main` eine Instanz der `PerformanceTracer`-Klasse (die in einer privaten statischen Variablen verwaltet wird). Über diese Instanz startet `Main` dann den Thread zur Messung der CPU-Auslastung und ruft die `Stress`-Methode auf. Nach der Rückkehr dieser Methode beendet `Main` den Thread und schließt das Trace-Protokoll, um die Protokolldatei zu schreiben.

```
using System;
using System.IO;
using System.Diagnostics;
using Addison_Wesley.Codebook.System;

namespace Prozess_Performance_Tracing
{
    class Start
    {
        // Eigenschaft für den Performance-Tracer
        private static PerformanceTracer performanceTracer;

        /* Methode, die eine hohe CPU-Auslastung erzeugt */
        public static void Stress()
        {
            // Protokollieren des Eintritts in die Methode
            performanceTracer.TraceRamUsage("Eintritt in Stress");

            // Prozessor- und speicherintensive Schleife
            for (int i = 0; i < 100000; i++)
            {
                // DateTime-Objekt zur Simulation von Speicherauslastung erzeugen
                DateTime d = new DateTime(0);

                Console.WriteLine("{0} ", i);
            }

            // Protokollieren des Austritts aus der Methode
            performanceTracer.TraceRamUsage("Austritt in Stress");
        }

        [STAThread]
        static void Main(string[] args)
        {
            // Protokolldatei löschen, falls diese vorhanden ist
            string logfileName = "c:\\\\trace.log";
```

Listing 283: Beispielhafte Anwendung der Klasse zum Tracen der Speicher- und der CPU-Auslastung in einer Konsolenanwendung

```

    if (File.Exists(logfileName))
        File.Delete(logfileName);

    // Trace-Listener zum Schreiben in eine Datei erzeugen und
    // an die Listeners-Auflistung der Trace-Klasse anfügen
    Trace.Listeners.Clear();
    Trace.Listeners.Add(new System.Diagnostics.TextWriterTraceListener(
        logfileName));

    // PerformanceTracer-Instanz erzeugen
    performanceTracer = new PerformanceTracer();

    // CPU-Auslastungs-Thread mit einem 100ms-Intervall starten
    performanceTracer.StartCPUUsageTraceThread(100);

    // Speicher- und prozessorintensive Methode aufrufen
    Stress();

    // CPU-Auslastungs-Thread stoppen
    performanceTracer.StopCPUUsageTraceThread();

    // Trace-Listener schließen
    Trace.Listeners[0].Close();
}
}
}

```

Listing 283: Beispielhafte Anwendung der Klasse zum Tracen der Speicher- und der CPU-Auslastung in einer Konsolenanwendung (Forts.)

Abbildung 110 zeigt eine durch das Beispielprogramm erzeugte Protokolldatei in einem Editor. Die Datei habe ich ein wenig manipuliert, sodass der Anfang und das Ende des Protokolls sichtbar sind.

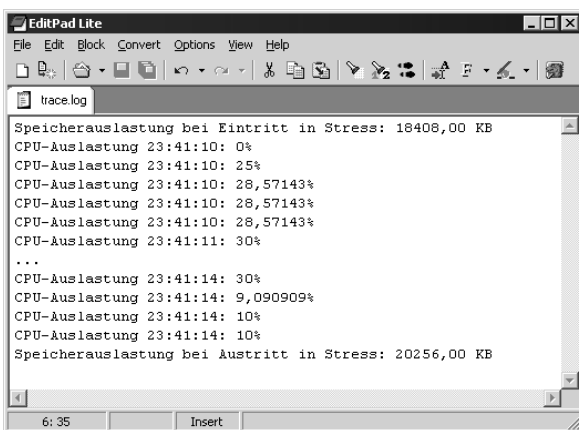


Abbildung 110: Die durch das Beispielprogramm erzeugte Trace-Protokolldatei

162 Eigene Leistungsindikatoren implementieren

In manchen Fällen ist es sinnvoll, bestimmte Leistungsdaten von Programmen von außen zu überwachen. Wenn Sie z.B. in einer Datenbankanwendung mit Transaktionen arbeiten, könnten Sie die Gesamtanzahl der Transaktionen und die Anzahl der aktuell offenen Transaktionen überwachen. Wenn Sie dazu eigene Leistungsindikatoren verwenden, können Sie diese Daten z.B. über den Systemmonitor überwachen (siehe Rezept 158).

Eigene Leistungsindikatoren können Sie über den Server-Explorer von Visual Studio .NET erzeugen. Wählen Sie dazu im Kontextmenü des Leistungsindikatoren-Eintrags (SERVER / <RECHNERNAME> / LEISTUNGSINDIKATOREN) den Befehl NEUE KATEGORIE ERSTELLEN. In der Praxis ist es aber sinnvoller, eigene Leistungsindikatoren im Programm zu erstellen, damit dieses problemlos auch auf Rechnern ausgeführt werden kann, die diese Leistungsindikatoren noch nicht besitzen.

Zur Erzeugung eigener Leistungsindikatoren müssen Sie zunächst eine eigene Leistungsindikatorenkategorie erstellen. Dazu ist wichtig zu wissen, dass Sie in bereits vorhandene Kategorien keine Leistungsindikatoren hinzufügen können. Das gilt sogar dann, wenn Sie die Kategorie selbst erzeugt haben. Sie können aber natürlich überprüfen, ob die Leistungsindikatorenkategorie bereits existiert, oder diese vor dem Erzeugen einfach löschen. Eigene Leistungsindikatoren und deren Kategorien bleiben innerhalb von Windows so lange bestehen, bis diese explizit gelöscht werden.

Zur Erzeugung einer Leistungsindikatorenkategorie rufen Sie die statische `Create`-Methode der `PerformanceCounterCategory`-Klasse aus dem Namensraum `System.Diagnostics` auf. Am ersten Argument übergeben Sie einen systemweit eindeutigen Namen für die Kategorie. Wenn Sie den Namen einer bereits existierenden Kategorie verwenden, resultiert `Create` in einer Ausnahme vom Typ `InvalidOperationException`. Verwenden Sie vielleicht einfach den Namen Ihrer Anwendung als Kategorienname.

Am zweiten Argument können Sie eine Beschreibung für diese Kategorie übergeben. Einer Überladung der `Create`-Methode können Sie dann noch den Namen eines Leistungsindikators und dessen Beschreibung übergeben. Da Sie damit aber wenig flexibel sind, sollten Sie die Überladung verwenden, die am dritten Argument eine Instanz der Klasse `CounterCreationDataCollection` erwartet. Diese Auflistung verwaltet Instanzen der Klasse `CounterCreationData`, die die zur Erstellung eines Leistungsindikators notwendigen Informationen speichern. Damit können Sie dann auch beliebig viele Leistungsindikatoren in der Kategorie erstellen. Der Konstruktor der `CounterCreationData`-Klasse erwartet am ersten Argument den Namen des Leistungsindikators, am zweiten eine Beschreibung (die z.B. im Systemmonitor angezeigt werden kann) und am dritten den Typ des Indikators (als Wert der Aufzählung `PerformanceCounterType`). Die wichtigen Typen habe ich bereits im Rezept 158 beschrieben.

Der Typ des Leistungsindikators bestimmt, wie dessen Wert beim Abruf über `NextValue` berechnet wird. Den rohen Wert eigener Leistungsindikatoren können Sie später zu beliebigen Zeitpunkten auf einen bestimmten Wert setzen oder einfach nur um einen bestimmten Wert erhöhen. Beim Abruf des berechneten Werts wird der Wert eines Leistungsindikators vom Typ `NumberOfItems32` z.B. einfach zurückgegeben. Andere Leistungsindikatoren wie z.B. `AverageTimer32` berechnen ihren Wert etwas komplizierter. Die Art der Berechnung wird bei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

der Dokumentation der `PerformanceCounterType`-Aufzählung recht gut beschrieben (wenn diese auch teilweise komplex ist). Ein `AverageTimer32`-Indikator berechnet z.B. die durchschnittliche Dauer einer Operation, die innerhalb eines Messintervalls u.U. mehrfach aufgerufen wird. Für einen solchen Indikator benötigen Sie einen Basis-Indikator vom Typ `AverageBase` und den `AverageTimer32`-Indikator selbst. Der `AverageBase`-Indikator, der dem `AverageTimer32`-Indikator direkt folgen muss, verwaltet die Gesamtanzahl der Operationen und muss bei jeder neuen Operation um 1 erhöht werden. Den `AverageTimer32`-Indikator aktualisieren Sie dann mit der gemessenen Zeitdauer der Operation in Systemticks (ein Systemtick ist das Basisintervall des Motherboard-Zeitgebers). Der Wert eines solchen Indikators wird automatisch in Sekunden umgerechnet.

ACHTUNG

Ein `AverageTimer32`-Indikator ist ein gutes Beispiel für eine schlechte Dokumentation. Bei diesem Indikator (und auch bei anderen, die Systemticks einsetzen) müssen Sie darauf achten, dass der Indikator mit Systemticks arbeitet und nicht, wie in vielen falschen Beispielen für diesen Indikator zu sehen, mit den Ticks des `DateTime`-Datentyps (100 Nanosekunden). Wenn Sie mit einem `AverageTimer32`-Indikator arbeiten, müssen Sie die Systemticks über die API-Funktion `QueryPerformanceCounter` ermitteln.

Listing 284 zeigt eine beispielhafte Verwendung eigener Leistungsindikatoren. Das Programm überprüft zunächst, ob die zu erzeugende Leistungsindikatorenkategorie (die den Namen der Anwendung trägt) bereits existiert, und löscht die Kategorie in diesem Fall. Damit wird sichergestellt, dass auch nachträgliche Änderungen an den Leistungsindikatoren problemlos im Quellcode vorgenommen werden können. Falls die Indikatoren nicht mehr geändert werden, können Sie die Kategorie auch einfach bestehen lassen und nur dann erzeugen, wenn sie noch nicht existiert.

Dann erstellt das Programm die Kategorie. Dazu wird zunächst eine Instanz der Klasse `CounterCreationDataCollection` erzeugt und mit den Daten für vier neue Leistungsindikatoren gefüllt. Die ersten zwei Indikatoren sind einfache Zähl-Indikatoren. Der dritte soll allerdings die mittlere Anzahl von offenen Transaktionen verwalten und ist deshalb vom Typ `AverageCount64`. Da dieser Indikator einen Basis-Indikator vom Typ `AverageBase` benötigt, der später mit der Gesamtanzahl der Zählvorgänge aktualisiert werden muss, wird dieser ebenfalls erzeugt und der Auflistung hinzugefügt. Über die `Create`-Methode der `PerformanceCounterCategory`-Klasse erzeugt das Programm dann die Kategorie.

Zum Kompilieren müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Diagnostics` und `System.Windows.Forms` importieren. Zur Ausführung des Programms ist Windows NT, Windows 2000, XP oder eine neuere Windows-Version notwendig.

```
// Ermitteln, ob die Leistungsindikatoren-Kategorie bereits existiert,
// und Löschen derselben, falls dies der Fall ist
if (PerformanceCounterCategory.Exists(Application.ProductName))
    PerformanceCounterCategory.Delete(Application.ProductName);
```

Listing 284: Erzeugen einer eigenen Leistungsindikatorkategorie mit Leistungsindikatoren

```
// CounterCreationDataCollection-Instanz für die Indikatorendaten der zu
// erzeugenden Leistungsindikatoren-Kategorie erzeugen und mit den Daten von
// drei Leistungs-Indikatoren (plus einem Basis-Indikator) füllen
CounterCreationDataCollection ccdCol = new CounterCreationDataCollection();
ccdCol.Add(new CounterCreationData("Gesamtanzahl der Transaktionen",
    "Verwaltet die Anzahl der insgesamt ausgeführten Transaktionen",
    PerformanceCounterType.NumberOfItems32));
ccdCol.Add(new CounterCreationData("Offene Transaktionen",
    "Zeigt die Anzahl der aktuell offenen Transaktionen an",
    PerformanceCounterType.NumberOfItems32));
ccdCol.Add(new CounterCreationData("Offene Transaktionen, Durchschnitt",
    "Zeigt die durchschnittlichen offenen Transaktionen",
    PerformanceCounterType.AverageCount64));
ccdCol.Add(new CounterCreationData("Offene Transaktionen, Durchschnitt, Basis",
    "Basis für den Counter 'Offene Transaktionen, Durchschnitt'",
    PerformanceCounterType.AverageBase));

// Leistungsindikatorenkategorie erzeugen
PerformanceCounterCategory.Create(Application.ProductName,
    "Demo für Leistungsindikatoren", ccdCol);
```

Listing 284: Erzeugen einer eigenen Leistungsindikatorenkategorie mit Leistungsindikatoren (Forts.)

Nun können Sie die Leistungsindikatoren referenzieren und aktualisieren. Dazu erzeugen Sie Instanzen der Klasse `PerformanceCounter`, denen Sie im Konstruktor den Kategorie- und den Leistungsindikatornamen übergeben. Am dritten Argument übergeben Sie `false`, damit der Indikator nicht schreibgeschützt ist.

```
PerformanceCounter pc1 = new PerformanceCounter(Application.ProductName,
    "Gesamtanzahl der Transaktionen", false);
PerformanceCounter pc2 = new PerformanceCounter(Application.ProductName,
    "Offene Transaktionen", false);
PerformanceCounter pc3 = new PerformanceCounter(Application.ProductName,
    "Offene Transaktionen, Durchschnitt", false);
PerformanceCounter pc4 = new PerformanceCounter(Application.ProductName,
    "Offene Transaktionen, Durchschnitt, Basis", false);
```

Listing 285: Referenzieren der eigenen Leistungsindikatoren

Den Wert von Leistungsindikatoren können Sie über die `Increment`- oder `IncrementBy`-Methode erhöhen bzw. über `IncrementBy` oder `Decrement` erniedrigen. Den rohen Wert des Indikators können Sie über die Eigenschaft `RawValue` lesen und setzen. Das folgende Beispiel zeigt dies für die Demo-Indikatoren. Das Programm erhöht in einer Schleife jeweils den Indikator für die Gesamtanzahl der Transaktionen um den Wert 1 und erzeugt für die Anzahl der offenen Transaktionen einen Zufallswert. Der Indikator für die Ermittlung des Durchschnitts der offenen Transaktionen wird über `IncrementBy` um die aktuelle Anzahl der offenen Transaktionen erhöht. Da dieser Indikator den Typ `AverageCount64` besitzt, wird der Basis-Indikator (`pc4`) dann noch um den Wert 1 erhöht. Damit das Ganze im Systemmonitor beobachtbar ist, ruft

das Programm am Ende der Schleife `Thread.Sleep` auf, um den aktuellen Thread für eine halbe Sekunde anzuhalten. Das Beispiel setzt übrigens den Indikator für die Gesamtanzahl der Transaktionen vor der Schleife zurück, da in meinen Versuchen der alte Wert des Indikators auch dann bestehen blieb, nachdem die Kategorie gelöscht und neu erzeugt wurde.

```
// Gesamtanzahl der Transaktionen zurücksetzen
pc1.RawValue = 0;

// Aktualisieren der eigenen Leistungsindikatoren
Random random = new Random();
for (int i = 0; i < 100; i++)
{
    int openTransactionCount = random.Next(10);
    pc1.Increment(); // Gesamtanzahl der Transaktionen erhöhen
    pc2.RawValue = openTransactionCount; // Anzahl der offenen Transaktionen
    int updateCount = i; // random.Next(10);
    pc3.IncrementBy(openTransactionCount); // Durchschnitt der offenen
                                           // Transaktionen
    pc4.Increment(); // Basis-Zähler für den Indikator pc3

    System.Threading.Thread.Sleep(500);
}
```

Listing 286: Aktualisieren von Leistungsindikatoren

Im Windows-Systemmonitor (siehe Rezept 158) können Sie nun Ihre Leistungsindikatoren hinzufügen (Abbildung 111).



Abbildung 111: Hinzufügen der eigenen Leistungsindikatoren im Systemmonitor. Das Programm, das die Indikatoren erzeugt hat, besitzt den Namen »Eigene Leistungsindikatoren«

Wenn Sie das Programm dann ausführen, können Sie die Leistungsindikatoren im Systemmonitor überwachen (Abbildung 112).

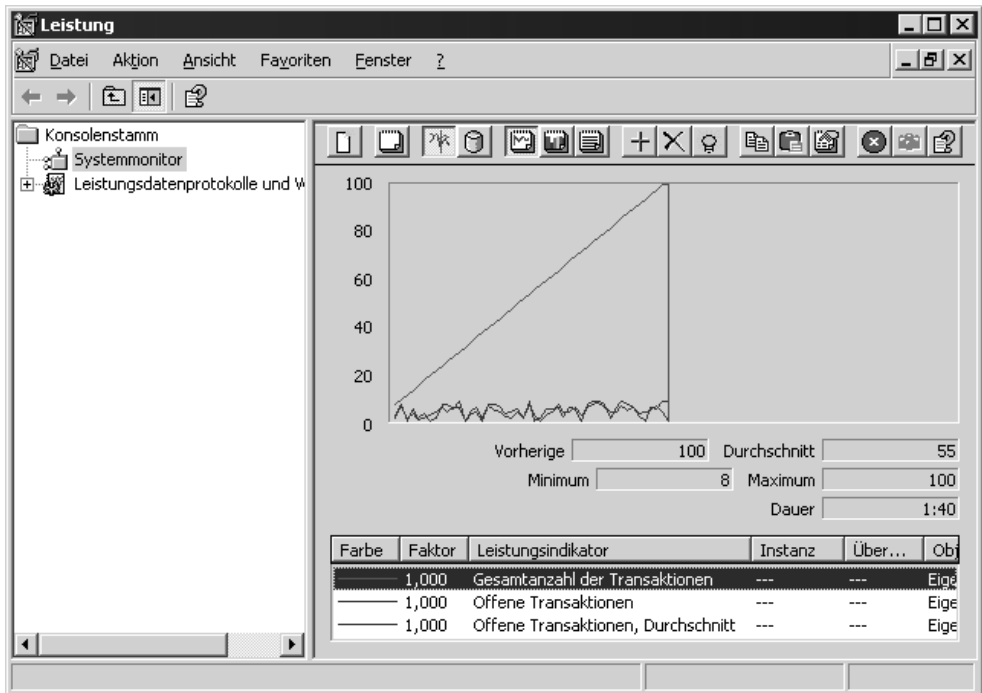


Abbildung 112: Die eigenen Leistungsindikatoren im Systemmonitor

Beachten Sie, dass der Indikator *Offene Transaktionen, Durchschnitt* vom Typ *AverageCount64* ist. Ein solcher Indikator berechnet (wie andere Durchschnitts-Indikatoren) bei einer Messung (also beim *Lesen* des Indikatorwerts, nicht beim *Schreiben* des rohen Werts!) den Durchschnitt aller Werte, die seit der vorhergehenden Messung geschrieben wurde. Er berechnet also den Durchschnitt der Aktionen im gegebenen Messintervall.

Erfolgt die Messung z. B. jede Sekunde und werden innerhalb dieser Sekunden die drei Werte 1, 5 und 6 geschrieben, ergibt das Lesen dieses Indikators den Wert 4. Daraus lässt sich erkennen, warum ein solcher Indikator über *IncrementBy* aktualisiert werden muss und warum ein zugeordneter *AverageBase*-Indikator die Anzahl der Werte verwaltet (wobei die folgenden Ausführungen lediglich eine Vermutung darstellen): Nach einer Messung setzt ein solcher Indikator den Wert auf 0 und speichert den aktuellen Wert des Basisindikators zwischen. *IncrementBy* erhöht den Wert dann ja wieder, u. U. bis zu nächsten Messung mehrfach. Bei einer Messung berechnet der Indikator einfach die Differenz zwischen dem aktuellen und dem letzten Wert des Basisindikators und teilt seinen Wert durch die so ermittelte Anzahl. Das ist doch einfach, oder (da frage ich mich, warum ich stundenlang in der Dokumentation und im Internet nach der Funktionsweise solcher Indikatoren gesucht habe 😊).

163 Monitor abschalten, in den Energiesparmodus schalten und wieder einschalten

Einen Monitor, der Energiespar-Features unterstützt, können Sie über die Windows-API-Funktion `SendMessage` abschalten, in den Energiesparmodus umschalten und wieder einschalten.

`SendMessage` sendet Windows-Nachrichten an einzelne oder an alle Top-Level-Fenster. Am ersten Argument übergeben Sie entweder den Handle des Zielfensters oder die Konstante `HWND_BROADCAST` (`0xFFFF`) zum Senden an alle Top-Level-Fenster.

Die zu sendenden Nachrichten werden am zweiten Argument über im Windows-API definierte `int`-Konstanten festgelegt. Die Konstante `WM_SYSCOMMAND` (`0x0112`) steht hier für das Senden einer Systemnachricht (die vom Anwender normalerweise über die Betätigung des Systemmenüs eines Fensters gesendet wird).

Über das dritte und vierte Argument übergeben Sie `SendMessage` Parameter für die Nachricht, zumeist wieder über im Windows-API definierte Konstanten. Um den Monitor ab-, um- oder einzuschalten, übergeben Sie am dritten Argument die Konstante `SC_MONITORPOWER` (`0xF170`). Bei dieser speziellen Nachricht bestimmt das vierte Argument, ob der Monitor abgeschaltet (2), in den Energiesparmodus umgeschaltet (1) oder eingeschaltet (-1) wird.

Da `SendMessage` Nachrichten immer an Fenster sendet, können Sie die Nachricht entweder an das Hauptfenster Ihrer Anwendung senden, indem Sie den Handle dieses Fensters aus der Eigenschaft `Handle` auslesen und an `SendMessage` übergeben. Um auf diese eigentlich unnötige Übergabe eines Fenster-Handles zu verzichten, können Sie die Nachricht stattdessen einfach an alle Top-Level-Fenster senden, indem Sie am ersten Argument `HWND_BROADCAST` übergeben. Da es sich dabei um eine `int`-Konstante handelt und `SendMessage` einen `IntPtr`-Typ erwartet, müssen Sie die Konstante in einen solchen konvertieren.

Die Klasse `Monitor` in Listing 287 implementiert das Abschalten, Umschalten und das Ausschalten des Monitors in statischen Methoden. Um diese Klasse kompilieren zu können, müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
public class Monitor
{
    // Deklaration der API-Funktion SendMessage
    [DllImport("User32.dll")]
    private static extern int SendMessage(IntPtr handle, int msg, int
        wparam, int lparam);

    // Deklaration der benötigten Konstanten
    private const int WM_SYSCOMMAND = 0x0112;
    private const int SC_MONITORPOWER = 0xF170;
    private const int HWND_BROADCAST = 0xFFFF;

    // Funktion zum Abschalten des Monitors
```

Listing 287: Methoden zum Ein-, Um- und Ausschalten des Monitors

```
public static void TurnOff()
{
    SendMessage((IntPtr)HWND_BROADCAST, WM_SYSCOMMAND, SC_MONITORPOWER, 2);
}

// Funktion zum Umschalten des Monitors in den Energiesparmodus
public static void SwitchToLowPower()
{
    SendMessage((IntPtr)HWND_BROADCAST, WM_SYSCOMMAND, SC_MONITORPOWER, 1);
}

// Funktion zum Einschalten des Monitors
public static void TurnOn()
{
    SendMessage((IntPtr)HWND_BROADCAST, WM_SYSCOMMAND,
        SC_MONITORPOWER, -1);
}
}
```

Listing 287: Methoden zum Ein-, Um- und Ausschalten des Monitors (Forts.)

Hinweis

Beim Test unter Windows 98 ließ sich der Monitor zwar abschalten, nicht aber (über einen Timer) wieder einschalten. Nach dem manuellen Einschalten des Monitors war das Programm abgestürzt und musste über den Taskmanager beendet werden. Die Fehlerursache habe ich nicht ausfindig machen können.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

164 Status der Internetverbindung abfragen

Wenn Sie abfragen wollen, ob auf einem Computer eine Internetverbindung besteht, finden Sie leider im .NET Framework keine Möglichkeit. Sie können jedoch die API-Funktionen `InternetGetConnectedState` und `InternetGetConnectedStateEx` aufrufen, um eine Information über den aktuellen Status der Internetverbindung zu erhalten. `InternetGetConnectedStateEx` bietet gegenüber der anderen Funktion den Vorteil, dass diese zusätzlich zum Status auch den Namen der Internetverbindung zurückliefert.

Listing 288 setzt den Aufruf dieser Funktion in der Methode `GetInternetConnectionState` um. `InternetGetConnectedStateEx` liefert einen Wert ungleich Null zurück, wenn zurzeit eine Internetverbindung besteht. Zusätzliche Informationen über den Status der Internetverbindung werden in Form einer `int`-Bitmaske am `out`-Argument `lpdwFlags` zurückgegeben. Die dazu verwendeten Konstanten sind in der Datei `wininit.h` deklariert. Am ersten Argument erwartet diese Funktion deshalb eine `int`-Variable. Am zweiten Argument wird der Name der Internetverbindung in Form eines String zurückgegeben, weswegen hier ein `StringBuilder`-Objekt übergeben wird, das ausreichend groß initialisiert wurde. Das dritte Argument teilt der API-Funktion lediglich mit, wie viele Zeichen in den String geschrieben werden können, das vierte (reservierte) Argument wird zurzeit nicht benutzt.

Listing 288 enthält zunächst die Deklaration der API-Funktion und der benötigten Konstanten. Die Methode `GetInternetConnectionState` gibt eine Instanz der Klasse `InternetConnectionState` zurück, die deswegen ebenfalls in Listing 288 deklariert ist. Diese Klasse beinhaltet den Namen und den Online-Status der Internetverbindung und alle verfügbaren Informationen über die Art der Verbindung. `GetInternetConnectionState` erzeugt zunächst eine `InternetConnectionState`-Instanz und ruft dann `InternetGetConnectedStateEx` auf, wobei die Rückgabe (in einen `bool`-Wert konvertiert) gleich in die Eigenschaft `Online` des `InternetConnectionState`-Objekts geschrieben wird. Danach werden der Name und die Status-Flags ausgewertet und in die `InternetConnectionState`-Instanz geschrieben.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System`, `System.Text` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration der API-Funktion InternetGetConnectedStateEx */
[DllImport("wininet.dll")]
private static extern int InternetGetConnectedStateEx(out int lpdwFlags,
    StringBuilder lpszConnectionName, int dwNameLen, int dwReserved);

/* Konstanten für InternetGetConnectedStateEx */
private const int INTERNET_CONNECTION_MODEM = 0x01;
```

Listing 288: Methode zur Ermittlung des Internetverbindungsstatus

```
private const int INTERNET_CONNECTION_LAN = 0x02;
private const int INTERNET_CONNECTION_PROXY = 0x04;
private const int INTERNET_RAS_INSTALLED = 0x10;
private const int INTERNET_CONNECTION_OFFLINE = 0x20;
private const int INTERNET_CONNECTION_CONFIGURED = 0x40;

/* Klasse für die Rückgabe des Internet-Verbindungsstatus */
public class InternetConnectionState
{
    // Der Name der Verbindung
    public string Name;
    // Info, ob die Verbindung online ist
    public bool Online;
    // Info, ob die Verbindung konfiguriert ist
    public bool Configured;
    // Info, ob eine Modem-Verbindung besteht (auch ISDN und DSL)
    public bool ModemConnection;
    // Info, ob die Verbindung über das LAN erfolgt (z. B. bei TDSL)
    public bool Lan;
    // Info, ob die Verbindung über einen Proxy erfolgt
    public bool ProxyConnection;
    // Info, ob RAS installiert ist
    public bool RASInstalled;
    // Info, ob das System im Offline-Modus ist
    public bool Offline;
}

/* Methode zur Ermittlung der Art der aktuellen Internetverbindung */
public static InternetConnectionState GetInternetConnectionState()
{
    // InternetConnectionState-Instanz erzeugen
    InternetConnectionState ics = new InternetConnectionState();

    // Verbindungsstatus abfragen
    StringBuilder icsName = new StringBuilder(1024);
    int flags;
    ics.Online = (InternetGetConnectedStateEx(out flags, icsName,
        1024, 0) != 0);
    ics.Name = icsName.ToString();
    ics.Configured = ((flags & INTERNET_CONNECTION_CONFIGURED) > 0);
    ics.Lan = ((flags & INTERNET_CONNECTION_LAN) > 0);
    ics.ModemConnection = ((flags & INTERNET_CONNECTION_MODEM) > 0);
    ics.Offline = ((flags & INTERNET_CONNECTION_OFFLINE) > 0);
    ics.ProxyConnection = ((flags & INTERNET_CONNECTION_PROXY) > 0);
    ics.RASInstalled = ((flags & INTERNET_RAS_INSTALLED) > 0);

    // Das InternetConnectionState-Objekt zurückgeben
    return ics;
}
```

Listing 288: Methode zur Ermittlung des Internetverbindungsstatus (Forts.)

Der Test der Methode `GetInternetConnectionState` hat einige logische Dinge, aber auch einige Ungereimtheiten ergeben:

- ▶ Die Eigenschaft `Online` ist (nach meinen Tests unter Windows XP und Windows 2000) dummerweise eigentlich fast immer `true`. Wenn zurzeit keine Internetverbindung besteht, aber eine LAN-Verbindung vorhanden ist, wird diese als Internetverbindung gewertet, auch wenn gar keine Internetverbindung besteht.
- ▶ Die Eigenschaft `Offline` sagt nicht aus, dass die Internetverbindung zurzeit nicht besteht, da diese Eigenschaft auch bei einer nicht bestehenden Internetverbindung `false` ist (!?).
- ▶ Die Eigenschaft `ModemConnection` ist zuverlässig nur dann `true`, wenn zurzeit eine Verbindung über ein Telefonmodem, ein DSL-Modem, ein ISDN-Modem oder eine ISDN-Karte besteht. Diese Eigenschaft ist also die einzige, die eine konkrete Aussage über den Zustand der Internetverbindung bei einer Modem-Verbindung oder einer Verbindung über eine ISDN-Karte macht.
- ▶ `ProxyConnection` ist `true`, wenn das System (über den Internet Explorer) so konfiguriert ist, dass ein Proxy-Server verwendet wird. Dazu muss weder eine Internetverbindung bestehen, noch der Proxy-Server ausgeführt werden.
- ▶ `Lan` ist `true`, wenn eine Lan-Verbindung besteht und `ModemConnection` `false` ist. `Lan` ist leider auch `true`, wenn gar keine Internetverbindung besteht oder wenn eine Modem-Internetverbindung offline ist.

Hinweis

Eine zuverlässige Aussage darüber, ob gerade eine Internetverbindung besteht, können Sie also nur dann über die Eigenschaft `ModemConnection` treffen, wenn diese auf dem lokalen System über ein Telefonmodem, ein DSL-Modem, ein ISDN-Modem oder eine ISDN-Karte ausgeführt wird. Um sicherzugehen, dass eine Verbindung besteht, sollten Sie vielleicht besser einen Ping an eine zuverlässige Internetadresse (z.B. `www.google.com`) ausführen (siehe Rezept 165).

Listing 289 zeigt eine beispielhafte Anwendung der `GetInternetConnectionState`-Methode.

```
try
{
    // Internetverbindungs-Status abfragen
    InternetConnectionState ics = GetInternetConnectionState();

    // Ergebnis ausgeben
    Console.WriteLine("Name: {0}", ics.Name);
    Console.WriteLine("Online: {0}", ics.Online);
    Console.WriteLine("Configured: {0}", ics.Configured);
    Console.WriteLine("Lan: {0}", ics.Lan);
    Console.WriteLine("ModemConnection: {0}", ics.ModemConnection);
    Console.WriteLine("Offline: {0}", ics.Offline);
}
```

Listing 289: Beispielhafte Anwendung der `GetInternetConnectionState`-Methode

```

    Console.WriteLine("ProxyConnection: {0}", ics.ProxyConnection);
    Console.WriteLine("RASInstalled: {0}", ics.RASInstalled);
}
catch (Exception ex)
{
    Console.WriteLine("Fehler bei der Abfrage des Internet-" +
        "Verbindungsstatus: {0}",
        ex.Message);
}

```

Listing 289: Beispielhafte Anwendung der `GetInternetConnectionState`-Methode (Forts.)



Abbildung 113: Ausführung des Beispielprogramms bei einer bestehenden DSL-Verbindung

165 Pingen: Ermitteln, ob eine Internetverbindung besteht

Wenn Sie ermitteln wollen, ob eine Internetverbindung besteht, reicht es nicht aus, den Status der Internetverbindung abzufragen (siehe Rezept 164), da der Aufruf der API-Funktion `InternetGetConnectedStateEx` leider auch `true` ergibt, wenn keine Internetverbindung, aber eine LAN-Verbindung besteht.

Zur Überprüfung auf eine geöffnete Internetverbindung könnten Sie die API-Funktion `InternetCheckConnection` einsetzen, der Sie eine URL oder `null` am ersten Argument übergeben können, um die Erreichbarkeit der URL oder einfach nur eine vorhandene Internetverbindung zu überprüfen. Leider arbeitete diese Funktion in meinen Tests nicht mit einer DSL-Verbindung oder einer Internetverbindung über einen Proxy-Server (der Aufruf führte zum (über `FormatMessage` ausgelesenen) Fehler »Diese Netzwerkverbindung ist nicht vorhanden«). Laut einigen Newsgroup-Beiträgen können Sie diese Funktion auch nicht einsetzen, wenn die Internetverbindung über das LAN erfolgt. `InternetCheckConnection` hat mit DSL- und LAN-Verbindungen wahrscheinlich ähnliche Probleme wie `InternetGetConnectedStateEx`.

Die sicherste Möglichkeit, eine offene Internetverbindung abzufragen, ist, an eine bekannte und zuverlässige IP-Adresse (wie z. B. 216.239.51.99 für `www.google.com`) einen oder mehrere Pings abzusetzen. Ein Ping (Packet Internet Groper = Paket-Internet-(Ab)Taster) ist eine Technik, bei der ein spezielles IP-Paket an den Port 7 eines Host gesendet wird. Jeder Host sollte die von einem Ping aufgerufene ICMP¹-Echo-Funktion (definiert in RFC 792) imple-

1. ICMP = Internet Control Message Protocol

mentieren. Der Empfänger sendet ein spezielles IP-Paket zurück, wenn er erreichbar ist. Da jedes Paket mit einer eindeutigen Sequenznummer und einer Checksumme versehen wird, kann der Sender beim mehrfachen Pinggen erkennen, wie viele Pakete auf dem Weg verloren gegangen sind, wie viele Pakete auf dem Weg zerstört wurden und wie viel Zeit das Senden und Empfangen in Anspruch genommen hat.

Zur Überprüfung einer offenen Internetverbindung sollten Sie explizit die IP-Adresse verwenden und nicht den Domännennamen. Der Grund dafür ist, dass ein Ping an einen Domännennamen eine DNS-Abfrage verursacht, die bei Systemen mit einer automatischen Wählverbindung dazu führt, dass die Verbindung geöffnet wird.

Listing 290 implementiert eine Methode `Ping`, über die Sie einen Ping absetzen können. Dieser Methode werden die Adresse des Host und ein Timeout für den Ping in Millisekunden übergeben. `Ping` gibt einen Wert der Aufzählung `PingResult` zurück, der aussagt, ob der Ping erfolgreich war oder ob ein Timeout oder Prüfsummenfehler eingetreten ist.

`Ping` ermittelt zunächst über `Dns.Resolve` die IP-Adresse des anzusprechenden Host. Da `Resolve` ein `IPHostEntry`-Objekt mit eventuell mehreren ermittelten Adressen in der Eigenschaft `AddressList` zurückgibt, wird die erste dieser Adressen als IP-Adresse verwendet.

Dann erzeugt `Ping` eine `EndPoint`- und eine `Socket`-Instanz für das spätere Senden der ICMP-Daten. Als zu sendende Daten wird ein acht Byte großes Byte-Array erzeugt, das entsprechend der in RFC 792 (<ftp.rfc-editor.org/in-notes/rfc792.txt>) definierten Echo-Nachricht zusammengesetzt wird. Am ersten Byte dieser Nachricht wird 8 für den Typ (ICMP-Echo) übergeben. Das zweite Byte (im Original ist das das Feld *Code*) muss mit 0 belegt werden. Das dritte und vierte Byte verwalten als 16-Bit-Wort die Prüfsumme. Diese Prüfsumme, die übrigens der TCP-Prüfsumme entspricht, ist das Einerkomplement der Einerkomplementsumme aller 16-Bit-Wörter der Nachricht. In Rezept 166 finden Sie eine Berechnung dieser Prüfsumme. `Ping` setzt hier den Wert `0xF7FF` ein, da dies die berechnete Prüfsumme für die gesendeten Daten ist.

Die beiden nächsten 16-Bit-Wörter (im Original: *Identifizier* und *Sequence Number*) können mit einem beliebigen Wert belegt werden, der zur Identifizierung der empfangenen Daten verwendet werden kann. `Ping` trägt hier jeweils 0 ein, da nur eine einzelne Echo-Nachricht abgesetzt wird. In den dem ICMP-Header folgenden Daten trägt `Ping` 8 Byte ein, die jedoch nur mit Nullen gefüllt werden.

Das so zusammengestellte Byte-Array wird dann über die `SentTo`-Methode der `Socket`-Instanz an den Port 7 der angegebenen Adresse gesendet. Danach überprüft `Ping` über `Socket.Select`, ob im gegebenen Timeout (der `Select` allerdings in Mikrosekunden übergeben wird) Daten zum Lesen bereitstehen. `Select` wartet bis zum angegebenen Timeout und entfernt alle `Socket`-Instanzen aus dem am ersten Argument übergebenen `ArrayList`-Objekt, die nicht gelesen werden können. Eigentlich wird `Select` verwendet, um aus einer Menge von Sockets diejenigen herauszufiltern, die zum Lesen bzw. Schreiben bereitstehen. Als kleinen Trick können Sie damit auch bis zu einem gegebenen Timeout darauf warten, dass in einem Socket Daten zum Lesen bereitstehen, indem Sie das `ArrayList`-Objekt nach dem Aufruf von `Select` daraufhin überprüfen, ob es nicht leer ist (wie es im Listing gezeigt wird).

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Stehen Daten zum Lesen zur Verfügung, liest `Ping` diese ein. Der Puffer für die Echo-Antwort wird dabei auf 200 Byte initialisiert, um genügend Platz für die Daten zu schaffen. Über die Methode `GetTcpChecksum` (aus dem Rezept 166) überprüft `Ping` dann die Prüfsumme des ICMP-Headers (der zuvor natürlich extrahiert wird). Bei der Berechnung einer Prüfsumme über einen Header, der bereits eine Prüfsumme enthält, resultiert 0, wenn die Daten des Headers in Ordnung sind. In diesem Fall gibt `Ping` den Wert `PingResult.OK` zurück. Ist die Prüfsumme nicht in Ordnung, wird `PingResult.ChecksumError` zurückgegeben.

Stehen keine Daten zum Lesen zur Verfügung, gibt `Ping` den Wert `PingResult.Timeout` zurück.

Zum Kompilieren der `Ping`-Methode müssen Sie die Namensräume `System`, `System.Net`, `System.Net.Sockets` und `System.Collections` importieren.

```

/* Aufzählung für die Rückgabe von Ping */
public enum PingResult
{
    OK,
    Timeout,
    ChecksumError
}

/* Methode zum Pinggen */
public static PingResult Ping(string address, int timeout)
{
    // IP-Adresse des Host ermitteln
    IPHostEntry ipHostEntry = Dns.Resolve(address);
    IPAddress ipAddress = ipHostEntry.AddressList[0];

    // IPEndPoint-Instanz für den Port 7 der angegebenen Adresse erzeugen
    IPEndPoint epHost = new IPEndPoint(ipAddress, 7);

    // Socket für das Senden von ICMP-Daten erzeugen
    Socket socket = new Socket(AddressFamily.InterNetwork,
        SocketType.Raw, ProtocolType.Icmp);

    // Byte-Array für den 8-Byte-ICMP-Echo-Header und für acht Byte Daten
    // erzeugen und initialisieren
    byte[] sendBuffer = {8, 0, 0xF7, 0xFF, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    // Daten senden
    int sent = socket.SendTo(sendBuffer, sendBuffer.Length,
        SocketFlags.None, epHost);

    // Über Socket.Select prüfen, ob im gegebenen Timeout Daten zum Lesen
    // bereitstehen
    ArrayList checkReadSockets = new ArrayList();
    checkReadSockets.Add(socket);
    Socket.Select(checkReadSockets, null, null, timeout * 1000);

```

Listing 290: Methode zum Pinggen

```

// Lesen der Daten
byte[] receiveBuffer = new byte[200];
int receivedBytes = 0;
if (checkReadSockets.Count > 0)
{
    // Wenn das Socket nicht aus der ArrayList gelöscht wurde,
    // sind Daten zum Lesen verfügbar: Die ECHO-Antwort empfangen
    receivedBytes = socket.Receive(receiveBuffer,
        receiveBuffer.Length, SocketFlags.None);

    // Prüfsumme vergleichen (muss bei einer erneuten Berechnung 0 ergeben)
    if (GetTCPChecksum(receiveBuffer) == 0)
        return PingResult.OK;
    else
        return PingResult.ChecksumError;
}
else
    return PingResult.Timeout;
}

```

Listing 290: Methode zum Pingen (Forts.)

Listing 291 zeigt eine Anwendung der Ping-Methode in einer Konsolenanwendung.

```

// Prüfen, ob eine Internetverbindung nach www.google.de möglich ist
string address = "www.google.de";
PingResult pingResult = Ping(address, 1000);
switch (pingResult)
{
    case PingResult.OK:
        Console.WriteLine("Ping an {0} war erfolgreich", address);
        break;
    case PingResult.Timeout:
        Console.WriteLine("Timeout beim Ping an {0}", address);
        break;
    case PingResult.ChecksumError:
        Console.WriteLine("Daten wurden empfangen, aber es ist ein " +
            "Prüfsummenfehler aufgetreten {0}", address);
        break;
}

```

Listing 291: Anwendung der Ping-Methode

166 TCP- und ICMP-Prüfsumme berechnen

Wenn Sie TCP- oder ICMP-Daten roh versenden oder empfangen, z.B. beim Pingen, wie ich es im Rezept 165 beschreibe, müssen Sie u.U. die Prüfsumme der Daten berechnen. Diese Prüfsumme wird mit den einzelnen Wörtern der TCP-/ICMP-Daten berechnet, bei TCP in das neunte Wort und bei ICMP in das zweite Wort geschrieben.

Die Methode `GetTCPChecksum` in Listing 292 nimmt diese Berechnung vor. `GetTCPChecksum` bekommt ein Byte-Array mit den TCP- oder ICMP-Daten übergeben. Bei der ersten Berechnung einer Prüfsumme müssen die Bytes, die die Prüfsumme aufnehmen, 0 speichern. Wenn Sie die Prüfsumme mit Daten berechnen, deren Header bereits eine Prüfsumme speichert, muss (aufgrund des Einer-Komplements) 0 resultieren. Daran können Sie erkennen, dass die Prüfsumme empfangener Daten in Ordnung ist und die Daten somit nicht verändert wurden.

Ich habe `GetTCPChecksum` übrigens basierend auf einigen Beispielen im Internet entwickelt. Das maßgebliche Beispiel war die C-Implementierung des Ping-Befehls des Ping-Erfinders Mike Muus (www.ping127001.com/pingpage/ping.html).

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` einbinden.

```
public static ushort GetTCPChecksum(byte[] tcpData)
{
    // Wenn das byte-Array am Ende kein ganzes Wort mehr ergibt, muss ein 0-
    // Byte angehängt werden
    byte[] buffer;
    if ((tcpData.Length % 2) > 0)
    {
        buffer = new byte[tcpData.Length + 1];
        tcpData.CopyTo(buffer, 0);
        buffer[buffer.Length - 1] = 0;
    }
    else
    {
        buffer = new byte[tcpData.Length];
        tcpData.CopyTo(buffer, 0);
    }

    // Die einzelnen Wörter addieren
    int checksum = 0;
    for (int i = 0; i < buffer.Length; i += 2)
    {
        byte lowByte = buffer[i+1];
        byte highByte = buffer[i];
        ushort wordValue = highByte;
        wordValue = (ushort)((wordValue << 8) + lowByte);
        checksum += wordValue;
    }

    // Nur die niedrigen 16 Bits addiert mit den hohen 16 Bits auslesen
    checksum = (checksum >> 16) + (checksum & 0xFFFF);
    checksum += (checksum >> 16);

    // Einerkomplement bilden
    return (ushort)(~checksum);
}
```

Listing 292: Methode zur Berechnung einer TCP/ICMP-Prüfsumme

167 Internetverbindung öffnen und schließen

Über die API-Funktion `InternetAutodial` können Sie die Standard-Internet-Wählverbindung des Rechners, auf dem das Programm ausgeführt wird, öffnen und über `InternetAutodialHangup` wieder schließen.

Hinweis

`InternetAutodial` öffnet leider nur Internet-Wählverbindungen (über ein Modem oder eine ISDN-Karte). DSL-Verbindungen (die keine Wählverbindungen sind), können mit `InternetAutodial` nicht geöffnet werden. Ähnliches gilt für `InternetAutodialHangup`, die eine DSL-Verbindung nicht schließt. Eine Lösung zum Öffnen und Schließen einer DSL-Verbindung habe ich bisher nicht gefunden.

`InternetAutodial` erwartet am ersten Argument eine `int`-Konstante, die Informationen darüber enthält, wie die Verbindung geöffnet werden soll:

- ▶ `INTERNET_AUTODIAL_FORCE_ONLINE` bewirkt, dass eine Online-Verbindung erzeugt wird,
- ▶ `INTERNET_AUTODIAL_FORCE_UNATTENDED` führt dazu, dass die Verbindung im Hintergrund geöffnet wird, ohne dass der Anwender im Einwahldialog den Wählen-Schalter betätigen muss,
- ▶ `INTERNET_AUTODIAL_FAILIFSECURITYCHECK` bewirkt, dass `InternetAutodial` fehlschlägt, wenn die Drucker- und Datei-Freigabe deaktiviert ist.

Sie können an diesem Argument nur eine der Konstanten angeben (das Argument ist keine Bitmaske). Am zweiten Argument können Sie den Handle eines Formulars übergeben, das dann als Parent für den Einwahl-Dialog verwendet wird. Wenn Sie kein Parent-Formular verwenden (wollen), können Sie hier auch 0 übergeben.

Die Funktion `InternetAutodialHangup`, der Sie am ersten, für spätere Zwecke reservierten Argument 0 übergeben, schließt die Standard-Internetverbindung.

Beide Funktionen geben `true` zurück, wenn die Ausführung erfolgreich war. Leider können Sie damit nicht unbedingt eine Aussage treffen, dass die Verbindung auch geöffnet ist: Ist die Standard-Netzwerkverbindung eine DSL-Verbindung, geben beide Funktionen ebenfalls `true` zurück, obwohl die Verbindung weder geöffnet noch geschlossen wird. Sie sollten also nach dem Öffnen gegebenenfalls überprüfen, ob eine Verbindung besteht (siehe Rezept 165).

Die im Folgenden beschriebenen Methoden `OpenLocalConnection` und `CloseLocalConnection` übernehmen das Öffnen und Schließen der Internetverbindung. Zum Kompilieren dieser Methoden müssen Sie die Namensräume `System`, `System.Text` und `System.Runtime.InteropServices` importieren und die notwendigen API-Funktionen und -Konstanten deklarieren. Um eventuelle (leider undokumentierte) Fehler bei der Ausführung der Funktionen auswerthen zu können, habe ich die Funktion `FormatMessage` mit aufgenommen. Da die Fehlermeldungen von `InternetAutodial` und `InternetAutodialHangup` nicht in der Windows-System-Meldungstabelle, sondern in der Datei `wininet.dll` gespeichert sind, ist die Funktion `GetModuleHandle` ebenfalls deklariert.

```

[DllImport("wininet.dll", SetLastError=true)]
public static extern int InternetAutodial(int dwFlags,
    IntPtr hwndParent);

[DllImport("wininet.dll", SetLastError=true)]
public static extern int InternetAutodialHangup(int dwReserved);

[DllImport("Kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
    int dwMessageId, int dwLanguageId,
    System.Text.StringBuilder lpBuffer, int nSize, string [] Arguments);

[DllImport("kernel32.dll")]
static extern IntPtr GetModuleHandle(string lpFileName);

/* Konstanten für InternetAutoDial */
private const int INTERNET_AUTODIAL_FORCE_ONLINE = 1;
private const int INTERNET_AUTODIAL_FORCE_UNATTENDED = 2;
private const int INTERNET_AUTODIAL_FAILIFSECURITYCHECK = 4;

/* Konstanten für FormatMessage */
private const int FORMAT_MESSAGE_FROM_HMODULE = 0x0800;

```

Listing 293: API-Deklarationen für InternetAutodial und InternetAutodialHangup

Die Methode `OpenLocalConnection` in Listing 294 öffnet eine Wahlverbindung zum Internet. Am Argument `unattended` wird definiert, ob das Öffnen unbeaufsichtigt erfolgen soll (also ohne dass der Anwender den Wählen-Schalter des Verbindungsdialogs betätigen muss). Das Argument `hwndParent` kann mit dem Windows-Handle eines Formulars belegt werden, das als Parent-Fenster des Einwahl-Dialogs verwendet werden soll.

`OpenLocalConnection` definiert zunächst die zu übergebenden Flags und ruft dann `InternetAutodial` auf. Wenn die Funktion 0 zurückliefert, liest `OpenLocalConnection` den letzten API-Fehlercode aus und versucht, damit aus der Datei *wininet.dll* eine Fehlerbeschreibung auszu-lesen.

Hinweis

Leider führten meine Versuche nicht dazu, dass eine Beschreibung ausgelesen werden konnte. Der durch einen Abbruch des Wahl-Dialogs erzeugte Fehlercode 631 ist nicht mit einer Meldung verknüpft (die Fehlercodes in der Meldungstabelle der *wininet.dll* beginnen auch erst bei 12000, was ich über das Test-Programm »Fehlercodes ermitteln« herausgefunden habe, das Sie im Ordner *Beispiele/Tools* auf der Buch-CD finden).

```

public static void OpenLocalConnection(bool unattended, int hwndParent)
{
    // Flags definieren
    int flags = 0;
    if (unattended)
        flags = INTERNET_AUTODIAL_FORCE_UNATTENDED;
    else
        flags = INTERNET_AUTODIAL_FORCE_ONLINE;

    // Verbindung wählen
    if (InternetAutodial(flags, (IntPtr)hwndParent) == 0)
    {
        // Fehler bei der Ausführung
        int apiError = Marshal.GetLastWin32Error();

        // Den Text des Fehlers aus der Datei wininet.dll auslesen und damit
        // eine Ausnahme werfen
        IntPtr hModule = GetModuleHandle("wininet.dll");
        StringBuilder message = new StringBuilder(1024);
        if (FormatMessage(FORMAT_MESSAGE_FROM_HMODULE, hModule,
            apiError, 0, message, 1024, null) > 0)
            throw new Exception(message.ToString());
        else
            throw new Exception("API-Fehler " + apiError);
    }
}

```

Listing 294: Methode zum Öffnen einer lokalen Internetverbindung

Die Methode `CloseLocalConnection` ruft `InternetAutodialHangup` auf um die aktuelle Verbindung zu schließen. Fehler bei der Ausführung werden auf dieselbe Weise ausgewertet wie bei `OpenLocalConnection`.

```

public static void CloseLocalConnection()
{
    // Verbindung schließen
    if (InternetAutodialHangup(0) == 0)
    {
        // Fehler bei der Ausführung
        int apiError = Marshal.GetLastWin32Error();

        // Den Text des Fehlers aus der Datei wininet.dll auslesen
        // und damit eine Ausnahme werfen
        IntPtr hModule = GetModuleHandle("wininet.dll");
        StringBuilder message = new StringBuilder(1024);
        if (FormatMessage(FORMAT_MESSAGE_FROM_HMODULE, hModule, apiError,
            0, message, 1024, null) > 0)
            throw new Exception(message.ToString());
    }
}

```

Listing 295: Methode zum Schließen einer lokalen Internetverbindung

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
        else
            throw new Exception("API-Fehler " + apiError);
    }
}
```

Listing 295: Methode zum Schließen einer lokalen Internetverbindung (Forts.)

Listing 296 zeigt eine Beispiel-Anwendung der Methode `OpenLocalConnection`, bei der das Öffnen nicht unbeaufsichtigt erfolgt.

```
try
{
    OpenLocalConnection(false, 0);

    Console.WriteLine("Internetverbindung ist geöffnet");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

Listing 296: Anwendung der Methode `OpenLocalConnection`



Abbildung 114: Der DFÜ-Einwahldialog, der nach dem Aufruf von `OpenLocalConnection` geöffnet wurde

168 Die IP-Adressen des Computers herausfinden

Normalerweise besitzt ein Computer mindestens eine, meist aber sogar mehrere IP-Adressen. Eine Adresse liegt häufig im Bereich des lokalen Netzwerks und ist der Netzwerkkarte zugeordnet, die für die lokale Netzwerkverbindung verwendet wird. Eine weitere Adresse ist häufig der aktuellen Internetverbindung zugeordnet, wenn der Computer eine solche direkt (und nicht über das Netzwerk) aufbaut.

Sie können die IP-Adressen eines Computers über DNS herausfinden. Die Methode `GetHostByName` der `Dns`-Klasse aus dem Namensraum `System.Net`, der Sie den Namen des Computers übergeben können, liefert ein `IPHostEntry`-Objekt zurück, das neben dem Host-Namen und Aliasnamen für den Host in der Eigenschaft `AddressList` ein Array von `IPAddress`-Objekten verwaltet. Über die Eigenschaft `Address` eines solchen Objekts erhalten Sie die jeweilige IP-Adresse, allerdings als `long`-Wert. Da die übliche Darstellung einer IP-Adresse die vier Bytes des `long`-Werts einzeln und durch Punkte getrennt darstellt, sollten Sie stattdessen die `ToString`-Methode aufrufen, die die IP-Adresse in der üblichen Form zurückgibt. `GetHostByName` können Sie leider nicht den Namen `localhost` oder die IP-Adresse `127.0.0.1` übergeben um den lokalen Computer anzusprechen, in diesem Fall resultiert lediglich die Adresse `127.0.0.1`. Sie müssen den DNS-Namen des Computers herausfinden und dieser Methode übergeben, was aber über `Dns.GetHostName` sehr einfach ist.

Das Beispiel erfordert den Import der Namensräume `System` und `System.Net`.

```
IPAddress[] addressList = Dns.GetHostByName(Dns.GetHostName()).AddressList;
for (int i = 0; i < addressList.Length; i++)
{
    string ipAddress = addressList[i].ToString();
    Console.WriteLine(ipAddress);
}
```

Listing 297: Die IP-Adressen des lokalen Computers über DNS auslesen

Diese Variante funktioniert nur dann, wenn DNS auf dem lokalen System korrekt konfiguriert und nicht deaktiviert ist. Das ist zwar bei den meisten Systemen der Fall, Sie können dies aber nicht voraussetzen. Deshalb sollten Sie die andere Lösung einsetzen, die die IP-Adressen über WMI abfragt. Diese Lösung geht alle Netzwerkadapter des Systems über die WMI-Klasse `Win32_NetworkAdapterConfiguration` durch, fragt ab, ob der Adapter IP unterstützt, und liest die den Adapter zugeordneten IP-Adressen aus.

Die Methode `GetLocalIPAddresses` in Listing 298 geht auf diese Weise vor. Um die Verarbeitung der Adressen zu erleichtern gibt `GetLocalIPAddresses` ein `string`-Array mit den ermittelten Adressen zurück. Um dieses füllen zu können, speichert die Methode die Adressen zunächst in einer `ArrayList`-Instanz und kopiert diese dann in das neu erzeugte `String`-Array. Zum Kompilieren der Methode müssen Sie die Assembly `System.Management.dll` referenzieren und die Namensräume `System`, `System.Collections`, `System.Net` und `System.Management` importieren.

```
public static string[] GetLocalIPAddresses()
{
    string[] result = null;

    try
    {
        ArrayList addressList = new ArrayList();

        // Alle Netzwerkadapter des Systems ermitteln und durchgehen
        ManagementClass mc = new ManagementClass(
            "Win32_NetworkAdapterConfiguration");
        ManagementObjectCollection moc = mc.GetInstances();
        foreach (ManagementObject mo in moc)
        {
            if ((bool)mo["IpEnabled"])
            {
                // Wenn der Adapter das IP-Protokoll unterstützt:
                // Adressen auslesen
                string[] ipAddresses = (string[])mo["IPAddress"];
                foreach (string ipAddress in ipAddresses)
                    addressList.Add(ipAddress);
            }
        }

        // ArrayList in String-Array kopieren
        if (addressList.Count > 0)
        {
            result = new string[addressList.Count];
            addressList.CopyTo(result);
        }
        else
            result = new string[0];
    }
    catch {}

    return result;
}
```

Listing 298: Methode zum Auslesen der lokalen IP-Adressen

Abbildung 115 zeigt die Ausgabe einer Konsolenanwendung, die die IP-Adressen einmal über DNS und zum anderen über `GetLocalIPAddresses` auf meinem System aufgerufen und die Adressen jeweils ausgegeben hat.

Das Problem, herauszufinden, welche der ermittelten Adressen nun die Adresse der Internetverbindung ist, konnte ich bisher leider nicht lösen. Ein Lösungsansatz ist, die IP-Adresse mit der Subnetzmaske des Providers zu vergleichen, sofern diese sich von der lokalen Subnetzmaske unterscheidet.

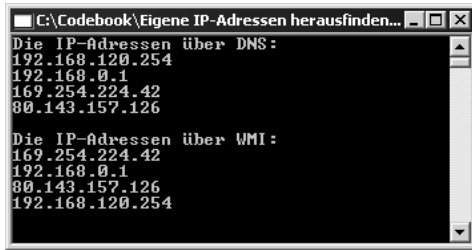


Abbildung 115: Auslesen der lokalen IP-Adressen

169 Mails über einen SMTP-Server versenden

Das Versenden von E-Mails ist nicht nur interessant für die Programmierung von eigenen E-Mail-Clients (von denen es ja eigentlich bereits mehr als genug gibt). E-Mails sind auch sinnvoll für Anwendungen, die bestimmte Personen (in einem Unternehmen) über einen neu eingetretenen Zustand informieren müssen. In einem Projekt-Planungssystem ist es z. B. sinnvoll, alle Projekt-Mitarbeiter in Kenntnis zu setzen, nachdem wesentliche Daten wie der Einführungsstermin vom Projektleiter geändert wurden. Eine elegante Programmlösung versendet entsprechende E-Mails in solchen Fällen automatisch und nimmt den Mitarbeitern so einiges an Arbeit ab.

Sofern ein SMTP-Server zur Verfügung steht, können Sie zum Senden von E-Mails die Klasse `Smtplib` aus dem Namensraum `System.Web.Mail` verwenden. Um diesen Namensraum zur Verfügung zu haben, müssen Sie die Assembly `System.Web.dll` referenzieren. Die einzige wichtige Einschränkung dieser Klasse ist, dass sie keine Authentifizierung gegenüber dem SMTP-Server unterstützt.

Falls kein SMTP-Server, sondern ein Exchange-Server zur Verfügung steht, finden Sie eine Lösung übrigens im Rezept 170.

Zum Senden einer E-Mail über einen SMTP-Server verwenden Sie die statische Methode `Send` der `Smtplib`-Klasse und übergeben idealerweise eine Instanz der Klasse `MailMessage`. Alternativ können Sie der `Send`-Methode auch die wichtigsten Daten der E-Mail direkt übergeben, dann fehlen Ihnen aber wichtige Möglichkeiten, wie z. B. das Anhängen von Dateien. Über eine `MailMessage`-Instanz können Sie die E-Mail sehr flexibel versenden. Listing 299 zeigt, wie Sie dies programmieren. Ich denke, die meisten Eigenschaften der `MailMessage`-Klasse erklären sich von selbst.

Interessant ist, dass Sie auch Mails im HTML-Format versenden und die Priorität einstellen können. Über die `Add`-Methode der `Attachments`-Eigenschaft können Sie auch Dateien anhängen. Dazu übergeben Sie eine neue Instanz der `MailAttachment`-Klasse, der Sie im Konstruktor den Dateinamen übergeben.

Bevor Sie die E-Mail dann versenden, sollten Sie auf jeden Fall in der Eigenschaft `SmtplibServer` den SMTP-Server festlegen: Diese Eigenschaft ist statisch, wenn also zuvor einmal ein SMTP-Server festgelegt wurde und Sie diesen nicht neu definieren, gilt die alte Einstellung. Wenn Sie den lokalen SMTP-Server angeben wollen (der ein Teil der Windows- Internet-Informationssdienstes ist), geben Sie einen Leerstring an.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die Mail versenden Sie dann über die `Send`-Methode. Hier sollten Sie eine Ausnahmebehandlung vorsehen, da das Senden u.U. nicht erlaubt ist (z.B. wenn der SMTP-Server eine Authentifizierung verlangt).

Das Beispiel in Listing 299 erfordert den Import der Namensräume `System` und `System.Web`.

```
// Nachricht erzeugen und initialisieren
MailMessage message = new MailMessage();
message.From = "juergen.bayer@jaddison-wesley.de";
message.To = "zaphod@galaxy.net";
message.Cc = "ford@galaxy.net";
message.Bcc = "trillian@galaxy.net";
message.Subject = "Party";
message.Body = "Hallo <b>Zaphod</b>, Lust auf eine Party im " +
    "<i>Restaurant am Ende der Galaxis</i>?";

// Mail-Format und Priorität festlegen
message.BodyFormat = MailFormat.Html;
message.Priority = MailPriority.High;

// Datei anhängen
string fileName = Path.Combine(Application.StartupPath, "dontpanic.gif");
message.Attachments.Add(new MailAttachment(fileName));

// SMTP-Server festlegen
SmtpMail.SmtpServer = "mailto:t-online.de";

// Mail senden
try
{
    SmtpMail.Send(message);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    if (ex.InnerException != null)
        Console.WriteLine(ex.InnerException.Message);
    if (ex.InnerException.InnerException != null)
        Console.WriteLine(ex.InnerException.InnerException.Message);
}
```

Listing 299: Senden einer Mail über `SmtpMail`

Das Versenden der E-Mail erfolgt intern über CDO (Collaborating Data Objects), eine COM-Komponente von Microsoft, die den Zugriff auf E-Mail-Funktionen, Outlook- und Exchange-Ordner ermöglicht. Das ist nicht unbedingt ein Vorteil. Problematisch ist z.B., dass die direkten Meldungen im Fehlerfall sehr »generisch« (also wenig aussagekräftig) sind. Ist der SMTP-Server z.B. nicht erreichbar, erhalten Sie die irreführende Meldung »Auf das Objekt 'CDO.Message' konnte nicht zugegriffen werden«. Die eigentliche Fehlermeldung versteckt sich in »der inneren Ausnahme der inneren Ausnahme der Ausnahme.« Deshalb fragt der Programmcode in Listing 299 diese inneren Ausnahmen ab, wenn eine Ausnahme eintritt.

Ein Problem der `SmtpMail`-Klasse ist, dass Sie keine Authentifizierungsdaten übergeben können. Erfordert der SMTP-Server einen Login, haben Sie keine Chance, die E-Mail zu versenden. Ausnahme: Wenn die Internetverbindung über den Provider läuft, der auch den SMTP-Server zur Verfügung stellt, dann erfolgt die Authentifizierung meist automatisch.

Als Lösung dieses Problems können Sie spezielle Mail-Komponenten wie die von *Quicksoft* (www.quicksoftcorp.com/landing/dotnet_mail.asp) oder *devmail.net* (www.devmail.net) verwenden, die meist gleich mehrere Möglichkeiten bieten, E-Mails zu verarbeiten. Die kostenfreie (»Free mode«-)Variante der Komponente von *devmail.net* bietet alle wichtigen Features und sogar die Authentifizierung gegenüber dem SMTP-Server. Die wesentlichen Nachteile gegenüber der Vollversion dieser Komponente sind, dass *devmail.net* im freien Modus die Größe des E-Mail-Textes auf 5 KB beschränkt, lediglich maximal zwei Anhänge mit maximal 50 KB Größe erlaubt und den Text »powered by devmail.net« an die E-Mail anhängt.

Sie können aber auch das relativ einfache SMTP-Protokoll über ein `TcpClient`-Objekt selbst implementieren. Informationen dazu finden Sie im RFC 2821 (<ftp://ftp.rfc-editor.org/in-notes/rfc2821.txt>). Problematisch dabei wird allerdings die Authentifizierung, da die verschiedenen Server unterschiedliche Verfahren einsetzen können. Grundlegende Informationen dazu finden Sie im RFC 2554 (<ftp://ftp.rfc-editor.org/in-notes/rfc2554.txt>).

170 E-Mails über MAPI/Outlook versenden

Viele Unternehmen bieten ihren Mitarbeitern E-Mail-Funktionalitäten nicht über einen SMTP-, POP3- und/oder IMAP-Server an, sondern betreiben einen Exchange-Server. Mails können Sie bei solchen Systemen nur über spezielle Microsoft-Komponenten versenden. Neben der COM-Komponente CDO (Collaborating Data Objects) können Sie dazu MAPI (Messaging API) verwenden. MAPI ist ein wesentlicher Teil des Windows-API zum Zugriff auf die Microsoft-Mail-Features (über Outlook und den Exchange Server). Leider ist die Programmierung sehr aufwändig. Ein von Thomas Scheidegger entwickeltes, hervorragendes Beispiel inklusive einer C#-Klasse zur Vereinfachung des MAPI-Zugriffs finden Sie auf der Website von *The Code Project* an der Adresse www.codeproject.com/csharp/simplemapidotnet.asp. Aber auch Microsoft liefert eine fertige Klassenbibliothek für den Zugriff. Sie können den Quellcode dieser Komponente über den Knowledge-Base-Artikel 315653 downloaden (support.microsoft.com/?kbid=315653).

Ich verwende in diesem Rezept die Klasse `Mapi` von Thomas Scheidegger, die als Public Domain veröffentlicht ist, und zeige lediglich, wie Sie eine einfache Mail über die dort enthaltenen Klassen verwenden. Schauen Sie sich auf jeden Fall auch das Beispiel an, das dem Quellcode beiliegt.

Kopieren Sie die Datei `MapiApi.cs` aus einem der beiden Beispiele (der Konsolen- oder der Windows-Anwendung) in den Projektordner Ihres Projekts. Um eine Mail zu versenden (was nur eine der Anwendungsmöglichkeiten dieser Klasse ist), erzeugen Sie eine Instanz der Klasse `Mapi`. Über die Methode `Logon` können Sie das Programm in MAPI einloggen (was aber eigentlich nur dann notwendig ist, wenn der Zugang geschützt ist). Wenn Sie einen spezifischen Benutzer einloggen wollen, übergeben Sie den Handle eines User-Token (als `IntPtr`). Dieses Handle können Sie über die API-Funktion `LogonUser` erzeugen. Informationen dazu

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Intern

Formu-
lare

Benut

Multi-
mediaAutom-
tionSons-
tiges

finden Sie im Rezept 211. `LogonUser` gibt, wie viele der Methoden der `Mapi`-Klasse, `false` zurück, wenn ein Fehler aufgetreten ist. Diesen (leider in vielen Fällen nicht sehr aussagekräftigen) Fehler können Sie über die Methode `Error` auslesen.

Nun können Sie über die Methode `AddRecip` Empfänger hinzufügen. Dazu übergeben Sie am ersten Argument die Adresse des Empfängers. Unklar dabei ist, dass der Name des ersten Arguments (`name`) eigentlich darauf hinweist, dass hier der Name des Empfängers übergeben wird. Das zweite Argument ist mit `addr` benannt, was suggeriert, dass hier die Adresse übergeben wird. Beim Test auf meinem System funktionierte aber keine andere Variante als die, die ich hier beschreibe. Das Problem wird wohl aber eher an MAPI liegen als an der .NET-Implementierung von Thomas Scheidegger.

Am dritten Argument können Sie mit `true` bestimmen, dass der Empfänger als CC hinzugefügt wird. Übergeben Sie `false`, wird der Empfänger normal hinzugefügt. Ein BCC ist scheinbar nicht möglich.

Über die `Attach`-Methode können Sie Dateien anfügen, indem Sie einfach den Dateinamen übergeben.

Die Nachricht können Sie schließlich über die `Send`-Methode senden. Am ersten Argument übergeben Sie den Betreff, am zweiten den Text der Nachricht. `Send` gibt `false` zurück, wenn beim Senden ein Fehler aufgetreten ist. Den Fehler können Sie wieder über die Methode `Error` auslesen.

Schließlich loggen Sie den eingeloggten Benutzer über `Logoff` wieder aus.

Das Beispielprogramm in Listing 300 benötigt den Import der Namensräume `System`, `System.IO`, `System.Windows.Forms` und `Win32Mapi`, eine Referenz auf die Assembly `System.Windows.Forms.dll`, und das Einbinden der Datei `MapiApi.cs`.

```
// Mapi-Instanz erzeugen
Mapi mapi = new Mapi();

// Logon in MAPI mit dem aktuellen Benutzer.
if(mapi.Logon(IntPtr.Zero) == false)
{
    Console.WriteLine("Der Login in MAPI ist fehlgeschlagen: " + mapi.Error());
    return;
}

// Empfänger hinzufügen
string mailAddress = "zaphod@galaxy.com";
mapi.AddRecip(mailAddress, null, false);

// Datei anfügen
string fileName = Path.Combine(Application.StartupPath, "dontpanic.gif");
mapi.Attach(fileName);

// Mail senden
```

Listing 300: Senden einer Mail über MAPI und die Klasse `Mapi` von Thomas Scheidegger


```

string subject = "Party";
string message = "Hallo Zaphod, hast du Lust auf eine Party im Restaurant " +
    "am Ende der Galaxis?";
if (mapi.Send(subject, message))
    Console.WriteLine("E-Mail erfolgreich versendet");
else
    Console.WriteLine("Das Senden ist fehlgeschlagen: {0}", mapi.Error());

// Aus MAPI ausloggen
mapi.Logoff();

```

Listing 300: Senden einer Mail über MAPI und die Klasse Mapi von Thomas Scheidegger (Forts.)

Das Senden der E-Mail erfolgt dann normalerweise über Outlook. Dabei erscheint u.U. ein Sicherheitshinweis (Abbildung 116).

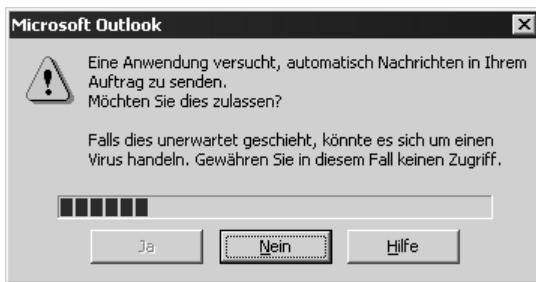


Abbildung 116: Der Sicherheitshinweis von Outlook ab Version 2000, SP 2

Dieser Sicherheitshinweis, bei dem der Ja-Schalter für fünf Sekunden deaktiviert ist, ist ab Outlook 2000, Service Pack 2 enthalten. Microsoft will damit verhindern, dass Virenprogramme unbemerkt E-Mails versenden können. Sie können den Dialog leider nicht unterbinden.

Sie können jedoch an der Adresse www.express-soft.com/mailmate/clickyes.html das Tool *Express ClickYes* downloaden, das die Betätigung des Ja-Schalters automatisch vornimmt. Nach der Installation finden Sie dieses Tool im Infobereich der Taskleiste. Wählen Sie im Kontextmenü dieses Eintrags die Option *START SUSPENDED* ab und schalten Sie die Option *START ON LOGON* ein, damit *Express ClickYes* beim nächsten Windows-Start automatisch ausgeführt wird. Nach der Installation müssen Sie einmal den Eintrag *RESUME* im Kontextmenü anklicken, damit *Express ClickYes* ohne einen Neustart arbeitet.

171 Browser starten

Den Default-Browser des Systems können Sie sehr einfach über die statische `Start`-Methode der `Process`-Klasse (aus dem Namensraum `System.Diagnostics`) starten. Als »Dateiname« geben Sie am ersten Argument die komplette Webadresse an (inklusive `http://`, `ftp://`, `file://` etc.). Wenn Sie einen spezifischen Browser starten wollen, erzeugen Sie ein neues `Process`-

Objekt, definieren in dessen `StartInfo`-Eigenschaft in der Eigenschaft `FileName` den Dateinamen und geben in der Eigenschaft `Arguments` die Webadresse an.

```
// Standard-Browser öffnen
Process.Start("http://www.google.de");

// Internet Explorer öffnen
Process process = new Process();
process.StartInfo.FileName = "iexplore.exe";
process.StartInfo.Arguments = "http://www.addison-wesley.de";
process.Start();
```

Listing 301: Starten des Standard- und eines speziellen Browsers

Eine andere Möglichkeit, die ich hier nicht beschreibe, ist übrigens das Fernsteuern des Internet Explorers ähnlich dem Fernsteuern von Word und Excel, das ich im Kapitel COM-Interop mit Office beschreibe. Das Ganze ist aber nicht allzu kompliziert, da Sie beim Internet Explorer auf dieselben Methoden und Eigenschaften zugreifen wie beim Webbrowser-Steuerelement, das ich im folgenden Rezept beschreibe.

172 Webseiten (HTML-Dokumente) in der Anwendung darstellen

Leider bietet das .NET Framework selbst keine Möglichkeit, Webseiten (HTML-Dokumente) in einer .NET-Windowsanwendung anzuzeigen (in einer Webanwendung geht das natürlich ...).

In einer Windowsanwendung müssen Sie auf ein ActiveX-Steuerelement zurückgreifen. Mit dem Internet Explorer ist z. B. ein solches installiert. Sie könnten aber auch z. B. das Steuerelement des Mozilla-Browsers verwenden, sofern dieser installiert ist.

Das ActiveX-Steuerelement des Internet Explorers versteckt sich in der Datei `shdocvw.dll` im Systemordner von Windows. In Visual Studio .NET können Sie dieses Steuerelement einbinden, indem Sie das Windows-Forms-Register der Toolbar anpassen. Klicken Sie dazu mit der rechten Maustaste auf die Toolbar und wählen Sie den Befehl `TOOLBAR ANPASSEN`. Im erscheinenden Dialog suchen Sie den Eintrag `MICROSOFT WEBBROWSER` und binden diesen ein.

Visual Studio .NET erzeugt für das Steuerelement die zwei Wrapper-Assemblies `AxSHDocVw.dll` und `SHDocVw.dll`, kopiert die Dateien in das Anwendungsverzeichnis und referenziert diese. `AxSHDocVw.dll` enthält das Steuerelement, `SHDocVw.dll` spezielle weitere Typen, die für die erweiterte Arbeit mit dem Steuerelement notwendig sind.

Wenn Sie Visual Studio nicht besitzen, erzeugen Sie die Wrapper-Assemblies über das Tool `aximp.exe` (ActiveX Import):

```
aximp c:\windows\system32\shdocvw.dll
```

Die Wrapper-Klasse für das Steuerelement befindet sich dann im Namensraum `AxSHDocVw` und heißt `AxWebBrowser`.

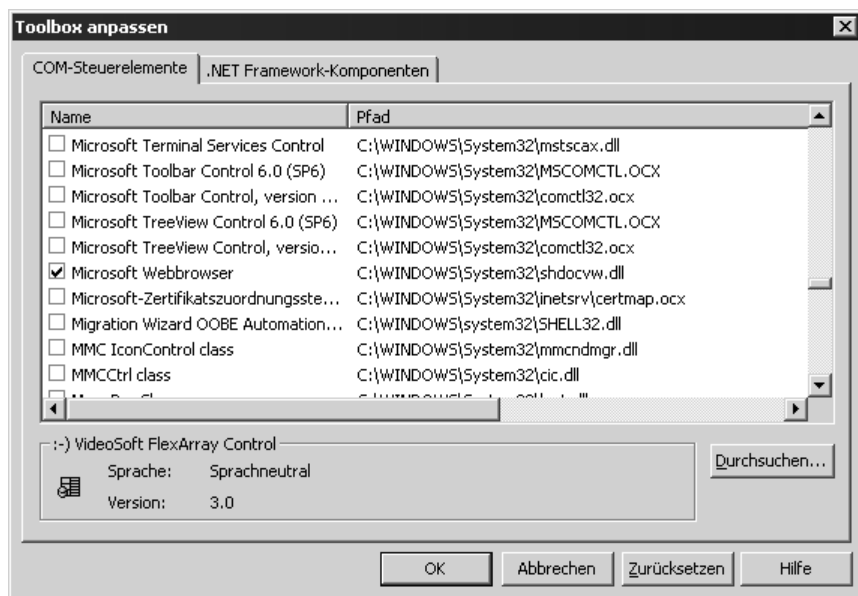


Abbildung 117: Einbinden des Microsoft Webbrowser-Steuerelements

Wenn Sie das Editieren des HTML-Dokuments ermöglichen bzw. auf das DHTML-Objektmodell zugreifen wollen, müssen Sie noch die Typbibliothek `mshtml.tlb` (im Windows-Systemordner) importieren, indem Sie in Visual Studio einen COM-Verweis auf diese Datei hinzufügen. Ohne Visual Studio verwenden Sie das Tool `tlbimp.exe` zur Erzeugung der Wrapper-Assembly:

```
tlbimp mshtml.tlb
```

Das Erzeugen dauert einige Zeit, weil sehr viele Typen importiert werden.

Das Webbrowser-Steuerelement können Sie dann wie ein normales Steuerelement auf das Formular ziehen und bearbeiten bzw. (ohne Visual Studio .NET) im Quellcode erzeugen und initialisieren. Die Arbeit mit dem Steuerelement ist grundsätzlich einfach, weil es wie alle anderen Steuerelemente auch Eigenschaften, Methoden und Ereignisse besitzt. Einige Aufgaben sind jedoch leider nur auf eine recht komplexe Weise zu programmieren. Dabei sollten Sie immer im Auge behalten, dass dieses Steuerelement die volle Funktionalität des Internet Explorers anbietet und Ihnen darüber hinaus über seine Elemente und im Besonderen seine Ereignisse eine Erweiterung dieser Funktionalität ermöglicht.

Informationen zur Arbeit mit dem Steuerelement und eine Dokumentation finden Sie im Artikel »Reusing the WebBrowser Control« an der Adresse msdn.microsoft.com/workshop/browser/webbrowser/webbrowser.asp.

Die Listings ab Listing 302 zeigen die wichtigsten Features des Steuerelements. Das Beispiel ist in einer Windows-Anwendung implementiert, die in Abbildung 118 dargestellt wird.



Abbildung 118: Das Webbrowser-Beispielprogramm

Über die `Navigate2`-Methode, die einige optionale Argumente besitzt, können Sie zu einer Website navigieren. Die URL übergeben Sie dazu am ersten Argument. An den folgenden Argumenten können Sie spezielle Flags übergeben (z.B. um zu bestimmen, dass die Adresse dem Verlauf hinzugefügt oder das Dokument in einem neuen Fenster angezeigt wird), den Ziel-Frame für Webseiten mit Framesets bestimmen und eigene POST- und HTTP-Headerdaten übergeben. Für die meisten Fälle benötigen Sie diese Argumente nicht und können hier einfach `Missing.Value` übergeben.

Die Navigation übernehmen im Beispiel das `Click`-Ereignis des Öffnen-Schalters und das `KeyPress`-Ereignis der Textbox (`urlTextBox`).

Das Programm benötigt neben dem für Windows.Forms-Anwendungen notwendigen Referenzen und Importen und den Wrapper-Assemblies für die Dateien `shdocvw.dll` und `mshtml.tlb` den Import des Namensraums `System.Reflection`.

```
private void openButton_Click(object sender, System.EventArgs e)
{
    // Navigieren
    object missing = Missing.Value;
    object url = urlTextBox.Text;
    browser.Navigate2(ref url, ref missing, ref missing, ref missing,
        ref missing);
}

private void urlTextBox_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
    if (e.KeyChar == '\r')
    {
        e.Handled = true; // Warnton verhindern

        // Navigieren
        object missing = Missing.Value;
        object url = urlTextBox.Text;
        browser.Navigate2(ref url, ref missing, ref missing, ref missing,
            ref missing);
    }
}
```

Listing 302: Navigieren zu einer Webseite

Die Schalter zum Wechseln zur Startseite (*Home*), zur vorherigen Seite (*Zurück*) und zur nächsten Seite (*Vor*) und zum Stoppen des aktuellen Vorgangs rufen die Methoden `GoHome`, `GoBack`, `GoForward` bzw. `Stop` auf:

```
private void homeButton_Click(object sender, System.EventArgs e)
{
    browser.GoHome();
}

private void backButton_Click(object sender, System.EventArgs e)
{
    browser.GoBack();
}

private void forwardButton_Click(object sender, System.EventArgs e)
{
    browser.GoForward();
}

private void stopButton_Click(object sender, System.EventArgs e)
{
    browser.Stop();
}
```

Listing 303: Wechseln zur Startseite, zur vorherigen und zur nächsten Seite

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Eine Ausnahmebehandlung habe ich hier nicht vorgesehen, obwohl der Aufruf der `GoBack`- und der `GoForward`-Methode eine Ausnahme erzeugt, wenn keine vorherige bzw. nächste Seite verfügbar ist. In Listing 308 sorgt das `CommandStateChange`-Ereignis aber dafür, dass diese Schalter nur dann verfügbar sind, wenn die entsprechende Möglichkeit auch besteht.

Beim Schalter zum erneuten Laden der Webseite wird es ein wenig komplex. Das erneute Laden erreichen Sie nämlich über die Methode `ExecWB`, die zur Ausführung spezieller Befehle vorgesehen ist. Am ersten Argument übergeben Sie eine Konstante der Aufzählung `OLECMDID`, die in der Assembly `SHDocVw` definiert ist. Die Konstante `OLECMDID_REFRESH` steht für den Reload-Befehl. Am zweiten Argument können Sie für spezifische Befehle mit den Konstanten der Aufzählung `SHDocVw.OLECMDEXECHOPT` bestimmen, ob der Benutzer aufgefordert wird, die Aktion zu bestätigen oder ob eine Hilfe angezeigt werden soll. Das Beispiel übergibt hier die Konstante `OLECMDEXECHOPT_DODEFAULT` um die Defaulteinstellung zu verwenden. An den letzten beiden Argumenten können Sie für spezielle Befehle weitere Argumente übergeben. Das dritte Argument erwartet einen Eingangs-Parameter, am vierten erhalten Sie dann einen Wert zurück. Für das erneute Laden einer Webseite übergeben Sie hier `Missing.Value`:

```
private void reloadButton_Click(object sender, System.EventArgs e)
{
    // Aufruf des Refresh-Befehls über ExecWB
    object missing = Missing.Value;
    browser.ExecWB(SHDocVw.OLECMDID.OLECMDID_REFRESH,
        SHDocVw.OLECMDEXECHOPT.OLECMDEXECHOPT_DODEFAULT,
        ref missing, ref missing);
}
```

Listing 304: Erneutes Laden der aktuellen Webseite

Der Schalter zum Bearbeiten soll nur demonstrieren, dass Sie das geladene Dokument auch bearbeiten können. Dazu referenzieren Sie die `IHTMLDocument2`-Schnittstelle des Dokuments, das Sie über die `Document`-Eigenschaft des Webbrowser-Steuerelements erreichen. Diese Schnittstelle ist in der Typbibliothek `mshtml.tlb` definiert, die Sie über den Namensraum `mshtml` zur Verfügung haben.

Über die `IHTMLDocument2`-Referenz können Sie das Dokument dann bearbeiten. Diese Schnittstelle gibt Ihnen Zugriff auf alle Elemente des HTML-Dokuments und ermöglicht neben dem Lesen auch das Verändern des Dokuments. Auf dieses recht komplexe Thema gehe ich hier aber nicht ein. Das Beispiel schaltet über die Eigenschaft `designMode` einfach den Design-Modus ein und aus, sodass der Anwender das Dokument direkt bearbeiten kann.

```
private void editButton_Click(object sender, System.EventArgs e)
{
    // Referenzieren der IHTMLDocument2-Schnittstelle des Dokuments
    mshtml.IHTMLDocument2 htmlDocument =
        (mshtml.IHTMLDocument2)browser.Document;
```

Listing 305: Ein- und Ausschalten des Design-Modus

```

    if (htmlDocument != null)
    {
        if (editButton.Text == "Bearbeiten")
        {
            // Edit-Modus einschalten
            editButton.Text = "Beenden";
            htmlDocument.designMode = "On";
        }
        else
        {
            // Edit-Modus ausschalten
            editButton.Text = "Bearbeiten";
            htmlDocument.designMode = "Off";
        }
    }
}

```

Listing 305: Ein- und Ausschalten des Design-Modus (Forts.)

Das Drucken und das Speichern des HTML-Dokuments geschieht ähnlich dem erneuten Laden über die `ExecWB`-Methode. Dazu übergeben Sie am ersten Argument die Konstante `OLECMDID_PRINT` bzw. `OLECMDID_SAVEAS`. Zum Speichern können Sie auch die Konstante `OLECMDID_SAVE` verwenden, dann muss das Dokument aber zuvor bereits über `OLECMDID_SAVEAS` gespeichert worden sein. `OLECMDID_SAVEAS` bewirkt auch, dass der Anwender den Dateinamen in einem Speichern-Unter-Dialog eingeben kann.

```

private void printButton_Click(object sender, System.EventArgs e)
{
    // Aufruf des Drucken-Befehls über ExecWB
    object missing = Missing.Value;
    browser.ExecWB(SHDocVw.OLECMDID.OLECMDID_PRINT,
        SHDocVw.OLECMDEXECOPT.OLECMDEXECOPT_PROMPTUSER,
        ref missing, ref missing);
}

private void saveButton_Click(object sender, System.EventArgs e)
{
    // Aufruf des Speichern-Befehls über ExecWB
    object missing = Missing.Value;
    browser.ExecWB(SHDocVw.OLECMDID.OLECMDID_SAVEAS,
        SHDocVw.OLECMDEXECOPT.OLECMDEXECOPT_DODEFAULT,
        ref missing, ref missing);
}

```

Listing 306: Drucken und Speichern der HTML-Seite

Über verschiedene Ereignisse können Sie auf Änderungen des Status reagieren. Das Ereignis `NavigateComplete2` wird z.B. aufgerufen, wenn das Laden eines Dokuments beendet ist, `NavigateError` wird aufgerufen, wenn beim Navigieren ein Fehler auftritt. Ich setze im Beispiel

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Intern

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

zunächst das Ereignis `TitleChange` ein um die Titelleiste des Formulars an den Titel des HTML-Dokuments anzupassen:

```
private void browser_TitleChange(object sender,
    AxSHDocVw.DWebBrowserEvents2_TitleChangeEvent e)
{
    this.Text = e.text + " - " + Application.ProductName;
}
```

Listing 307: Anpassen der Titelleiste des Formulars an den Titel des aktuellen Dokuments

Das Ereignis `CommandStateChange` wird aufgerufen, wenn der Status eines Befehls (z.B. dessen Verfügbarkeit) sich ändert. Dieses Ereignis können Sie sehr gut nutzen, um die Befehle Ihrer Anwendung entsprechend zu aktivieren oder zu deaktivieren. Ich verzichte hier auf eine weitere Erläuterung, da ich denke, dass die Kommentare in Listing 308 alles Notwendige aussagen.

```
private void browser_CommandStateChange(object sender,
    AxSHDocVw.DWebBrowserEvents2_CommandStateChangeEvent e)
{
    // Änderung des Befehls-Status
    switch (e.command)
    {
        case (int)SHDocVw.CommandStateChangeConstants.CSC_NAVIGATEBACK:
            // Der Status des Zurück-Befehls hat sich geändert
            backButton.Enabled = e.enable;
            break;

        case (int)SHDocVw.CommandStateChangeConstants.CSC_NAVIGATEFORWARD:
            // Der Status des Vorwärts-Befehls hat sich geändert
            forwardButton.Enabled = e.enable;
            break;

        case (int)SHDocVw.CommandStateChangeConstants.CSC_UPDATECOMMANDS:
            // Der Status eines Befehls, der sich auf ein geladenes Dokument
            // bezieht, hat sich geändert

            // Den int-Wert für die Flags ermitteln, die bestimmen, dass
            // ein Befehl unterstützt wird und aktiviert ist
            int enabled = Convert.ToInt32(SHDocVw.OLECMDIF.OLECMDIF_SUPPORTED)
                + Convert.ToInt32(SHDocVw.OLECMDIF.OLECMDIF_ENABLED);

            // Den Status der einzelnen Befehle abfragen und mit dem int-Wert für
            // den unterstützten, aktivierten Befehl vergleichen
            reloadButton.Enabled = (enabled ==
                (int)(browser.QueryStatusWB(SHDocVw.OLECMDID.OLECMDID_REFRESH)));
            saveButton.Enabled = (enabled ==
                (int)(browser.QueryStatusWB(SHDocVw.OLECMDID.OLECMDID_SAVEAS)));
```

Listing 308: Definieren des Befehlsstatus über das CommandStateChange-Ereignis


```

        printButton.Enabled = (enabled ==
            (int)(browser.QueryStatusWB(SHDocVw.OLECMDID.OLECMDID_PRINT)));
        editButton.Enabled = (enabled ==
            (int)(browser.QueryStatusWB(SHDocVw.OLECMDID.OLECMDID_SAVEAS)));
        printButton.Enabled = (enabled ==
            (int)(browser.QueryStatusWB(SHDocVw.OLECMDID.OLECMDID_PRINT)));
        // Der Status des Stop-Buttons muss über die Busy-Eigenschaft
        // abgefragt werden, damit dieser korrekt ausgewertet wird
        stopButton.Enabled = browser.Busy;
        break;
    }
}

```

Listing 308: Definieren des Befehlsstatus über das CommandStateChange-Ereignis (Forts.)

Und damit ist der einfache Webbrowser fertig. Das Programm lässt sich aber natürlich über die vielfältigen Fähigkeiten des Webbrowser-Steuerelements noch erweitern.

173 Dateien downloaden

Dateien können Sie aus dem Internet sehr einfach über ein `WebClient`-Objekt (aus dem Namensraum `System.Net`) herunterladen. Die `DownloadFile`-Methode ermöglicht den einfachen Download einer Datei und das direkte Speichern. Dummerweise überschreibt diese Methode vorhandene Dateien ohne Warnung und erzeugt für den Fall, dass die Datei nicht geschrieben werden kann (weil eine zu überschreibende Datei z.B. schreibgeschützt ist), keine spezielle `IOException`, sondern eine Ausnahme vom Typ `WebException` mit der Fehlermeldung »Während einer `WebClient`-Anforderung ist ein Ausnahmefehler aufgetreten«. Außerdem besitzen Sie bei der Verwendung dieser Methode keine Möglichkeit, einen Fortschritt des Download anzuzeigen.

Ich gehe in diesem Rezept deshalb einen flexibleren Weg. Sie können die Daten auch über ein `WebRequest`-Objekt anfordern. Über die `GetResponse`-Methode eines solchen Objekts erhalten Sie die Antwort des Servers in Form eines `WebResponse`-Objekts. Dessen `GetResponseStream`-Methode wiederum liefert einen Stream, aus dem Sie die Daten einlesen können.

Ähnliches ist übrigens auch über die `OpenRead`-Methode eines `WebClient`-Objekts möglich. `WebRequest`- und `WebResponse`-Objekte liefern aber erweiterte Möglichkeiten wie eine asynchrone Anforderung, asynchrones Lesen und das Ermitteln der Länge der Daten.

Die Klasse `WebDownload`, die ich im Folgenden beschreibe, soll möglichst flexibel arbeiten. Sie ermöglicht deshalb einen synchronen und einen asynchronen Download einer Datei aus dem Internet in einen beliebigen Stream. Zum Kompilieren dieser Klasse müssen Sie die Namensräume `System`, `System.Net`, `System.IO` und `System.Threading` importieren.

`WebDownload` enthält zunächst drei Delegates. Der Delegate `DownloadProgress` wird beim Download aufgerufen, während Daten gelesen werden. Dieser Delegate bekommt neben Informationen über die aktuell eingelesenen Bytes und die Gesamtgröße auch eine Information über den Status in Form eines Werts der Aufzählung `DownloadState` übergeben. Der Delegate `DownloadEnd` wird am Ende des Downloads aufgerufen. Das Argument `destStream`,

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Intern

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

das diesem Delegate übergeben wird, referenziert den Stream, der beim Aufruf einer der Download-Methoden übergeben wurde. Über diese Referenz können Sie den Stream schließen oder weiterverarbeiten, was besonders beim asynchronen Download wichtig ist. Der Delegate `DownloadError` wird aufgerufen, wenn beim Download Fehler auftreten. Dieser Delegate ist ebenfalls beim asynchronen Download wichtig.

```
public class WebDownload
{
    /* Aufzählung und Delegates für den Fortschritt und das Ende der
     * Download-Methode und für Fehlermeldungen */
    public enum DownloadState
    {
        OpeningConnection,
        ReadingData,
        DataReady
    }
    public delegate void DownloadProgress(DownloadState downloadState,
        int currentBytes, long totalBytes);
    public delegate void DownloadEnd(Stream destStream);
    public delegate void DownloadError(string message);
}
```

Listing 309: Aufzählung und Delegates für den Fortschritt und das Ende der Download-Methode

Nun habe ich aus Gründen, die ich weiter unten bei der Erläuterung des asynchronen Downloads beschreibe, eine privaten Klasse `Download` implementiert, deren `PerformDownload`-Methode den Download ausführt. Zur Ausführung des Downloads erzeugen die Download-Methoden von `WebDownload` eine `Download`-Instanz und rufen die `PerformDownload`-Methode auf.

Im Konstruktor der `Download`-Klasse werden die URL der herunterzuladenden Datei, eine Referenz auf einen Stream, in den die Methode die Daten schreiben soll, eine Blockgröße, die beim blockweisen Einlesen der Daten berücksichtigt wird, und je eine Referenz auf die bereits genannten Delegates übergeben.

`PerformDownload` erzeugt zunächst eine `WebRequest`-Instanz für die Anforderung der Daten und ermittelt das dazu gehörende `WebResponse`-Objekt. Die Gesamtlänge der Daten wird dann aus der Eigenschaft `ContentLength` dieses Objekts ausgelesen. Diese Länge wird übrigens im Response-Header gesendet, der den Daten vorangestellt ist, und steht deswegen bereits zur Verfügung, während die Daten noch auf dem Weg sind.

Dann liest `PerformDownload` in einer Schleife den Response-Stream ein, der über die `GetResponse`-Methode ermittelt wird. Beim Einlesen wird die am Argument `blockSize` übergebene Blockgröße berücksichtigt um den Arbeitsspeicher nicht mit zu großen Dateien zu belasten. Innerhalb der Schleife ruft `PerformDownload` den Fortschritts-Delegate auf, damit der Aufrufer eine Info über den Fortschritt erhält.

Tritt während des Downloads eine Ausnahme ein, ruft `PerformDownload` den Delegate für die Fehlermeldung auf, falls einer übergeben wurde. Im anderen Fall wird die Ausnahme weitergeworfen. Die Verwendung des Delegates ist im Besonderen für den asynchronen Aufruf wichtig, da eine Ausnahme in einem Thread nicht von der Anwendung behandelt werden kann.

```
private class Download
{
    public string Url;
    public int BlockSize;
    public DownloadProgress DownloadProgress;
    public DownloadEnd DownloadEnd;
    public DownloadError DownloadError;
    public Stream DestStream;

    /* Konstruktor */
    public Download(string url, Stream destStream, int blockSize,
        DownloadProgress downloadProgress, DownloadEnd downloadEnd,
        DownloadError downloadError)
    {
        this.Url = url;
        this.BlockSize = blockSize;
        this.DownloadProgress = downloadProgress;
        this.DownloadEnd = downloadEnd;
        this.DestStream = destStream;
        this.DownloadError = downloadError ;
    }

    /* Methode, die den Download ausführt */
    public void PerformDownload()
    {
        WebRequest request = null;
        WebResponse response = null;
        Stream responseStream = null;
        try
        {
            // WebRequest-Instanz für den Download erzeugen, die Antwort
            // anfordern und die Länge der Daten ermitteln
            if (this.DownloadProgress != null)
                this.DownloadProgress(DownloadState.OpeningConnection,
                    0, 0);
            request = WebRequest.Create(this.Url);
            response = request.GetResponse();
            long fileSize = response.ContentLength;

            // Den Antwort-Stream ermitteln, diesen blockweise lesen und
            // in den Zielstream schreiben
            responseStream = response.GetResponseStream();
            int bytesRead = 0;
            int totalBytesRead = 0;
            byte[] buffer = new byte[this.BlockSize];
            do
            {
                bytesRead = responseStream.Read(buffer, 0, this.BlockSize);
                totalBytesRead += bytesRead;
                this.DestStream.Write(buffer, 0, bytesRead);
            }
        }
        catch { }
    }
}
```

Listing 310: Klasse mit einer Methode zum Download einer Datei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        // Fortschritt melden
        if (this.DownloadProgress != null)
            this.DownloadProgress(DownloadState.ReadingData,
                totalBytesRead, fileSize);
    } while (bytesRead > 0);
}
catch (Exception ex)
{
    if (this.DownloadError != null)
        // Delegate für Fehlermeldungen aufrufen
        this.DownloadError(ex.Message);
    else
        // Ausnahme weiterwerfen
        throw ex;
}
finally
{
    try
    {
        // Den Response-Stream und das WebResponse-Objekt schließen
        responseStream.Close();
        response.Close();
    }
    catch {}

    // Den Delegate für das Ende des Downloads aufrufen
    if (this.DownloadEnd != null)
        this.DownloadEnd(this.DestStream);
}
}
}

```

Listing 310: Klasse mit einer Methode zum Download einer Datei (Forts.)

Die Methode `DownloadSync` führt einen synchronen Download durch, indem sie die `PerformDownload-Methode einer neuen Download-Instanz aufruft:`

```

public void DownloadSync(string url, Stream destStream, int blockSize,
    DownloadProgress downloadProgress, DownloadEnd downloadEnd,
    DownloadError downloadError)
{
    // Download-Objekt erzeugen und initialisieren
    Download download = new Download(url, destStream, blockSize,
        downloadProgress, downloadEnd, downloadError);

    // Download synchron starten
    download.PerformDownload();
}

```

Listing 311: Synchroner Download einer Datei

DownloadSync besitzt einen Vor- und einen Nachteil: Wegen des synchronen Aufrufs ist der aufrufende Thread so lange blockiert, bis die Methode zurückkehrt (was schon der Nachteil ist). Der Vorteil dabei ist, dass das Programm sicher sein kann, dass nach dem Aufruf alle Daten verarbeitet wurden und es somit den Stream direkt nach dem Aufruf verarbeiten kann.

Da das Blockieren des aktuellen Thread aber in Anwendungen, die größere Dateien herunterladen, nicht allzu schön ist, habe ich die weitere Methode `DownloadAsync` zum asynchronen Download einer Datei entwickelt. Dabei musste ich mich unter den zwei Möglichkeiten entscheiden, dass dazu eine asynchrone Anforderung (über `BeginGetResponse`) und ein asynchrones Lesen (über `BeginRead`) oder die Programmierung in einem Thread möglich ist. Ich habe mich für die Thread-Variante entschieden, weil diese deutlich einfacher zu programmieren und zu verstehen ist (und weniger Platz im Buch in Anspruch nimmt). Die Variante mit den asynchronen Aufrufen finden Sie neben diesem Beispiel auf der Buch-CD.

Die Thread-Variante erzeugt für den gesamten Vorgang einen Thread. Dabei musste ich das Problem lösen, dass diesem Thread Daten wie z.B. die URL, die Blockgröße und der Ziel-Stream übergeben werden müssen. Die Signatur einer Thread-Methode ist aber festgelegt und erlaubt keine Argumente.

Und hier kommt die Klasse `Download` ins Spiel. `DownloadAsync` erzeugt eine Instanz dieser Klasse, initialisiert diese und übergibt deren Methode `PerformDownload` an den `ThreadStart-Delegate`. Da diese Methode der Signatur des Delegates entspricht, ist der Aufruf möglich. `PerformDownload` kann auf alle Elemente der Instanz zugreifen und voilà: So können Sie beliebige Daten an einen Thread übergeben.

```
public void DownloadAsync(string url, Stream destStream, int blockSize,
    DownloadProgress downloadProgress, DownloadEnd downloadEnd,
    DownloadError downloadError)
{
    // Download-Objekt erzeugen und initialisieren
    Download download = new Download(url, destStream, blockSize,
        downloadProgress, downloadEnd, downloadError);

    // Thread für den Download starten
    Thread downloadThread = new Thread(
        new ThreadStart(download.PerformDownload));
    downloadThread.Start();
}
}
```

Listing 312: Methode zum asynchronen Download einer Datei

Die Anwendung der `WebDownload`-Klasse ist nun recht einfach. Ich beschreibe diese kurz am Beispiel einer Windowsanwendung (Abbildung 119).

In der Formulkasse habe ich zunächst die Methoden für die Delegates implementiert. Die Methode `DownloadProgressHandler` fragt den jeweiligen Status ab und aktualisiert mit den übergebenen Daten die Info-Textboxen und das `ProgressBar`-Steuerelement des Formulars.

Die Initialisierung wird in einem `lock`-Block vorgenommen um zu verhindern, dass mehrere Threads gleichzeitig auf das Formular zugreifen können. Die Methode `DownloadEndHandler` schließt den Stream, der in den Methoden für das Click-Ereignis der Download-Schalter geöffnet wurde. Die Methode `DownloadErrorHandler` gibt einfach eine Fehlermeldung aus.

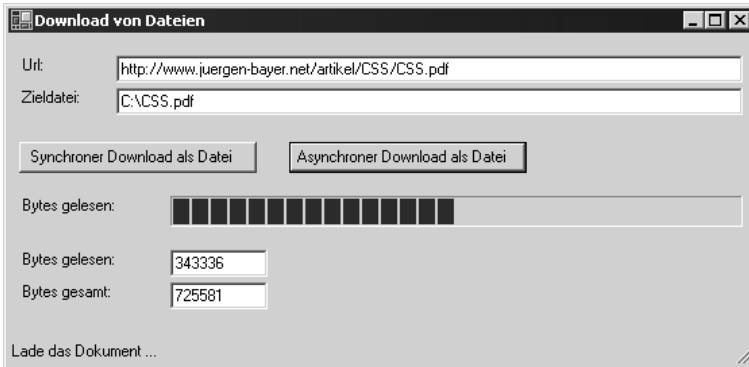


Abbildung 119: Beispielanwendung der `WebDownload`-Klasse

```

/* Methode für den Fortschritts-Delegate */
public void DownloadProgressHandler(WebDownload.DownloadState downloadState,
    int currentBytes, long totalBytes)
{
    lock (this)
    {
        switch (downloadState)
        {
            case WebDownload.DownloadState.OpeningConnection:
                this.progressBar.Value = 0;
                this.statusBar.Text = "Öffne Verbindung ...";
                this.bytesReadTextBox.Text = "0";
                this.bytesTotalTextBox.Text = "0";
                break;

            case WebDownload.DownloadState.ReadingData:
                this.statusBar.Text = "Lade das Dokument ...";
                int progress = (int)((currentBytes / (double)totalBytes) * 100);
                this.progressBar.Value = progress;
                this.bytesReadTextBox.Text = currentBytes.ToString();
                this.bytesTotalTextBox.Text = totalBytes.ToString();
                break;

            case WebDownload.DownloadState.DataReady:
                this.progressBar.Value = 0;
                this.statusBar.Text = "Fertig";
                break;
        }
    }
}

```

Listing 313: Methoden für die Delegates der `WebDownload`-Klasse

```
    }  
}  
  
/* Methode für den Download-Ende-Delegate */  
public void DownloadEndHandler(Stream destStream)  
{  
    // Die Datei schließen  
    destStream.Close();  
  
    // Info ausgeben  
    this.statusBar.Text = "Fertig ...";  
    this.progressBar.Value = 0;  
}  
  
/* Methode für den Fehler-Delegate */  
public void DownloadErrorHandler(string error)  
{  
    MessageBox.Show(error, Application.ProductName, MessageBoxButtons.OK,  
        MessageBoxIcon.Error);  
}
```

Listing 313: Methoden für die Delegates der WebDownload-Klasse (Forts.)

Die Methoden für Click-Ereignis der Download-Schalter erzeugen jeweils eine Instanz der Klasse `WebDownload` und rufen den synchronen bzw. asynchronen Download auf. Da der Programm-Quelltext bis auf den Methodennamen identisch ist, zeige ich lediglich die Methode zum asynchronen Download:

```
private void asyncDownloadAsFile_Click(object sender, System.EventArgs e)  
{  
    FileStream fileStream = null;  
    try  
    {  
        // FileStream für die Datei erzeugen  
        fileStream = new FileStream(this.destFileNameTextBox.Text,  
            FileMode.Create, FileAccess.Write);  
    }  
    catch (Exception ex)  
    {  
        MessageBox.Show(ex.Message, Application.ProductName,  
            MessageBoxButtons.OK, MessageBoxIcon.Error);  
  
        // FileStream schließen  
        try {fileStream.Close();}  
        catch {}  
  
        return;  
    }  
}
```

Listing 314: Asynchroner Download

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// Datei asynchron herunterladen
try
{
    WebDownload webDownload = new WebDownload();
    webDownload.DownloadAsync(this.urlTextBox.Text, fileStream, 1024,
        new WebDownload.DownloadProgress(this.DownloadProgressHandler),
        new WebDownload.DownloadEnd(this.DownloadEndHandler),
        new WebDownload.DownloadError(this.DownloadErrorHandler));
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
```

Listing 314: Asynchroner Download (Forts.)

Der asynchrone Download funktionierte in meinen Test mit einer Ausnahme einwandfrei. Es war sogar problemlos möglich, mehrere Downloads parallel auszuführen, wenigstens so lange der Webzugriff nicht durch eine Firewall oder Ähnliches verhindert wurde. Blockierte eine Firewall den Zugriff der Anwendung beim ersten Downloadversuch, schlug der quasi-parallel gestartete zweite Versuch kurz danach mit der Meldung fehl, dass die zugrunde liegende Verbindung geschlossen wurde. Das galt leider auch dann, wenn der Zugriff der Anwendung kurz nach der Blockierung gewährt wurde, da der zweite Thread sehr schnell nach dem ersten ausgeführt wurde. Den wirklichen Grund für diesen Fehler habe ich nicht ausfindig gemacht, da das Blockieren eines einzelnen asynchronen Aufrufs durch die Firewall kein Problem darstellte, sofern der Zugriff innerhalb des Timeout der Verbindung freigegeben wurde.

Beim gleichzeitigen Ausführen mehrerer asynchroner Downloads müssen Sie natürlich aufpassen, dass die einzelnen Threads über die Delegates nicht gleichzeitig auf Ressourcen zugreifen, und gegebenenfalls `lock`-Blöcke einsetzen.

Die Klasse `WebDownload` kann übrigens noch erweitert werden. Ein in diesem Rezept nicht gelöstes Problem ist, dass der Abbruch der gestarteten Threads nicht möglich ist. Sogar dann, wenn die Anwendung beendet wird, laufen die gestarteten Threads noch weiter (und laden die Dateien im Hintergrund bis zum Ende weiter, was ja eigentlich gar nicht so schlecht ist). Aber das überlasse ich ganz Ihnen ...

Formulare und Steuerelemente

174 Formulare ohne Titelleiste

Eine in Newsgroups häufiger auftauchende Frage ist, wie ein Formular ohne Titelleiste erstellt wird. Eine mögliche Antwort wäre, einfach die Eigenschaft `FormBorderStyle` auf `None` zu setzen. Dann besitzt das Formular aber keinen Rahmen und sieht etwas unschön aus. Die Lösung ist aber ganz einfach: Setzen Sie die Eigenschaft `ControlBox` auf `false` und löschen Sie den Inhalt der Eigenschaft `Text`. Dann besitzt das Formular den in `FormBorderStyle` definierten Rahmen, aber keine Titelleiste.

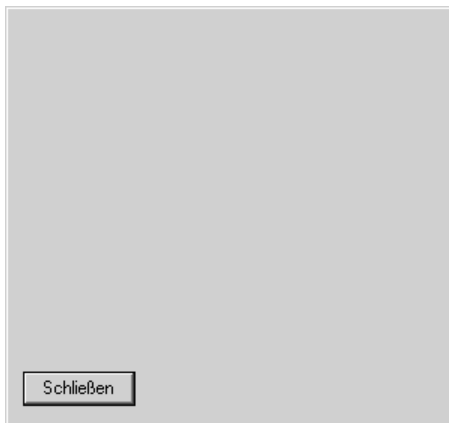


Abbildung 120: Formular mit einem veränderbaren Rahmen, `ControlBox = false` und keinem Titel

175 (Rahmenlose) Formulare über den Clientbereich verschiebbar machen

Ein Formular mit `FormBorderStyle = FormBorderStyle.None`, ohne Titelleiste (siehe Rezept 174) oder mit einer besonderen Form (siehe Rezept 180) kann vom Anwender nicht verschoben werden, da die Titelleiste fehlt. Dieses Problem können Sie aber lösen, indem Sie dem Anwender ermöglichen, das Formular über den Clientbereich zu verschieben. Diese Technik können Sie natürlich auch bei normalen Formularen einsetzen.

Die eleganteste Lösung dazu ist, die Methode `WndProc` des Formulars zu überschreiben. Diese Methode wird von Windows immer dann aufgerufen, wenn eine Windows-Nachricht an das Fenster gesendet wurde. Das ist z.B. dann der Fall, wenn der Anwender die Maus über dem Formular bewegt oder mit der Maus auf dieses klickt.

Basics

Datum
Zeit

Anwen-
dunge

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

Die Nachricht können Sie über das By Reference übergebene Argument `m` lesen und auch überschreiben. In der Eigenschaft `Msg` des damit referenzierten `Message`-Objekts wird die Art der Nachricht in Form eines `int`-Werts verwaltet. Die dazu verwendeten Konstanten sind größtenteils in der Headerdatei *winuser.h* deklariert. Die Konstante `WM_NCHITTEST` steht für die Nachricht, dass der Anwender die Maus auf dem Fenster bewegt oder mit der Maus darauf geklickt hat.

Listing 315 zeigt, wie Sie die `WndProc`-Methode so überschreiben, dass das Bewegen des Formulars über den Clientbereich ermöglicht wird. Dabei wird die Nachricht `WM_NCHITTEST` abgefangen und überprüft, ob der Mauscursor sich im Clientbereich des Formulars befindet. Ist dies der Fall, simuliert `WndProc`, dass der Mauscursor auf der Titelleiste bewegt wird, indem die Konstante `HTCAPTION` in das Ergebnis der Nachricht geschrieben und die Methode beendet wird. Am Schluss wird die geerbte `WndProc`-Methode aufgerufen, damit alle anderen Nachrichten korrekt vom Formular verarbeitet werden.

```
protected override void WndProc(ref Message m)
{
    const int WM_NCHITTEST = 0x0084;
    const int HTCAPTION = 2;

    // Abfangen der Nachricht WM_NCHITTEST
    if (m.Msg == WM_NCHITTEST)
    {
        if (this.ClientRectangle.Contains(this.PointToClient(Cursor.Position)))
        {
            // Wenn der Cursor sich im Clientbereich des Formulars befindet:
            // Simulieren, dass der Cursor sich auf der Titelleiste befindet
            m.Result = (IntPtr)HTCAPTION;
            return;
        }
    }

    base.WndProc(ref m);
}
```

Listing 315: Abfangen und Abändern der Nachricht `WM_NCHITTEST` zum Simulieren der Mauscursorbewegung auf der Titelleiste

Ein Nachteil dieser Technik ist, dass Sie Mausereignisse im Formular nicht mehr über die Formularereignisse auswerten können, da `WndProc` die Mausnachrichten abfängt und nicht an die geerbte `WndProc`-Methode weitergibt. Mausereignisse der Steuerelemente werden allerdings weiter behandelt, da diese eine eigene `WndProc`-Methode besitzen.

Eine andere Lösung des Verschiebe-Problems setzt die API-Funktion `ReleaseCapture` ein, die die Maus freigibt, und die Funktion `SendMessage`, die eine Nachricht an ein Fenster (oder an Windows) sendet. Über die Nachricht `WM_NCLBUTTONDOWN` mit dem Argument `HTCAPTION` können Sie die Betätigung einer Maustaste auf der Titelleiste simulieren.

Dazu benötigen Sie zunächst den Import des Namensraums `System.Runtime.InteropServices` und die Deklaration der verwendeten API-Funktionen und -Konstanten:

```
[DllImport("User32.dll")]
private static extern int SendMessage(IntPtr handle, int msg, int wParam,
    int lParam);

[DllImport("User32.dll")]
public static extern int ReleaseCapture();

private const int WM_NCLBUTTONDOWN = 0x00A1;
private const int HTCAPTION = 2;
```

Listing 316: Deklaration der für die Maus-Simulation benötigten Konstanten

Im `MouseDown`-Ereignis des Formulars überprüfen Sie, ob die linke Maustaste betätigt wurde. Ist dies der Fall, rufen Sie `ReleaseCapture` auf um die Maus vom Fenster freizugeben. Über `SendMessage` senden Sie schließlich die Nachricht `WM_NCLBUTTONDOWN` mit dem Parameter `HTCAPTION`:

```
private void WndProcDemoForm_MouseDown(object sender,
    System.Windows.Forms.MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        // Wenn die linke Maustaste betätigt wurde: Maus freigegeben
        // und simulieren, dass der Benutzer auf der Titelleiste
        // geklickt hat
        ReleaseCapture();
        SendMessage(Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0);
    }
}
```

Listing 317: Simulieren eines Klicks auf die Titelleiste eines Formulars

Diese Technik ist zwar etwas komplexer, besitzt aber den Vorteil, dass Sie Mausereignisse im Formular ganz normal auswerten können.

176 Unbewegbare Formulare mit Titelleiste

Wenn Sie Formulare unbewegbar machen wollen, können Sie den Formular-Rahmen einfach auf `FormBorderStyle.None` setzen. Soll das Formular aber einen Titel besitzen, müssen Sie eine andere Technik einsetzen. Sie können dazu im Systemmenü einfach den Befehl zum Verschieben entfernen. Fehlt dieser Befehl, kann der Anwender das Formular auch nicht mehr über die Titelleiste verschieben.

Dazu verwenden Sie die API-Funktion `DeleteMenu`, der Sie am ersten Argument den Handle des Systemmenüs, am zweiten die Konstante `SC_MOVE` (für das Verschieben-Menü) und am

dritten Argument (an dem Flags übergeben werden) die Konstante `MF_BYCOMMAND` übergeben (um zu definieren, dass das zweite Argument nicht als Index, sondern als Menü-Id interpretiert werden soll). Den Handle des Systemmenüs erhalten Sie über die API-Funktion `GetSystemMenu`. Am ersten Argument übergeben Sie den Handle des Formulars, am zweiten bestimmen Sie mit 0 (was dem `FALSE` des Windows-APIs entspricht), dass eine veränderbare Kopie des Systemmenüs erzeugt werden soll. Das Ganze programmieren Sie entweder im Konstruktor des Formulars oder im Load-Ereignis.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
[DllImport("user32.dll")]
private static extern IntPtr GetSystemMenu(IntPtr hWnd, int bRevert);

[DllImport("user32.dll")]
private static extern int DeleteMenu(IntPtr hMenu, int uPosition, int uFlags);

private const int SC_MOVE = 0xF010;
private const int MF_BYCOMMAND = 0;

private void StartForm_Load(object sender, System.EventArgs e)
{
    // Handle des Systemmenüs ermitteln und den Verschieben-Befehl löschen
    IntPtr sysMenuHandle = GetSystemMenu(this.Handle, 0);
    DeleteMenu(sysMenuHandle, SC_MOVE, MF_BYCOMMAND);
}
```

Listing 318: Unbewegbar-Machen eines Formulars

177 Andockende Formulare

Formulare, die am inneren Rand eines Parent-Formulars bzw. aneinander andocken, wie in Abbildung 121 gezeigt, können Sie recht einfach implementieren.

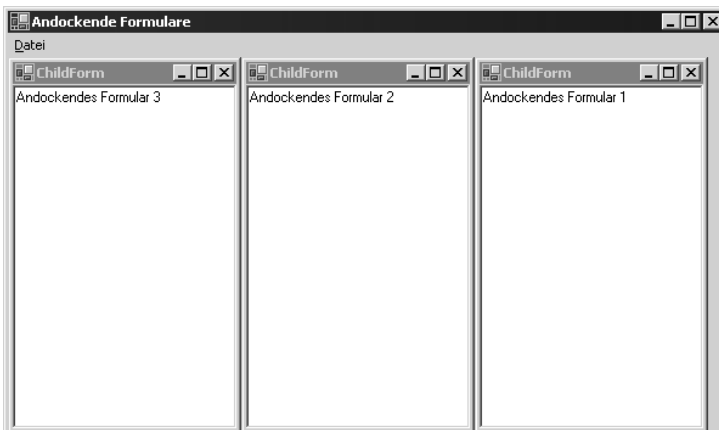


Abbildung 121: Andockende Formulare

Der Trick ist, dass Sie das neue Formular der `Controls`-Auflistung des `Parent`-Formulars zuweisen und die von `Control` geerbte Eigenschaft `Dock` auf einen Wert der `DockStyle`-Aufzählung setzen. Wichtig ist dann nur noch, dass die `TopLevel`-Eigenschaft des Formulars auf `false` eingestellt wird:

```
ChildForm f = new ChildForm();
f.TopLevel = false;
this.Controls.Add(f);
f.Dock = DockStyle.Left;
f.Show();
```

Listing 319: Anzeigen eines andockenden Formulars

Das Beispiel erzeugt eine neue Instanz der Klasse `ChildForm`, die ein ganz normales Formular (mit einer `TextBox`) ist und im `Click`-Ereignis des `Datei/Neu`-Menüs programmiert.

Wenn Sie nun noch über das Rezept 176 dafür sorgen, dass der Anwender das Formular nicht bewegen kann, bleiben die Formulare immer schön aneinander und am `Parent`-Formular andockt.

178 Ändern des Systemmenüs: Entfernen des Schließen-Eintrags und -Schalters

Den Schließen-Eintrag und den entsprechenden Schalter im Systemmenü bzw. in der Titelleiste eines Formulars können Sie nicht separat abschalten. Die Eigenschaft `ControlBox` erlaubt nur das komplette Abschalten des Systemmenüs. Wenn Sie aber ein Formular mit Systemmenü und Minimieren- und Maximieren-Schaltfläche, aber ohne den Schließen-Befehl benötigen, müssen Sie das Systemmenü über API-Funktionen verändern. Die Vorgehensweise entspricht der des Rezepts 176, nur dass Sie hier eben den Schließen-Befehl über die Konstante `SC_CLOSE` löschen.

Listing 320 implementiert dieses Löschen im `Load`-Ereignis eines Formulars. Neben dem Schließen-Befehl wird auch der Trennbalken entfernt, der vor dem Schließen-Eintrag platziert ist. Da dieser nicht über eine `Id` erreichbar ist, löscht `StartForm_Load` den Balken über dessen Position.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` einbinden.

```
[DllImport("user32.dll")]
private static extern IntPtr GetSystemMenu(IntPtr hWnd, int bRevert);

[DllImport("user32.dll")]
private static extern int DeleteMenu(IntPtr hMenu, int uPosition, int uFlags);

private const int SC_CLOSE = 0xF060;
```

Listing 320: Entfernen des Schließen-Befehls aus dem Systemmenü

```
private const int MF_BYCOMMAND = 0;
private const int MF_BYPOSITION = 0x00000400;

private void StartForm_Load(object sender, System.EventArgs e)
{
    // Handle des Systemmenüs ermitteln und den Schließen-Befehl und
    // den nun überflüssigen Separator löschen
    IntPtr sysMenuHandle = GetSystemMenu(this.Handle, 0);
    DeleteMenu(sysMenuHandle, SC_CLOSE, MF_BYCOMMAND);
    DeleteMenu(sysMenuHandle, 5, MF_BYPOSITION);
}
```

Listing 320: Entfernen des Schließen-Befehls aus dem Systemmenü (Forts.)

Das Löschen des Trennbalkens funktioniert auf diese Weise natürlich nur korrekt, sofern das Systemmenü nicht bereits vorher geändert wurde. Gegebenenfalls müssen Sie den Index (per Voreinstellung 5) anpassen.

179 Formulare verlaufend füllen

In einigen Anwendungen werden Formulare in einer Farbe verlaufend gefüllt dargestellt. Dieses »Feature« können Sie auch recht einfach programmieren ohne dafür ein Hintergrundbild verwenden zu müssen. GDI+ stellt dazu im Namensraum `System.Drawing.Drawing2D` die Klasse `LinearGradientBrush` zur Verfügung. Bei der Erzeugung eines solchen Pinsels geben Sie in einer Variante des Konstruktors das Ausgaberechteck an (damit der Verlauf korrekt berechnet werden kann), eine Startfarbe, eine Endfarbe und den Winkel der Füllrichtung. Im `Paint`-Ereignis können Sie einen `LinearGradientBrush`-Pinsel verwenden, um über `FillRectangle` ein gefülltes Rechteck zu zeichnen:

```
private void StartForm_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // LinearGradientBrush für die verlaufende Füllung erzeugen
    LinearGradientBrush brush = new LinearGradientBrush(
        this.ClientRectangle, Color.Blue, Color.Black, 30, false);

    // Formularfläche mit dem Pinsel füllen
    g.FillRectangle(brush, this.ClientRectangle);
}
```

Listing 321: Verlaufendes Füllen eines Formulars

Das Beispiel benötigt neben den für eine Windows-Forms-Anwendung verwendeten Namensräumen den Import der Namensräume `System`, `System.Drawing` und `System.Drawing.Drawing2D`.

Abbildung 122 zeigt das Ergebnis.



Abbildung 122: Ein mit einem Winkel von 30 Grad verlaufend gefülltes Formular

180 Formulare mit speziellen Formen

Formulare mit speziellen Formen (»Shaped Forms«) wie z. B. die Fenster, die mit vielen Multimedia-Anwendungen wie dem Windows Media Player und dem Winamp über deren Skins möglich sind, sind in einer Windows.Forms-Anwendung ebenfalls möglich. Grundlage dazu ist, dem Formular ein Bild zuzuweisen, das der späteren Form entspricht, oder auf dem Formular eine Grafik zu zeichnen. Zur Erstellung der besonderen Form können Sie zwei Techniken verwenden: einen transparenten Hintergrund und eine Region.

Zum Kompilieren der in diesem Rezept beschriebenen Methoden benötigen Sie neben dem Formular und der Referenz auf die Assembly *System.Forms.dll* zwei Bitmaps, die Sie der Manifest-Ressource der Anwendung zuweisen. Unter Visual Studio binden Sie dazu einfach die Bitmap-Dateien in das Projekt ein und stellen deren Eigenschaft *Buildaktion* auf *Eingebettete Ressource*. Daneben müssen Sie die Namensräume *System*, *System.Drawing*, *System.Drawing.Drawing2D* und *System.Reflection* importieren.

Transparenter Hintergrund

Die einfachste Lösung zur Erstellung eines Formulars mit einer speziellen Form ist, einen einfarbigen Hintergrund zu verwenden und der Eigenschaft *TransparencyKey* des Formulars diese Hintergrundfarbe zuzuweisen. Stellen Sie dann noch die Eigenschaft *FormBorderStyle* auf *FormBorderStyle.None*, kann das Ergebnis sich so einigermaßen sehen lassen.

Abbildung 123 zeigt ein solches Formular, dessen *BackColor*- und *TransparencyKey*-Eigenschaft auf *Color.Silver* eingestellt ist.

Der Schalter ist übrigens ein ganz normaler Button im flachen Stil, dem als Hintergrundbild dieselbe Textur zugewiesen wurde, die auch für das später gezeichnete Rechteck verwendet wird.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

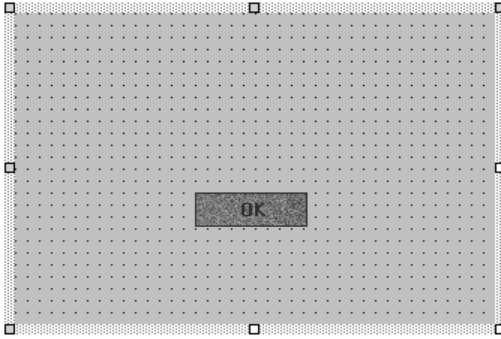


Abbildung 123: Das Basisformular in der Entwicklungsumgebung

Im `Paint`-Ereignis des Formulars werden ein gefülltes Rechteck und ein gefüllter Kreis gezeichnet:

```
private void ShapedForm1_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // Die beiden Texturen aus der Ressource lesen und damit zwei
    // TextureBrush-Objekte für das Zeichnen erzeugen
    Assembly assembly = Assembly.GetExecutingAssembly();
    TextureBrush brush1 = new TextureBrush(new Bitmap(
        assembly.GetManifestResourceStream(
            "Formulare_mit_freien_Formen.texture1.jpg")));
    TextureBrush brush2 = new TextureBrush(new Bitmap(
        assembly.GetManifestResourceStream(
            "Formulare_mit_freien_Formen.texture2.jpg")));

    // Gefülltes Rechteck und gefüllten Kreis zeichnen
    g.SmoothingMode = SmoothingMode.HighQuality;
    g.FillRectangle(brush1, 0, 40, 300, 100);
    g.FillEllipse(brush2, 65, 5, 180, 180);
}
```

Listing 322: Zeichnen eines mit einer Textur gefüllten Rechtecks und Kreises

Abbildung 124 zeigt das Ergebnis.

Diese Technik führt zwar zu einem schnellen Ergebnis, ist allerdings nicht optimal: Der Hintergrund des Formulars ist zwar durchsichtig, der Anwender spricht aber beim Klicken auf diesen nicht die darunter liegenden Objekte, sondern das Formular an. Ein anderes Problem ist, dass alle Formularbereiche in der als transparent definierten Farbe durchsichtig erscheinen. Wäre der Schalter auf dem Formular in Abbildung 124 ein normaler Schalter, wären die silbernen Teile des Schalters ebenfalls durchsichtig. Besonders schön sieht das dann nicht mehr aus.

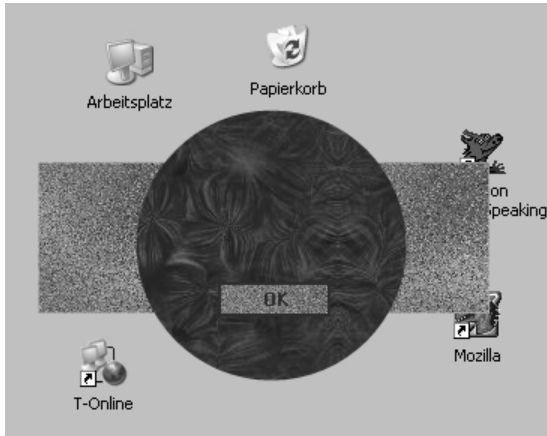


Abbildung 124: Ein Formular mit transparentem Hintergrund

ACHTUNG

Beim Test unter einem frisch installierten Windows 98 (SE) trat ein weiteres Problem auf: Das Transparent-Schalten funktionierte dort einfach nicht. Ich konnte allerdings nicht herausfinden, woran dieser Fehler lag.

Regionen

Die Lösung dieser Probleme ist eine *Region*. Eine solche definiert den sichtbaren Fensterbereich eines Formulars (oder eines Steuerelements) und kann in beliebigen Formen erstellt werden. Die Region eines Formulars erreichen Sie über dessen Eigenschaft `Region`. Per Default ist einem Formular (logischerweise) keine Region zugewiesen, weswegen die Eigenschaft zunächst den Wert `null` speichert.

Sie können aber eigene Regionen erzeugen und dem Formular zuweisen. Beim Erzeugen eines `Region`-Objekts (aus dem Namensraum `System.Drawing`) können Sie u.a. ein `Rectangle`-Objekt übergeben (das eine einfache, rechteckige Region definiert). Für komplexere Regionen übergeben Sie dem Konstruktor jedoch ein `GraphicsPath`-Objekt. Die `GraphicsPath`-Klasse (aus dem Namensraum `System.Drawing.Drawing2D`) erlaubt über eine Vielzahl an Methoden die Definition der Region. Die Methode `AddLine` fügt dem Pfad z.B. eine Linie hinzu, `AddArc` fügt einen Ellipsen-Ausschnitt hinzu, `AddRectangle` ein Rechteck, `AddEllipse` eine Ellipse oder einen Kreis etc. Diese Methoden sind funktional identisch zu den entsprechenden *Draw*-Methoden der `Graphics`-Klasse, was deren Anwendung erheblich vereinfacht. Das Geniale an diesen Methoden ist, dass Linien oder Ellipsenbögen, die sich an den Endpunkten berühren, automatisch verbunden werden. So können Sie mit ein wenig Arbeit auch die komplexesten Formen realisieren.

Listing 324 zeigt die Definition einer Region für das Formular in Abbildung 124. Die Region wird im `Load`-Ereignis des Formulars erzeugt. Das Beispiel ist bewusst etwas umständlich (wie Sie in Listing 323 sehen, geht es in unserem Fall auch etwas einfacher). Daran können Sie erkennen, wie komplexe Regionen erzeugt werden können.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
private void ShapedForm_Load(object sender, System.EventArgs e)
{
    // Neue Region über ein GraphicsPath-Objekt definieren
    GraphicsPath path = new GraphicsPath();

    // Obere Linie mit Kreis-Halbbogen
    path.AddLine(0, 40, 85, 40);
    path.AddArc(65, 5, 180, 180, 218, 104);
    path.AddLine(225, 40, 300, 40);

    // Untere Linie mit Kreis-Halbbogen
    path.AddLine(0, 140, 78, 140);
    path.AddArc(65, 5, 180, 180, 30, 120);
    path.AddLine(232, 140, 300, 140);

    // Linke und rechte Linie
    path.AddLine(0, 40, 0, 140);
    path.AddLine(300, 40, 300, 140);

    // Region dem Formular zuweisen
    this.Region = new Region(path);
}
```

Listing 323: Definieren der Region des Formulars über ein GraphicsPath-Objekt

Das Ergebnis des Programms entspricht zusammen mit dem Zeichnen im `Paint`-Ereignis der Abbildung 124, nur dass eben hier die Probleme mit der transparenten Farbe und dem Klicken auf dem transparenten Formularhintergrund vermieden werden.

Als kleinen Trick habe ich die Region übrigens zuvor zunächst im `Paint`-Ereignis testweise simuliert, indem ich statt `path.Add` einfach `e.Graphics.Draw` eingesetzt und am ersten Argument der `Draw`-Methoden ein `Pen`-Objekt eingefügt habe:

```
private void ShapedForm_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // Obere Linie mit Kreis-Halbbogen
    Pen p = new Pen(Color.Black, 2);
    g.DrawLine(p, 0, 40, 85, 40);
    g.DrawArc(p, 65, 5, 180, 180, 218, 104);
    g.DrawLine(p, 225, 40, 300, 40);

    // Untere Linie mit Kreis-Halbbogen
    g.DrawLine(p, 0, 140, 78, 140);
    g.DrawArc(p, 65, 5, 180, 180, 30, 120);
    g.DrawLine(p, 232, 140, 300, 140);
}
```

Listing 324: Testweise Simulation einer Region im Paint-Ereignis

```
// Linke und rechte Linie
g.DrawLine(p, 0, 40, 0, 140);
g.DrawLine(p, 300, 40, 300, 140);
}
```

Listing 324: Testweise Simulation einer Region im Paint-Ereignis (Forts.)

Das Ergebnis dieses Tests sehen Sie in Abbildung 125.

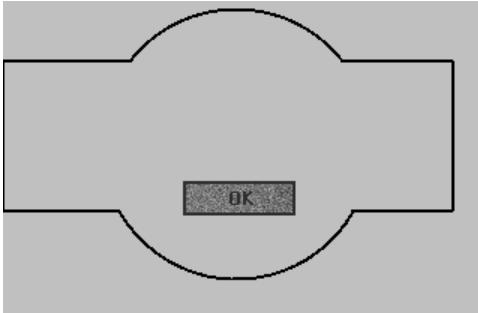


Abbildung 125: Testweises Zeichnen der Region im Paint-Ereignis

Wie bereits gesagt können Sie die Region in unserem Fall auch einfacher definieren. Dazu fügen Sie dem Pfad einfach ein Rechteck und einen Kreis hinzu, der den im Paint-Ereignis gezeichneten entspricht. Dabei kommt ein weiteres interessantes Feature von Regionen ins Spiel: Per Voreinstellung werden geschlossene Pfade, die sich überlagern, alternierend unsichtbar/sichtbar geschaltet. Auf diese Weise können Sie auch Formen erzeugen, die im Innenbereich transparent sind. Die Definition eines Rechtecks mit der gefolgten Definition eines Kreises würde den Bereich, an dem der Kreis das Rechteck überschneidet, durchsichtig erscheinen lassen. Wenn Sie dieses Verhalten ändern wollen, geben Sie am ersten Argument des GraphicsPath-Konstruktors den Wert FillMode.Winding an.

```
private void ShapedForm_Load(object sender, System.EventArgs e)
{
    GraphicsPath path = new GraphicsPath(FillMode.Winding);

    path.AddRectangle(new Rectangle(0, 40, 300, 100));
    path.AddEllipse(65, 5, 180, 180);

    this.Region = new Region(path);
}
```

Listing 325: Einfachere Definition einer Region mit geometrischen Formen

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

181 Splash-Formulare

Splash-Formulare sind Formulare, die beim Starten einer Anwendung erscheinen, um den Anwender während einer länger andauernden Initialisierungsphase Informationen anzubieten oder um einfach nur anzuzeigen, dass das Programm im Moment initialisiert wird.

Wenn Sie die Initialisierungen im `Load`-Ereignis des Startformulars vornehmen, ist die Anzeige eines Splash-Formulars sehr einfach. Listing 326 zeigt dies am Beispiel. Das Splash-Formular mit dem Namen *SplashForm* wird in diesem Beispiel nach dem Erzeugen und Anzeigen in einer Schleife aktualisiert. Dazu schreibt `StartForm_Load` den aktuellen Indexwert in ein Label mit dem Namen *infoLabel*, das auf dem Splash-Formular angelegt ist, und ruft die `Refresh`-Methode dieses Labels auf um die Anzeige zu aktualisieren.

Der Aufruf von `Application.DoEvents` nach dem Anzeigen des Splash-Formulars ist übrigens sehr wichtig. Ohne diesen Aufruf würde das Formular – besonders wenn es Bilder enthält – nicht unbedingt vor der Schleife vollständig gezeichnet werden. Der Grund dafür liegt in der Tatsache, dass die Priorität von Ausgaben auf einem Fenster in der Windows-Nachrichtenwarteschleife nur sehr gering ist. Ohne `DoEvents` würde das Programm die Schleife schon abarbeiten, während Windows das Fenster noch gar nicht zu Ende gezeichnet hat. `DoEvents` zwingt Windows hingegen, seine Nachrichtenwarteschleife komplett abzuarbeiten.

Die Ausnahmebehandlung in Listing 326 ist für die Praxis ebenfalls sehr wichtig. Falls beim Initialisieren Fehler auftreten, sollte die Anwendung normalerweise nicht starten. Das Beispiel ruft deswegen im Fehlerfall `Application.Exit` auf.

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // Splash-Formular erzeugen und anzeigen
    SplashForm f = new SplashForm();
    f.Show();
    Application.DoEvents();

    // Simulation einer Initialisierung
    try
    {
        for (int i = 0; i < 100; i++)
        {
            f.infoLabel.Text = "Lese Datensätze. Datensatz " + i;
            f.infoLabel.Refresh();

            Thread.Sleep(30);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fehler beim Initialisieren: " + ex.Message,
            Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
        Application.Exit();
    }
}
```

Listing 326: Anzeigen eines Splash-Formulars im `Load`-Ereignis des Startformulars

```

    }

    // Splash-Formular wieder schließen
    f.Close();
}

```

Listing 326: Anzeigen eines Splash-Formulars im Load-Ereignis des Startformulars (Forts.)

Zum Kompilieren dieser Methode benötigen Sie neben der Referenz auf die Assembly *System.Windows.Forms.dll* den Import der Namensräume `System`, `System.Windows.Forms` und `System.Threading`.

Abbildung 126 zeigt das Beispiel-Splash-Formular beim Start der Anwendung.



Abbildung 126: Splash-Formular beim Start einer Anwendung

Auf eine ähnliche Weise können Sie natürlich auch in der `Main`-Methode der Anwendung initialisieren und das Splash-Formular anzeigen:

```

static void Main()
{
    // Splash-Formular erzeugen und anzeigen
    SplashForm f = new SplashForm();
    f.Show();
    Application.DoEvents();

    // Simulation einer Initialisierung
    try
    {
        for (int i = 0; i < 100; i++)
        {
            f.infoLabel.Text = "Lese Datensätze. Datensatz " + i;
            f.infoLabel.Refresh();

            System.Threading.Thread.Sleep(30);
        }
    }
}

```

Listing 327: Anzeigen eines Splash-Formulars in der Main-Methode

Basics

Datum
Zeit

Anwen-
dungen

Dateti-
systeme

Dateti-

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

```

    }
    catch (Exception ex)
    {
        MessageBox.Show("Fehler beim Initialisieren: " + ex.Message,
            Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
        Application.Exit();
    }

    // Splash-Formular wieder schließen
    f.Close();

    Application.Run(new StartForm());
}

```

Listing 327: Anzeigen eines Splash-Formulars in der Main-Methode (Forts.)

182 Menüs in der Laufzeit erstellen oder erweitern

Viele Anwendungen erfordern, dass Menüs in der Laufzeit erstellt oder wenigstens erweitert werden. Ein gutes Beispiel dafür sind die Datei-Menüs von Office-Anwendungen, die beim Start der Anwendung und während der Arbeit automatisch die letzten geöffneten Dateien zum Öffnen anbieten. Das Erstellen eines Laufzeitmenüs ist nun sehr einfach, weil ein `MenuItem`-Objekt (das ein Menü oder einen Menüeintrag darstellt) eine Auflistung `MenuItem`s besitzt, über dessen `Add`-Methode Sie neue `MenuItem`-Instanzen hinzufügen können. Das einzige Problem (und damit der Grund für dieses Rezept) ist, dass Sie Menüeinträge, die Befehle darstellen, einen Delegate für die Behandlung des `Click`-Ereignisses zuweisen und in diesem den betätigten Menüeintrag ermitteln müssen.

Als Beispiel verwende ich ein Formular mit einem einfachen Menü (Abbildung 127).



Abbildung 127: Das Formular, dessen Datei-Menü in der Laufzeit erweitert werden soll

Im `Load`-Ereignis soll das Datei-Menü um die Dateinamen der zuletzt geöffneten Dateien erweitert werden (was hier aber nur simuliert wird). Dazu ist zunächst eine Methode für das `Click`-Ereignis des Menüeintrags notwendig:

```
private void DynamicMenuHandler(object sender, System.EventArgs e)
{
    MenuItem menuItem = (MenuItem)sender;
    string fileName = menuItem.Text;
    MessageBox.Show(fileName);
}
```

Listing 328: Methode für das Click-Ereignis der dynamisch erzeugten Menüeinträge

Diese Methode kann den Menüeintrag einfach über dessen Text identifizieren, was in Listing 329 auch so umgesetzt ist.

In der Delegate-Methode für das Load-Ereignis des Formulars wird das Datei-Menü nun dynamisch erweitert:

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // Hinzufügen von Menübefehlen für alle Textdateien
    // im Windows-Systemordner zum Datei-Menü
    DirectoryInfo winDir = new DirectoryInfo(Environment.SystemDirectory);
    FileInfo[] files = winDir.GetFiles("*.txt");
    for (int i = 0; i < files.Length && i < 9; i++)
    {
        // MenuItem-Objekt erzeugen und den Text und die Click-Ereignismethode
        // übergeben
        MenuItem menuItem = new MenuItem(files[i].Name,
            new EventHandler(this.DynamicMenuHandler));

        // Menü ab dem Index 2 + i an das Dateimenü anfügen
        this.fileMenu.MenuItems.Add(2 + i, menuItem);
    }
}
```

Listing 329: Dynamisches Erweitern eines Menüs

Das Beispiel geht alle Textdateien im Windows-Systemordner durch und fügt die Namen von bis zu maximal neun Dateien an das Dateimenü an. Beim Erzeugen des MenuItem-Objekts werden der Text des Eintrags und ein Delegate für die Click-Behandlungsmethode übergeben. Am im Beispiel nicht verwendeten dritten Argument können Sie übrigens noch einen Shortcut für das Menü definieren. Der Menüeintrag wird dann dem Dateimenü (das über die Eigenschaft fileMenu angesprochen wird) über dessen Add-Methode angefügt. Wie Sie sehen, gehen Sie beim Ändern eines Menüs mit einer ganz normalen Auflistung um. Über die Remove-Methode können Sie Menüeinträge z. B. auch wieder löschen, was beim dynamischen Verändern des Menüs während der Programmausführung notwendig wird.

Das Ergebnis zeigt Abbildung 128.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

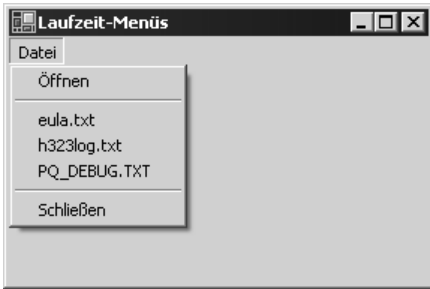


Abbildung 128: Das Programm mit dem dynamisch erweiterten Datei-Menü

183 Steuerelemente mit transparentem Hintergrund

Eine einfach gelöste, aber in Newsgroups immer wieder auftauchende Frage ist die, wie erreicht wird, dass Steuerelemente einen transparenten Hintergrund besitzen. Das gilt besonders für `Label`, die ihre Hintergrundfarbe zwar per Voreinstellung an die Hintergrundfarbe des Formulars anpassen, bei Formularen, denen ein Bild zugewiesen ist, aber sehr unschön aussehen.

Die Lösung dieses Problems ist einfach: Weisen Sie der Hintergrundfarbe des Steuerelements die Farbe `Color.Transparent` zu. Im Eigenschaftfenster von Visual Studio .NET finden Sie diese Farbe im Register `WEB`.

Ebenfalls interessant sind halb-durchsichtige Steuerelemente. Dies erreichen Sie, indem Sie im Programm eine Farbe zuweisen, deren Alpha-Wert kleiner als 255 ist. Der Alpha-Wert bestimmt die Transparenz. Ein Wert von 255 führt zu einer Undurchsichtigkeit, 0 ist vollständig transparent. Dazwischen können Sie alle Werte einsetzen, um die Transparenz gezielt einzustellen. Dazu verwenden Sie die Methode `FromArgb` der `Color`-Struktur (aus dem Namensraum `System.Drawing`) in der Variante mit vier Argumenten und geben den Alpha-Wert am ersten Argument an. So können Sie z. B. ein halbtransparent-rotes Label erzeugen:

```
this.demoLabel2.BackColor = Color.FromArgb(100, 255, 0, 0);
```



Abbildung 129: Undurchsichtiges, transparentes und halb-transparentes Label auf einem Formular

184 TextBox automatisch beim Eintritt selektieren

Die Selektion von `TextBox`-Steuerelementen wird beim Öffnen eines Formulars automatisch so eingestellt, dass der gesamte Text beim Fokuserhalt selektiert ist. Textboxen merken sich die aktuelle Selektion, sodass diese wiederhergestellt wird, wenn der Fokus von der Textbox wegbewegt und wieder auf diese gesetzt wird. Hat der Anwender die Selektion zwischenzeitlich geändert, wird die Textbox beim nächsten Fokuserhalt nicht mehr vollständig selektiert.

Wenn Sie eine sichere vollständige Selektion beim Fokuserhalt erreichen wollen, können Sie im `Enter`-Ereignis der Textbox einfach die `SelectAll`-Methode aufrufen:

```
private void firstNameTextBox_Enter(object sender, System.EventArgs e)
{
    this.firstNameTextBox.SelectAll();
}
```

Listing 330: Vollständiges Selektieren des Inhalts einer Textbox beim Fokuserhalt

Da dies recht aufwändig ist, wäre es vorteilhaft, eine eigene Textbox zu entwickeln, die diese Arbeit automatisch erledigt. Dazu verwenden Sie in Visual Studio .NET idealerweise ein neues Projekt vom Typ *Windows-Steuerelementbibliothek*. Die Klasse könnte dann aussehen wie die Klasse `AutoSelectTextBox` in Listing 331. Diese Klasse ist von `TextBox` abgeleitet und implementiert eine Eigenschaft `AutoSelect`, die das automatische Selektieren steuert. Die Attribute `Category` und `Description` sind übrigens nur enthalten, damit das neue Steuerelement innerhalb des Eigenschaftfensters von Visual Studio .NET in der Kategorie *Verhalten* einsortiert wird und eine Beschreibung zur neuen Eigenschaft erscheint.

Zur Umsetzung der automatischen Selektion überschreibt die Klasse einfach die geschützte Methode `OnEnter`.

```
using System;
using System.ComponentModel;
using System.Windows.Forms;

namespace Addison_Wesley.Codebook.Controls
{
    public class AutoSelectTextBox: TextBox
    {
        /* Eigenschaft für das automatische Selektieren */
        private bool autoSelect = true;
        [Category("Verhalten")]
        [Description("Definiert, ob der Inhalt der TextBox beim Fokuserhalt " +
            "automatisch selektiert wird")]
        public bool AutoSelect
        {
            get {return this.autoSelect;}
        }
    }
}
```

Listing 331: Klasse für eine erweiterte Textbox, die das automatische Selektieren beim Fokuserhalt ermöglicht

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        set {this.autoSelect = value;}
    }

    /* Überschreiben der OnEnter-Methode */
    protected override void OnEnter(System.EventArgs e)
    {
        if (this.autoSelect)
            this.SelectAll();
    }
}

```

Listing 331: Klasse für eine erweiterte TextBox, die das automatische Selektieren beim Fokuserhalt ermöglicht (Forts.)

Hinweis

Nur für den Fall, dass Sie nicht wissen, wie Sie dieses neue Steuerelement in Visual Studio .NET einsetzen: Kompilieren Sie die Klassenbibliothek und speichern Sie diese gegebenenfalls in einem Ordner, den Sie für Ihre allgemeinen eigenen Steuerelemente vorgesehen haben. In Visual Studio passen Sie nun die Toolbox an, indem Sie im Kontextmenü den Befehl TOOLBOX ANPASSEN wählen. Suchen Sie die Assembly des Steuerelements und fügen Sie diese hinzu. Nun können Sie das Steuerelement verwenden wie jedes andere auch. Bei der Entwicklung eines Steuerelements ist es zudem hilfreich, das Steuerelement-Projekt gemeinsam mit einem Testprojekt geöffnet zu haben. Änderungen am Steuerelement-Projekt wirken sich nach dem Kompilieren dann natürlich auch auf das Testprojekt aus.

185 TextBox auf Zahleingaben beschränken

In meinen Seminaren wird immer wieder gefragt, wie man erreichen kann, dass eine `TextBox` nur Zahleingaben zulässt. Das .NET Framework bietet dazu keine direkten Möglichkeiten. Sie können natürlich spezielle `TextBox`-Steuerelemente über Dritthersteller oder aus dem Internet beziehen. Sie können aber auch selbst eine solche `TextBox` programmieren. Da Sie dabei beachten müssen, dass Eingaben über die Tastatur, das Einfügen über die Zwischenablage und das Schreiben in die `Text`-Eigenschaft kontrolliert werden müssen, sollten Sie dazu direkt ein eigenes Steuerelement erstellen. Ansonsten haben Sie bei der Programmierung eines Formulars mit einigen `TextBox`-Steuerelementen einfach zu viel Arbeit.

Meine aus einem Artikel bei *The Code Project* (www.codeproject.com/cs/miscctrl/NumEdit.asp) abgeleitete Klasse wird also von `TextBox` abgeleitet. Um den Eingabetyp bestimmen zu können, habe ich eine Aufzählung `InputTypeEnum` und eine Eigenschaft `InputType` integriert.

Zum Kompilieren der Klasse `NumberTextBox` müssen Sie die Namensräume `System`, `System.ComponentModel` und `System.Windows.Forms` importieren.

```

public class NumberTextBox: TextBox
{
    public enum InputTypeEnum
    {
        Double,
        Integer
    }

    private InputTypeEnum inputType = InputTypeEnum.Double;

    [Category("Verhalten")]
    [Description("Definiert, welche Eingaben die TextBox zulässt")]
    public InputTypeEnum InputType
    {
        get {return this.inputType;}
        set {this.inputType = value;}
    }
}

```

Listing 332: Beginn der Klasse NumberTextBox mit der Eigenschaft InputType

Um Eingabefehler später auswerten zu können, folgt die Deklaration des Ereignisses `InvalidInput` mit dem dazu passenden Delegate:

```

[Description("Wird aufgerufen, wenn eine Eingabe nicht dem " +
    "Schema entspricht")]
public delegate void InvalidInputHandler(string invalidText);
public event InvalidInputHandler InvalidInput;

```

Listing 333: Ereignis für Eingabefehler-Meldungen

Nun folgt die Methode `inputValid`, die überprüft, ob die Eingabe in Ordnung ist. Diese Methode bekommt den potenziell neuen Text der `TextBox` übergeben und überprüft diesen einfach über eine Konvertierung nach `double` bzw. `long` auf Gültigkeit. Im Fehlerfall ruft `inputValid` das Ereignis `InvalidInput` auf, falls dieses zugewiesen ist:

```

private bool inputValid(string input)
{
    switch (this.inputType)
    {
        case InputTypeEnum.Double:
            try
            {
                Convert.ToDouble(input);
                return true;
            }
            catch
            {
                if (InvalidInput != null) InvalidInput(input);
            }
        }
    }
}

```

Listing 334: Methode zum Überprüfen, ob die Eingabe in Ordnung ist

Basics

Datum
Zeit

Anwen-
dunge

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

```
        return false;
    }

    default: // InputTypeEnum.Integer
    try
    {
        Convert.ToInt64(input);
        return true;
    }
    catch
    {
        if (InvalidInput != null) InvalidInput(input);
        return false;
    }
}
}
```

Listing 334: Methode zum Überprüfen, ob die Eingabe in Ordnung ist (Forts.)

Um ein Schreiben in die `Text`-Eigenschaft zu überprüfen, wird diese überschrieben. In der `set`-Methode ruft `Text inputValid` auf. Nur wenn der Text in Ordnung ist, wird dieser über die geerbte `Text`-Eigenschaft in die `TextBox` geschrieben:

```
public override string Text
{
    get {return base.Text;}
    set
    {
        if (this.inputValid(value))
            base.Text = value;
    }
}
```

Listing 335: Überschreiben der Text-Eigenschaft

Die `OnKeyPress`-Methode wird ebenfalls überschrieben, um Tastatureingaben überprüfen zu können. Dabei wird das eingegebene Zeichen so in den alten Text integriert, dass eine eventuelle Selektion oder eine Cursorposition vor dem Ende des Textes berücksichtigt wird. Der potenziell neue Text wird dann wieder über `inputValid` überprüft. Im Fehlerfall setzt `OnKeyPress e.Handled` auf `true`, damit die Eingabe nicht weiter verarbeitet wird. Die Überprüfung erfolgt nur dann, wenn es sich nicht um ein Steuerzeichen wie z.B. einen Tabulator oder die Backspace-Taste handelt.

```
protected override void OnKeyPress(KeyPressEventArgs e)
{
    if (Char.IsControl(e.KeyChar) == false)
    {
        // Den potenziell neuen Text zusammensetzen
        string newText = base.Text.Substring(0, base.SelectionStart)
            + e.KeyChar.ToString() + base.Text.Substring(
                base.SelectionStart + base.SelectionLength);

        if (this.inputValid(newText) == false)
            // Die Eingabe führt zu einem ungültigen Ergebnis,
            // also verwerfen
            e.Handled = true;
    }

    base.OnKeyPress(e);
}
```

Listing 336: Überschreiben der OnKeyPress-Methode zum Überprüfen von Tastatureingaben

Schließlich muss noch das Einfügen aus der Zwischenablage überprüft werden. Dazu wird die `ProcessCmdKey`-Methode überschrieben. Diese Methode wird für alle Befehlstasten ausgerufen. Nach der Überprüfung, ob es sich um die Tasten zum Einfügen handelt (`Strg` + `V` oder `Shift` + `Einfüg`), setzt meine Variante der `ProcessCmdKey`-Methode den potenziell neuen Text ähnlich wie bei `OnKeyPress` zusammen und überprüft diesen wieder über `inputValid`. Im Fehlerfall wird die Methode mit `true` beendet um das Ende der Befehlstastenverarbeitung zu signalisieren.

```
protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
{
    if (keyData == (Keys)Shortcut.CtrlV ||
        keyData == (Keys)Shortcut.ShiftIns)
    {
        // Daten aus der Zwischenablage auslesen
        IDataObject iData = Clipboard.GetDataObject();

        // Die neuen Daten mit dem Text außerhalb der aktuellen Auswahl
        // verknüpfen um den neuen Text zu erhalten
        string newText = base.Text.Substring(0, base.SelectionStart) +
            (string)iData.GetData(DataFormats.Text) +
            base.Text.Substring(base.SelectionStart +
                base.SelectionLength);

        // Den potenziell neuen Text überprüfen
        if (this.inputValid(newText) == false)
            // Beenden und true zurückgeben, um die Verarbeitung
            // abzuschließen
            return true;
    }
}
```

Listing 337: Überschreiben der ProcessCmdKey-Methode zum Abfangen der Tastenkombinationen für das Einfügen von Daten

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    }

    return base.ProcessCmdKey(ref msg, keyData);
}
}

```

Listing 337: Überschreiben der ProcessCmdKey-Methode zum Abfangen der Tastenkombinationen für das Einfügen von Daten (Forts.)

Hinweis

Ein Problem ist in dieser Version der `NumberedTextBox` übrigens noch nicht gelöst: Das Einfügen nicht numerischer Daten aus der Zwischenablage ist leider noch über das Kontextmenü der `TextBox` möglich. In der mir zur Verfügung stehenden Zeit habe ich keine Möglichkeit gefunden, dieses Einfügen abzufangen. Falls Sie eine Lösung finden, würde ich mich über eine E-Mail freuen 😊.

186 Auf Return in einer TextBox reagieren

Eine Reaktion auf die Betätigung der Return-Taste in einer `TextBox` ist eigentlich viel zu einfach, um hier als Rezept zu erscheinen. Dazu überprüfen Sie im `KeyPress`-Ereignis lediglich die betätigte Taste auf `'\r'`. Vielleicht wissen Sie aber nicht, wie Sie den Systemton abschalten können, den Windows automatisch bei einem Return in einer `TextBox` ausgibt:

```

private void demoTextBox_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == '\r')
    {
        // Ereignis als behandelt kennzeichnen um den Systemton zu verhindern
        e.Handled = true;

        // Weitere Programmierung
    }
}

```

Listing 338: Reaktion auf die Betätigung der Return-Taste in einer TextBox

187 ComboBox mit Autovervollständigung

Ein `ComboBox`-Steuerelement, das den zur Anwendereingabe nächsten passenden Listeneintrag sucht und die Eingabe nach rechts mit einer Selektion versehen entsprechend erweitert (siehe Abbildung 130), ist für Eingaben sehr hilfreich. Solch eine `ComboBox` können Sie selbst recht einfach erzeugen.

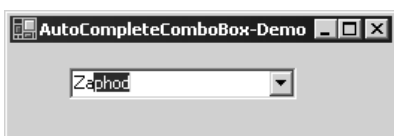


Abbildung 130: ComboBox mit Autovervollständigung

Da zur Erstellung einer solchen `ComboBox` wieder einmal etwas Programmierung notwendig ist, sollten Sie ein eigenes Steuerelement entwickelt, das Sie von der Klasse `ComboBox` ableiten. Die Klasse `AutoCompleteComboBox` in Listing 339 implementiert ein solches Steuerelement. Diese Klasse überschreibt zunächst die `OnTextChanged`-Methode. Diese Methode sucht über die geerbte `FindString`-Methode nach einem Eintrag, der mit der aktuellen Eingabe im Textfeld beginnt. Wird ein solcher gefunden, merkt sich die Methode den originalen Text, schreibt den gefundenen Eintrag in das Textfeld und selektiert über die Länge der beiden Texte den differenten Text. Diese Selektion bewirkt, dass der Anwender einfach auch mit anderen Zeichen weiter schreiben kann, als denen, die seiner bisherigen Eingabe nun folgen. Um eine Rekursion zu vermeiden (das Ändern der `SelectedIndex`-Eigenschaft führt wieder zum Aufruf der `OnTextChanged`-Methode) wird vor dem Setzen der `SelectedIndex`-Eigenschaft die private Eigenschaft `dontWork` auf `true` gesetzt und danach wieder auf `false`. Da diese Eigenschaft zu Anfang der Methode abgefragt wird, führt der erneute Aufruf nicht zu einer Rekursion.

Um die Backspace- und die Löschen-Taste zu erlauben, überschreibt `AutoCompleteComboBox` schließlich die `OnKeyDown`-Methode. Wurde eine der genannten Tasten betätigt, setzt `OnKeyDown` einfach `dontWork` auf `true`, in allen anderen Fällen auf `false`. Damit wird verhindert, dass `OnTextChanged` bei der Betätigung der Backspace- oder der Löschen-Taste arbeitet.

Die Klasse `AutoCompleteComboBox` erfordert den Import der Namensräume `System` und `System.Windows.Forms`.

```
public class AutoCompleteComboBox: ComboBox
{
    /* Private Eigenschaft zur Vermeidung rekursiver Aufrufe */
    private bool dontWork = false;

    protected override void OnTextChanged(System.EventArgs e)
    {
        if (this.dontWork == false)
        {
            // Nach einem Eintrag suchen, der am Anfang dem eingegebenen
            // Text entspricht
            int index = base.FindString(base.Text);
            if (index >= 0)
            {
                // Eintrag gefunden: Alten Text merken, neuen Eintrag
                // auswählen und den differenten Teil selektieren
                string input = base.Text;
                this.dontWork = true;
                base.SelectedIndex = index;
                this.dontWork = false;
                base.Select(input.Length, base.Text.Length);
            }
        }

        base.OnTextChanged(e);
    }
}
```

Listing 339: `ComboBox` mit Auto-Vervollständigung

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    }

    protected override void OnKeyDown(System.Windows.Forms.KeyEventArgs e)
    {
        // Verarbeitung vermeiden, wenn die Backspace- oder die Löschen-
        // Taste betätigt wurde
        this.dontWork = (e.KeyCode == Keys.Back || e.KeyCode == Keys.Delete);
        base.OnKeyDown(e);
    }
}

```

Listing 339: ComboBox mit Auto-Vervollständigung (Forts.)

188 Daten neben den Einträgen einer ListBox oder ComboBox verwalten

Häufig besteht der Bedarf, neben den Einträgen in einer `ListBox` oder `ComboBox` weitere Daten zu verwalten. Das ist z.B. dann der Fall, wenn Sie Datensätze aus einer Datenbanktabelle auslesen und in einer `ListBox` zur Auswahl darstellen. Um den ausgewählten Datensatz eindeutig identifizieren zu können, müssen Sie den Primärschlüssel des Datensatzes neben dem Listeneintrag speichern (wenigstens dann, wenn Sie diesen nicht mit anzeigen wollen).

Die Lösung dieses Problems ist sehr einfach: Erstellen Sie für die Listeneinträge eine Klasse oder Struktur, deren `ToString`-Methode den Text zurückgibt, der in der Liste angezeigt werden soll. In den Eigenschaften der Klasse/Struktur können Sie natürlich beliebige Daten verwalten. Beim Einlesen der Liste erzeugen Sie für jeden Eintrag eine Instanz und hängen diese der Liste an. Beim Auslesen der Liste können Sie das Objekt am jeweiligen Eintrag wieder in seinen Typ casten und somit auf die Eigenschaften zugreifen.

Um dies an einem Beispiel zu erläutern, sollen Personendaten in einer `ListBox` ausgegeben werden. Dazu habe ich die Klasse `Person` implementiert:

```

private class Person
{
    public int Id;
    public string FirstName;
    public string LastName;

    /* Konstruktor */
    public Person(int id, string firstName, string lastName)
    {
        this.Id = id;
        this.FirstName = firstName;
        this.LastName = lastName;
    }
}

```

Listing 340: Klasse für die Speicherung von Personendaten zur Ausgabe in einer ListBox oder ComboBox

```
/* Die ToString-Methode wird überschrieben */
public override string ToString()
{
    return this.FirstName + " " + this.LastName;
}
}
```

Listing 340: Klasse für die Speicherung von Personendaten zur Ausgabe in einer ListBox oder ComboBox (Forts.)

Im Load-Ereignis eines Formulars werden Personendaten aus einer Datenbank eingelesen und in eine ListBox geschrieben (was hier nur simuliert wird):

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // Einige Personen an die ListBox anhängen
    this.personList.Items.Add(new Person(1001, "Zaphod", "Beeblebrox"));
    this.personList.Items.Add(new Person(1002, "Tricia", "McMillan"));
    this.personList.Items.Add(new Person(1003, "Arthur", "Dent"));
    this.personList.Items.Add(new Person(1004, "Ford", "Prefect"));
    this.personList.Items.Add(new Person(1005, "Marvin", ""));
}
```

Listing 341: Füllen der Liste mit neuen Person-Objekten

Bei der Auswahl eines Eintrags liest das Beispiel das aktuelle Objekt aus der Liste aus und stellt dessen Daten in einem Label dar:

```
private void personList_SelectedIndexChanged(object sender,
    System.EventArgs e)
{
    // Auslesen des selektierten Objekts
    Person p = (Person)personList.SelectedItem;
    this.infoLabel.Text = "Id: " + p.Id + "\r\n" +
        "Vorname: " + p.FirstName + "\r\n" +
        "Nachname: " + p.LastName;
}
```

Listing 342: Auslesen des gewählten Objekts bei der Änderung des selektierten Eintrags

Das Ergebnis zeigt Abbildung 131.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

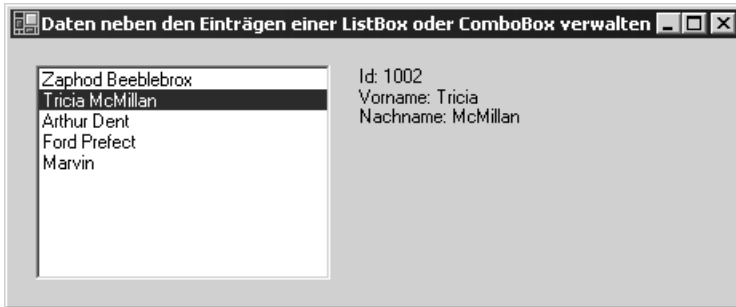


Abbildung 131: Das Beispielprogramm in Aktion

189 ListView sortieren

ListView-Steuerelemente lassen sich in vielen Anwendungen in der Detailansicht durch einen Klick auf eine Spalte sortieren. Standardmäßig sortiert ein ListView-Steuerelement die Einträge aber nur nach der ersten Spalte, wenn Sie in der Eigenschaft `Sorting` den Wert `SortOrder.Ascending` oder `SortOrder.Descending` angeben.

Sie können aber stattdessen auch `Sorting` auf `SortOrder.None` einstellen und zum Sortieren die `Sort`-Methode aufrufen. `Sort` sortiert die Liste nach dem Objekt, das der Eigenschaft `ListViewItemSorter` zugewiesen ist und das die Schnittstelle `IComparer` implementiert. Und das ist der Trick.

Abbildung 132 zeigt ein Beispielformular mit einem ListView-Steuerelement mit Namen `bookList`. Die einzelnen Spalten sollen bei einem Klick auf die Überschrift automatisch sortiert werden. Das Beispiel ist nach der ISBN-Nummer sortiert.

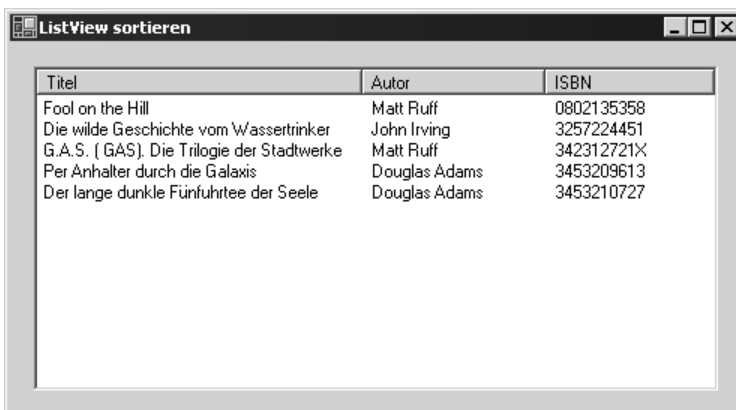


Abbildung 132: Beispielformular für das Sortieren eines ListView-Steuerelements

Zur Lösung des Sortierproblems können Sie nun eine Klasse implementieren, die die `Compare`-Methode der Schnittstelle `IComparer` implementiert:

```
public class ColumnComparer: IComparer
{
    /* Eigenschaft für die aktuelle Spalte */
    public int CurrentColumn = 0;

    /* Implementieren der Compare-Methode */
    public int Compare(object x, object y)
    {
        ListViewItem firstItem = (ListViewItem)x;
        ListViewItem secondItem = (ListViewItem)y;

        return string.Compare(
            firstItem.SubItems[CurrentColumn].Text,
            secondItem.SubItems[CurrentColumn].Text);
    }
}
```

Listing 343: Klasse zum Sortieren eines ListView-Steuerelements

Die Eigenschaft `CurrentColumn` nimmt später den Index der aktuell angeklickten Spalte auf. Über diesen Index vergleicht die `Compare`-Methode die Spaltenwerte der übergebenen `ListViewItem`-Objekte. `Compare` muss einen Wert kleiner Null zurückgeben, wenn das `x` kleiner `y` ist, einen Wert größer Null, wenn `x` größer `y` ist, und Null, wenn beide Objekte gleich sind. Da die `Compare`-Methode der `String`-Klasse einen passenden Vergleich liefert, ruft `Compare` einfach diese Methode auf.

Im Formular benötigen Sie dann eine private Eigenschaft, die eine Instanz der `ColumnComparer`-Klasse aufnimmt:

```
private ColumnComparer columnComparer = new ColumnComparer();
```

Im `Load`-Ereignis (oder wo auch immer Sie das `ListView`-Steuerelement initialisieren) weisen Sie der Eigenschaft `ListViewItemSorter` die Referenz auf die `ColumnComparer`-Instanz zu:

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // Füllen des ListView-Steuerelements
    this.bookList.Items.Add(new ListViewItem(new string[] {
        "Fool on the Hill", "Matt Ruff", "0802135358"}));
    this.bookList.Items.Add(new ListViewItem(new string[] {
        "G.A.S. ( GAS). Die Trilogie der Stadtwerke", "Matt Ruff",
        "342312721X"}));
    this.bookList.Items.Add(new ListViewItem(new string[] {
```

Listing 344: Initialisieren des ListView-Steuerelements

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        "Per Anhalter durch die Galaxis", "Douglas Adams",
        "3453209613")));
this.bookList.Items.Add(new ListViewItem(new string[] {
    "Der lange dunkle Fünfuhrtee der Seele", "Douglas Adams",
    "3453210727")));
this.bookList.Items.Add(new ListViewItem(new string[] {
    "Die wilde Geschichte vom Wassertrinker", "John Irving",
    "3257224451")));

// Comparer zuweisen
this.columnComparer.CurrentColumn = 2; // Sortieren nach ISBN
this.bookList.ListViewItemSorter = this.columnComparer;
}

```

Listing 344: Initialisieren des ListView-Steuerelements (Forts.)

Nun müssen Sie nur noch den Klick auf eine Spalte auswerten. Dazu können Sie das `ColumnClick`-Ereignis verwenden. Hier stellen Sie die aktuelle Spalte für Ihr `ColumnComparer`-Objekt ein und rufen die `Sort`-Methode des `ListView`-Steuerelements auf:

```

private void bookList_ColumnClick(object sender, ColumnClickEventArgs e)
{
    this.columnComparer.CurrentColumn = e.Column;
    this.bookList.Sort();
}

```

Listing 345: Umsortieren der Liste beim Klick auf eine Spalte

Fertig ist das Sortieren.

190 Knoten einer Ebene in einem TreeView-Steuerelement vertauschen

Für die Implementierung spezieller Features wie z.B. der Möglichkeit für den Anwender, die Knoten eines `TreeView`-Steuerelements definiert zu verschieben, ist es notwendig, zwei Knoten einer Ebene gegeneinander zu tauschen. Dazu können Sie aber nicht einfach nur die Referenzen der `TreeNode`-Objekte tauschen, wie es das folgende Beispiel zeigt, das versucht, die ersten beiden Unterknoten des ersten Hauptknotens des `TreeView` *bookTree* zu vertauschen:

```

TreeNode node = bookTree.Nodes[0].Nodes[0];
bookTree.Nodes[0].Nodes[2] = bookTree.Nodes[0].Nodes[1];
bookTree.Nodes[0].Nodes[1] = node;

```

Listing 346: Fehlerhaftes Vertauschen zweier TreeView-Knoten

Bei diesem Versuch gerät das TreeView-Steuerelement durcheinander und zeigt Einträge auf einmal doppelt an, obwohl die Anzahl der TreeNode-Objekte dieselbe ist wie vorher. Abbildung 133 zeigt eine Beispielanwendung mit zuvor eindeutigen Einträgen nach dem dreimaligen Aufruf der oben beschriebenen Tausch-Variante.



Abbildung 133: Fehlerhafte Darstellung eines TreeView-Steuerelements nach drei inkorrekten Tauschvorgängen

Das Problem liegt darin begründet, dass jeder Knoten Referenzen zu seinem Vorgänger und seinem Nachfolger verwaltet. Und diese müssten Sie ebenfalls umbiegen, was in der Praxis nur sehr schwierig zu programmieren ist.

Die Lösung des Problems verwendet die Methoden `RemoveAt` und `Insert` der `TreeNodeCollection`-Instanz, deren Unterknoten getauscht werden sollen.

Um sicherzustellen, dass der Tausch bei gültigen Indizes immer funktioniert, vergleicht die Methode `SwapTreeViewNodes` in Listing 347 zunächst, ob die beiden Indizes gleich sind (in dem Fall darf und muss nichts gemacht werden). Zur Sicherheit werden die Indizes anschließend noch so umgebogen, dass in `i1` der kleinere und in `i2` der größere gespeichert ist. Dann können die Knoten gefahrlos entfernt und wieder neu eingefügt werden.

Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System` und `System.Windows.Forms` importieren.

```
public static void SwapTreeViewNodes(TreeNodeCollection nodes, int index1,
    int index2)
{
    if (index1 != index2)
    {
        int i1 = Math.Min(index1, index2);
        int i2 = Math.Max(index1, index2);
```

Listing 347: Methode zum Vertauschen zweier TreeView-Knoten

```

        TreeNode node1 = nodes[i1];
        TreeNode node2 = nodes[i2];
        nodes.RemoveAt(i1);
        nodes.RemoveAt(i2-1);
        nodes.Insert(i1, node2);
        nodes.Insert(i2, node1);
    }
}

```

Listing 347: Methode zum Vertauschen zweier TreeView-Knoten (Forts.)

191 Einzelne Knoten eines TreeView-Steuerelements sortieren

Ein TreeView-Steuerelement erlaubt über die `Sorted`-Eigenschaft das generelle Sortieren nach den Knoten. Dabei werden allerdings immer alle Ebenen sortiert. Wenn Sie in einem TreeView-Steuerelement nur bestimmte Unterknoten bzw. nur bestimmte Ebenen sortieren wollen, müssen Sie etwas anders vorgehen.

In den C#-Newsgroups finden Sie dazu zurzeit (als ich diese Zeilen schrieb ...) recht wenig Anregungen und schon gar keine fertigen Lösungen. Die einzige dort angedeutete Lösung ist der Aufruf der API-Funktion `SendMessage` mit der Nachricht `TVM_SORTCHILDRENCB`, die genau dazu vorgesehen ist. Leider ist diese wahrscheinlich sehr schnelle Lösung nicht einfach umzusetzen, weswegen ich im Buch darauf verzichte. Nur damit Sie wissen, worum es dabei geht:

Bei dieser Lösung müssen Sie eine Callback-Methode zur Verfügung stellen, die drei `IntPtr`-Argumente übergeben bekommt. Die Adresse dieser Methode übergeben Sie über eine Struktur vom Typ `TVSORTCB` und einen Delegate neben dem Handle des zu sortierenden Knotens und der Nachricht an `SendMessage`. Der Knoten ruft dann zur Sortierung für jeweils zwei Unterknoten die Callback-Methode auf. In dieser vergleichen Sie die beiden ersten übergebenen `IntPtr`-Werte (der dritte ist für benutzerdefinierte Zwecke vorgesehen). Ist der linke größer als der rechte, muss die Methode einen Wert kleiner Null zurückgeben, ist der rechte größer als der linke, muss ein Wert größer Null zurückgegeben werden, sind beide gleich groß, muss die Methode Null zurückgeben. So weit, so gut. Leider werden in den Argumenten aber keine Referenzen auf die Knoten übergeben (was prinzipiell gar nicht möglich ist, da API-Funktionen keine Referenzen kennen), sondern benutzerdefinierte Daten. Und das ist das Problem, denn diese müssen Sie beim oder nach dem Füllen des TreeView in einer `TVITEM`-Struktur über eine `TVM_GETITEM`-Nachricht einlesen und über eine `TVM_SETITEM`-Nachricht in den Knoten zurückschreiben. Im Feld `lParam` der `TVITEM`-Struktur können Sie vor dem Schreiben einen `int`-Wert ablegen, der der Callback-Methode beim Sortieren übergeben wird. Dummerweise können Sie hier nicht einfach den Index des Knotens ablegen (was den Zugriff erleichtern würde), denn dieser wird beim Umsortieren ja verändert. Sie müssen sich einen Algorithmus überlegen, der aus den dargestellten Daten des Knotens einen sortierbaren `int`-Wert generiert. Und das ist meiner Ansicht nach fast unmöglich.

Deshalb habe ich eine einfachere Lösung entwickelt. Die Methode `SortTreeViewNodes` in Listing 348 sortiert eine `TreeNodeCollection` (die Sie über die `Node`-Eigenschaft eines Knotens

erreichen) über den guten alten Bubblesort. Das einzige Problem dabei ist, dass zum Vertauschen von zwei `TreeNode`-Objekten nicht einfach nur die Referenzen vertauscht werden können. Wenn Sie das machen, bringen Sie das `TreeView`-Steuerelement durcheinander (siehe Rezept 190). Um dies zu vermeiden ruft `SortTreeViewNodes` zum Tauschen einfach die entsprechenden Methoden der `TreeNode`-Objekte auf.

Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Windows.Forms.dll` referenzieren und die Namensräume `System` und `System.Windows.Forms` importieren.

```
public static void SortTreeViewNodes(TreeNodeCollection nodes)
{
    // Sortieren der Unterknoten über einen einfachen BubbleSort
    for (int i = 0; i < nodes.Count; i++)
    {
        for (int j = nodes.Count - 1; j > i; j--)
        {
            if (String.Compare(nodes[i].Text, nodes[j].Text, true) > 0)
            {
                // Tauschen
                TreeNode node1 = nodes[i];
                TreeNode node2 = nodes[j];
                nodes.RemoveAt(i);
                nodes.RemoveAt(j-1);
                nodes.Insert(i, node2);
                nodes.Insert(j, node1);
            }
        }
    }
}
```

Listing 348: Methode zum Sortieren einer `TreeNodeCollection`

Abbildung 134 zeigt eine Anwendung, die die zweite Ebene eines `TreeView`-Steuerelements über `SortTreeViewNodes` sortiert hat. Die Unterknoten des dritten Hauptknotens wurden in dieser Anwendung über einen zufällig ermittelten String erzeugt.



Abbildung 134: TreeView mit einer Sortierung der zweiten Ebene

Die Geschwindigkeit dieser Methode ist bei nicht allzu vielen Unterknoten einigermaßen gut. Bei 100 zufällig ermittelten (und damit sehr unsortierten) Unterknoten benötigte der Vorgang auf einem Pentium-4-Rechner mit 2,4 GHz etwa 0,08 Sekunden, wenn der übergeordnete Knoten zugeklappt war. War dieser jedoch aufgeklappt, benötigte die Methode schon 0,2 Sekunden. Bei 1000 Unterknoten war die Performance allerdings nicht mehr allzu gut. Hier benötigte `SortTreeViewNodes` stolze 34 Sekunden. Ein Vergleich mit einem ebenfalls implementierten `QuickSort` brachte übrigens sogar ein schlechteres Ergebnis (66 Sekunden für 1000 Unterknoten bei zugeklapptem Hauptknoten). Sie finden diese Sortierung im Beispiel zu diesem Rezept und im Repository. Die Messwerte habe ich übrigens über die Klasse `HighResStopClock` aus dem Rezept 58 ermittelt.

192 Ein datengebundenes DataGridView anpassen

Ein `DataGridView`-Steuerelement erzeugt seine Spalten und Zeilen per Voreinstellung nach den Informationen seiner Datenquelle selbst. Diese automatisch erzeugten Spalten entsprechen nicht immer dem, was eine Anwendung erfordert. Sie können die Spalten jedoch auch selbst bestimmen. Wie Sie dies programmieren, zeigt dieses Rezept. Außerdem erfahren Sie, wie Sie erreichen können, dass Zeilen, die einer bestimmten Bedingung entsprechen, mit einer definierten Hintergrundfarbe dargestellt werden.

Zum Kompilieren des Beispiel-Programms benötigen Sie neben den für Formular-Anwendungen üblichen Referenzen den Import der Namensräume `System`, `System.Drawing`, `System.Data`, `System.Data.SqlClient` und `System.Configuration`.

Anpassen der Spalten

Zur Demonstration der Anpassung der Spalten des `DataGridView` wird im `Load`-Ereignis eines Formulars, das mit einem `DataGridView`-Steuerelement namens *productGrid* ausgestattet ist, zunächst ein `DataSet` mit den Daten der `Products`-Tabelle der `Northwind`-Datenbank gefüllt:

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // Einlesen der Products-Tabelle der Northwind-Datenbank
    string server = ConfigurationSettings.AppSettings["server"];
    SqlConnection connection = new SqlConnection("Server=" + server +
        ";Database=Northwind;Trusted_Connection=Yes");
    try
    {
        connection.Open();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fehler beim Verbindungsaufbau zur Northwind-" +
            "Datenbank:" + ex.Message + "\r\nKontrollieren Sie die " +
            "Einstellungen in der Anwendungskonfiguration",
            Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
        Application.Exit();
    }
}
```

Listing 349: Füllen eines DataTable-Objekts in einem DataSet mit den Daten der Products-Tabelle

```
SqlDataAdapter dataAdapter = new SqlDataAdapter("SELECT ProductId, " +  
    "ProductName, Discontinued FROM Products", connection);  
DataSet dataSet = new DataSet();  
dataAdapter.Fill(dataSet, "Products");
```

Listing 349: Füllen eines DataTable-Objekts in einem DataSet mit den Daten der Products-Tabelle

Nun können Sie die Spalten des DataGrid definieren. Dazu erzeugen Sie ein `DataGridTableStyle`-Objekt (aus dem Namensraum `System.Windows.Forms`). Dieses wird später der `TableStyles`-Auflistung des `DataGrid`-Steuerelements zugewiesen. Diese Auflistung verwaltet ein oder mehrere dieser Objekte, die jedes für sich den Stil einer der eventuell mehreren im DataGrid dargestellten Tabellen definieren. Wenn Sie selbst kein `DataGridTableStyle`-Objekt für eine Tabelle erzeugen, definieren und zuweisen, erzeugt das DataGrid bei der Anzeige der Daten ein Default-Objekt.

Nach dem Erstellen des `DataGridTableStyle`-Objekts ist es wichtig, dass Sie über die Eigenschaft `MappingName` die Tabelle definieren, für die dieses Objekt gelten soll. Vergessen Sie dies, wird das Objekt einfach nicht verwendet.

Die einzelnen Spalten definieren Sie dann zunächst über Instanzen der Klassen `DataGridTextBoxColumn` (Textspalte) und `DataGridBoolColumn` (CheckBox-Spalte). Über verschiedene Eigenschaften können Sie die Spalten einstellen. Wichtig sind die folgenden:

- ▶ `MappingName`: Hier geben Sie den Namen der Spalte in der Datenquelle an, deren Daten dargestellt werden sollen,
- ▶ `HeaderText`: bestimmt den Text der Überschrift,
- ▶ `Width`: bestimmt die Breite der Spalte,
- ▶ `Format`: bestimmt das Format der Spalte,
- ▶ `ReadOnly`: legt fest, ob die Spalte schreibgeschützt ist,
- ▶ `AllowNull`: bestimmt, ob null-Werte in der Spalte erlaubt sind.

Die einzelnen Spalten fügen Sie dann der `GridColumnStyles`-Auflistung des `DataGridTableStyle`-Objekts hinzu und dieses schließlich der `TableStyles`-Auflistung des `DataGrid`-Steuerelements.

```
// DataGridTableStyle-Objekt erzeugen und initialisieren  
DataGridTableStyle tableStyle = new DataGridTableStyle();  
tableStyle.MappingName = "Products";  
  
// Spalten des DataSet definieren  
DataGridTextBoxColumn coll = new DataGridTextBoxColumn();  
coll.HeaderText = "Artikel-Nr";  
coll.MappingName = "ProductId";
```

Listing 350: Definieren der Spalten eines DataGrid und Anbinden des DataSet

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

coll.Width = 70;
coll.ReadOnly = true;
tableStyle.GridColumnStyles.Add(coll);

DataGridTextBoxColumn col2 = new DataGridTextBoxColumn();
col2.HeaderText = "Artikelname";
col2.MappingName = "ProductName";
col2.Width = 150;
tableStyle.GridColumnStyles.Add(col2);

DataGridBoolColumn col3 = new DataGridBoolColumn();
col3.HeaderText = "Auslaufartikel";
col3.MappingName = "Discontinued";
col3.AllowNull = false; // Dreiwertigkeit unterbinden
col3.Width = 80;
tableStyle.GridColumnStyles.Add(col3);

// DataGridTableStyle-Instanz dem DataGrid zuweisen
this.productsGrid.TableStyles.Add(tableStyle);

// DataSet an das DataGrid anbinden
this.productsGrid.DataMember = "Products";
this.productsGrid.DataSource = dataSet;
}

```

Listing 350: Definieren der Spalten eines DataGrid und Anbinden des DataSet (Forts.)

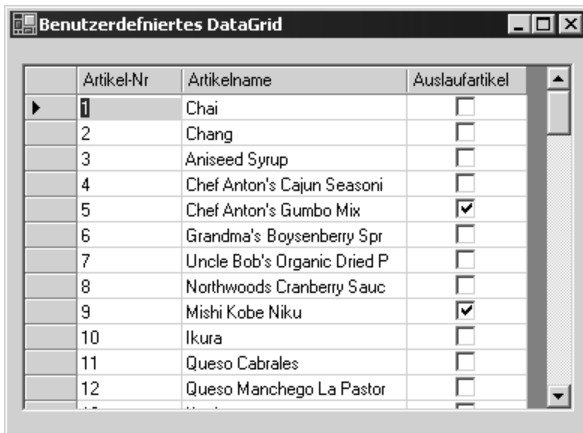


Abbildung 135: Ein benutzerdefiniertes DataGrid

Auf diese Weise können Sie die einzelnen Spalten schon sehr gut einstellen. Im Gegensatz zum Default-Verhalten ist in diesem Beispiel die erste Spalte schreibgeschützt und erlaubt die dritte Spalte keine Nullwerte (womit der dritte Zustand der `CheckBox` entfällt). Außerdem ist natürlich die Breite der einzelnen Spalten definiert.

Sie können aber auch eigene, besondere Spalten realisieren, wie z. B. eine, die eine Datenliste in einer `ComboBox` zur Auswahl anbietet. Dazu müssen Sie allerdings eigene Klassen implementieren, die Sie von der abstrakten Basisklasse `DataGridColumnStyle` oder von einer der konkreten Klassen ableiten können. Ein Beispiel dazu finden Sie im folgenden Abschnitt und im Rezept 193.

Bedingungsabhängiges Anpassen der Zellenfarbe

Die Farbe der einzelnen Zeilen eines `DataGrid` können Sie, wie Sie ja wahrscheinlich wissen, über die Eigenschaften `BackColor` und `AlternatingBackColor` beeinflussen. Setzen Sie `BackColor` z. B. auf `Color.White` und `AlternatingBackColor` auf `Color.Silver`, erscheint jede zweite Zeile weiß und jede andere hellgrau.

Wenn Sie aber Zeilen oder einzelne Zellen anders darstellen wollen, wenn diese einer bestimmten Bedingung entsprechen, müssen Sie etwas mehr programmieren.

Die Grundidee zur Veränderung einzelner Zellen ist die Implementierung einer eigenen Klasse für die Spalte. Wenn Ihnen die Grundfunktionalität der vordefinierten Klassen entgegenkommt, können Sie einfach diese als Basisklasse verwenden. Über das Überschreiben verschiedener Methoden können Sie das Verhalten der Spalte nun sehr gut beeinflussen.

Ich zeige hier nur, wie Sie die `Paint`-Methode überschreiben, um die Zellenfarbe zu ändern. Ich denke, mit dem hier erworbenen Wissen können Sie alle weiteren Anpassungen relativ problemlos implementieren. Über diese Technik ist es z. B. kein Problem, in einer Spalte eines `DataGrid` ein Bild anzuzeigen. Leiten Sie Ihre Klasse dann von `DataGridColumnStyle` ab und zeichnen Sie das Bild, das Sie gegebenenfalls aus der Datentabelle auslesen, in der überschriebenen `Paint`-Methode über die `DrawImage`-Methode des übergebenen `Graphics`-Objekts.

Einer Variante der `Paint`-Methode werden alle Informationen übergeben, die Sie benötigen. Diese sollten Sie überschreiben, damit Sie alles, was Sie benötigen, zur Verfügung haben.

Listing 351 zeigt, wie Sie dies umsetzen. Das Beispiel leitet eine neue Klasse von der Klasse `DataGridTextBoxColumn` ab und überschreibt die `Paint`-Methode. Über die Eigenschaft `List` des übergebenen `CurrencyManager`-Objekts (das über eine Liste von `Binding`-Objekten die Datenbindung für das `DataGrid`-Steuerelement verwaltet) ermittelt `Paint` eine Referenz auf das dem `DataGrid` zugrunde liegende (interne) `DataRowView`-Objekt (das `DataGrid`-Objekt verwendet für die Darstellung der Daten das `DefaultView`-Objekt von gebundenen `DataTable`-Objekten). Hier müssen Sie etwas aufpassen, wenn Sie das `DataGrid` nicht über ein `DataSet`-, `DataTable`- oder `DataRowView`-Objekt an Daten binden. Der von der `List`-Eigenschaft referenzierte Datentyp ist dann u. U. ein anderer.

Über dieses `DataRowView`-Objekt liest `Paint` dann den Wert der Spalte `Discontinued` aus. Ist dieser `true` (was bedeutet, dass der Artikel ein Auslaufartikel ist), definiert `Paint` die beiden für den Hintergrund und den Vordergrund übergebenen `Brush`-Objekte neu. Schließlich wird noch die geerbte `Paint`-Methode (mit den gegebenenfalls umdefinierten `Brush`-Objekten) aufgerufen, damit die Zelle gezeichnet wird.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

private class ExtDataGridTextBoxColumn: DataGridViewTextBoxColumn
{
    /* Überschreiben der Paint-Methode */
    protected override void Paint(Graphics g, Rectangle bounds,
        CurrencyManager source, int rowNum, Brush backBrush,
        Brush foreBrush, bool alignToRight)
    {
        // Überprüfen, ob es sich um einen Auslaufartikel handelt
        DataView dataView = (DataView)source.List;
        bool discontinued = (bool)dataView[rowNum]["Discontinued"];
        if (discontinued)
        {
            backBrush = new SolidBrush(Color.Red);
            foreBrush = new SolidBrush(Color.Yellow);
        }
        // Aufruf der geerbten Methode mit den eventuell geänderten Pinseln
        base.Paint(g, bounds, source, rowNum, backBrush,
            foreBrush, alignToRight);
    }
}

```

Listing 351: Bedingungsabhängiges Anpassen der Zellenfarbe

Diese Klasse müssen Sie nun noch bei der Definition der DataGrid-Spalten an Stelle der DataGridViewTextBoxColumn-Klasse einsetzen. Damit ist Ihre benutzerdefinierte Anpassung der Spalte beendet. Listing 352 zeigt den relevanten Ausschnitt aus dem Load-Ereignis des Formulars:

```

private void StartForm_Load(object sender, System.EventArgs e)
{
    ...

    DataGridViewTextBoxColumn col1 = new ExtDataGridTextBoxColumn();
    col1.HeaderText = "Artikel-Nr";
    col1.MappingName = "ProductId";
    col1.Width = 70;
    col1.ReadOnly = true;
    tableStyle.GridColumnStyles.Add(col1);

    DataGridViewTextBoxColumn col2 = new ExtDataGridTextBoxColumn();
    col2.HeaderText = "Artikelname";
    col2.MappingName = "ProductName";
    col2.Width = 150;
    tableStyle.GridColumnStyles.Add(col2);

    ...
}

```

Listing 352: Änderungen bei der Definition der DataGrid-Spalten zur Darstellung der eigenen Spalte

Abbildung 136 zeigt das Ergebnis:

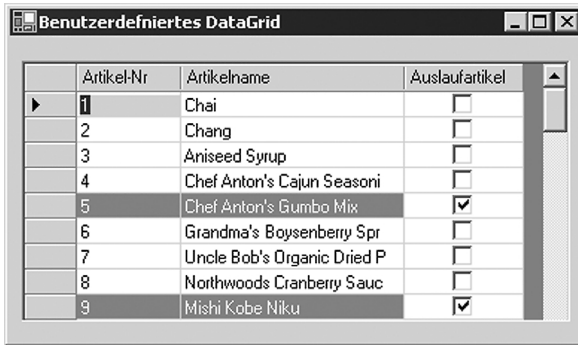


Abbildung 136: Ein benutzerdefiniertes DataGrid mit bedingungsabhängiger Zellenfarbe

193 Ein DataGrid mit ComboBox-Spalten

Das DataGrid-Steuerelement bietet schon recht flexible Möglichkeiten, Daten tabellenförmig darzustellen und dem Anwender zum Bearbeiten zur Verfügung zu stellen. Ein für die Praxis wichtiges Feature fehlt jedoch: Per Voreinstellung können Sie dem DataGrid keine Spalten zuweisen, die eine Liste von Werten in einer ComboBox zur Verfügung stellen. Sie können die einzelnen Spalten zwar über ein DataGridTableStyle-Objekt selbst definieren, dazu stehen Ihnen aber nur die vordefinierten Spaltentypen DataGridBooleanColumn und DataGridTextBoxColumn zur Verfügung.

Wenn Sie ein DataGrid erzeugen wollen wie in Abbildung 137, müssen Sie selbst programmieren.

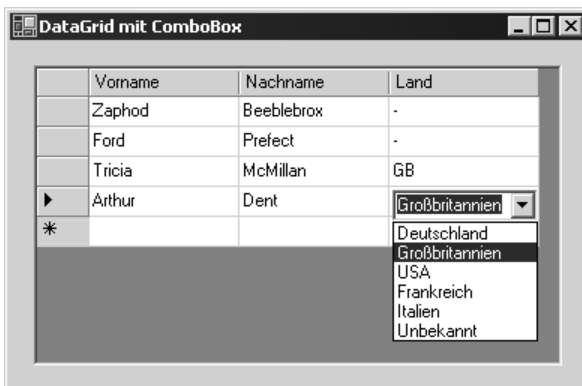


Abbildung 137: Ein DataGrid mit einer ComboBox-Spalte

Das `DataGrid` in Abbildung 137 ist bewusst praxisnah gestaltet. In diesem virtuellen Szenario stammen die Daten des Grid aus einer Datenbank-Tabelle. Die Spalte *Land* verwaltet eine Land-Kurzbezeichnung (die Id des Landes). Die Länderdaten sind in einer separaten Datenbanktabelle gespeichert. Die `ComboBox` wurde mit diesen Länderdaten gefüllt und stellt in ihrer Liste an Stelle der Land-Id den Land-Namen dar. Im `DataGrid` wird aber immer die Land-Id gespeichert, damit die Aktualisierung der Datenbanktabelle keine Probleme bereitet.

Eine mögliche Lösung beschreibt Microsoft im Knowledge-Base-Artikel 323167 (support.microsoft.com/?id=323167). Bei dieser Lösung werden verschiedene Ereignisse des `DataGrid` ausgewertet um eine `ComboBox`, die der `Controls`-Auflistung des `DataGrid` zugewiesen wurde, bei Bedarf in der aktuellen Zelle zu positionieren und sichtbar bzw. unsichtbar zu schalten. Diese Lösung ist für die Praxis zu aufwändig, weil Sie die Ereignisse immer wieder neu programmieren müssen.

Besser wäre es, einen Spaltentyp für eine `ComboBox`-Spalte zu besitzen und das `DataGrid` damit initialisieren zu können. Die Lösung dazu basiert auf einem Artikel von *George Shepherd* (www.syncfusion.com/faq/winforms/search/480.asp), wurde aber von mir vereinfacht.

Dazu benötigen Sie zunächst eine Klasse, die von der Klasse `DataGridTextBoxColumn` abgeleitet ist. Diese Klasse stellt eine normale Spalte im `DataGrid` dar. Im Anzeigemodus wird der Spaltenwert als Text ausgegeben, im Editiermodus legt ein `DataGridTextBoxColumn`-Objekt eine `TextBox` über die Zelle, sodass der Benutzer den Text der Zelle ändern kann. Ähnliches wird unsere Klasse ausführen, nur dass die `TextBox` noch von einem `ComboBox`-Objekt überlagert wird.

Die Verwendung der Klasse soll folgendermaßen möglich sein:

```
private void StartForm_Load(object sender, System.EventArgs e)
{
    // DataTable-Objekte für die Daten erzeugen und initialisieren
    DataTable personTable = new DataTable();
    personTable.Columns.Add("FirstName", typeof(string));
    personTable.Columns.Add("LastName", typeof(string));
    personTable.Columns.Add("Country", typeof(string));
    personTable.Rows.Add(new object[] { "Zaphod", "Beeblebrox", "-" });
    personTable.Rows.Add(new object[] { "Ford", "Prefect", "-" });
    personTable.Rows.Add(new object[] { "Tricia", "McMillan", "GB" });
    personTable.Rows.Add(new object[] { "Arthur", "Dent", "GB" });

    DataTable countryTable = new DataTable();
    countryTable.Columns.Add("Id", typeof(string));
    countryTable.Columns.Add("Country", typeof(string));
    countryTable.Rows.Add(new object[] { "DE", "Deutschland" });
    countryTable.Rows.Add(new object[] { "GB", "Großbritannien" });
    countryTable.Rows.Add(new object[] { "US", "USA" });
    countryTable.Rows.Add(new object[] { "FR", "Frankreich" });
    countryTable.Rows.Add(new object[] { "IT", "Italien" });
}
```

Listing 353: Initialisierung eines `DataGrid`-Steuerelements mit zwei Text- und einer `DataGridComboBoxColumn`-Spalte

```
countryTable.Rows.Add(new object[] { "-", "Unbekannt" });

// Die Spalten des DataGrid definieren
DataGridTableStyle tableStyle = new DataGridTableStyle();

// Eine TextBox-Spalte
DataGridTextBoxColumn col1 = new DataGridTextBoxColumn();
col1.MappingName = "FirstName";
col1.HeaderText = "Vorname";
col1.Width = 100;
tableStyle.GridColumnStyles.Add(col1);

// Noch eine TextBox-Spalte
DataGridTextBoxColumn col2 = new DataGridTextBoxColumn();
col2.MappingName = "LastName";
col2.HeaderText = "Nachname";
col2.Width = 100;
tableStyle.GridColumnStyles.Add(col2);

// Eine Spalte mit der DataGridComboBoxColumn
DataGridComboBoxColumn col3 = new DataGridComboBoxColumn();
col3.MappingName = "Country";
col3.HeaderText = "Land";
col3.Width = 100;
tableStyle.GridColumnStyles.Add(col3);

// Festlegen der bevorzugten Zeilenhöhe auf die Höhe der ComboBox
tableStyle.PreferredRowHeight = col3.ComboBox.Height;

// Füllen der ComboBox
col3.ComboBox.ValueMember = "Id";
col3.ComboBox.DisplayMember = "Country";
col3.ComboBox.DataSource = countryTable;

// Tabellenstil an das DataGrid übergeben
this.personGrid.TableStyles.Add(tableStyle);

// DataTable an das DataGrid anbinden
this.personGrid.DataSource = personTable;
}
```

Listing 353: Initialisierung eines DataGrid-Steuerelements mit zwei Text- und einer DataGridComboBoxColumn-Spalte (Forts.)

Das Beispiel erzeugt im Load-Ereignis eines Formulars zwei DataTable-Objekte, die als Datenquelle für das DataGrid und die ComboBox dienen sollen. In der Praxis würden diese Daten aus einer Datenbank ausgelesen werden, darauf wollte ich hier aber verzichten. Über ein DataGridTableStyle-Objekt werden dann die Spalten des DataGrid definiert. Neben den ersten beiden Text-Spalten wird für die dritte Spalte die Klasse DataGridComboBoxColumn verwendet, um eine Spalte mit einer ComboBox zu erzeugen. Diese Klasse muss natürlich noch entwickelt werden. Über die Eigenschaft ComboBox des DataGridComboBoxColumn-Objekts wird die ComboBox

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

mit Werten gefüllt, wobei hier das `DataTable`-Objekt `countryTable` als Datenquelle verwendet wird. Um nah an der Praxis zu sein verwaltet dieses Objekt neben der Id eines Landes auch den Land-Namen. In der `ComboBox` wird der Name angezeigt, die Id wird als Wert des `ComboBox`-Eintrags verwaltet. So können Sie der `ComboBox` problemlos die für Datenbanken typischen Lookup-Tabellen zuweisen.

Schließlich wird das `DataTableStyle`-Objekt dem `DataGrid` zugewiesen und dieses über das andere `DataTable`-Objekt `personTable` gefüllt.

Das Ergebnis haben Sie ja bereits in Abbildung 137 gesehen. Nun muss nur noch die Klasse `DataGridComboBoxColumn` implementiert werden. Diese Klasse wird von `DataGridTextBoxColumn` abgeleitet um deren Funktionalität zu erben. Zunächst werden einige Eigenschaften deklariert. Über die Eigenschaft `ComboBox` können Sie, wie es ja bereits in Listing 353 gezeigt wurde, von außen auf die `ComboBox` zugreifen und diese füllen. Die anderen Eigenschaften werden intern verwendet (und später erläutert). `DataGridComboBoxColumn` implementiert zudem ein Ereignis `ComboValueChanged`, das aufgerufen wird, wenn der Wert der `ComboBox` geändert wird.

Zum Kompilieren der `DataGridComboBoxColumn`-Klasse müssen Sie die Namensräume `System`, `System.Drawing` und `System.Windows.Forms` importieren.

```
public class DataGridComboBoxColumn: DataGridTextBoxColumn
{
    /* Eigenschaften */
    public ComboBox ComboBox = null;
    private bool inEditMode;
    private CurrencyManager currencyManager = null;
    private int currentRow;
    private bool dontHandleSelectedItemChanged = false;

    /* Delegate und Ereignis für das ComboValueChanged-Ereignis */
    public delegate void ComboValueChanged(int changingRow,
        object newValue);
    public event ComboValueChanged ComboValueChangedEvent;
}
```

Listing 354: Beginn der Klasse `DataGridComboBoxColumn`

Im Konstruktor wird die `ComboBox` erzeugt. Da verschiedene Ereignisse der `ComboBox` ausgewertet werden müssen, werden diese den Ereignisbehandlungsmethoden zugewiesen, die weiter unten, ab Listing 355, deklariert werden:

```
public DataGridComboBoxColumn()
{
    // ComboBox erzeugen und die Ereignisbehandlungs-Methoden zuweisen
    this.ComboBox = new ComboBox();
    this.ComboBox.Leave +=
        new EventHandler(this.ComboBoxLeaveHandler);
    this.ComboBox.SelectedIndexChanged +=
```

Listing 355: Der Konstruktor

```

        new EventHandler(this.ComboBoxSelectedIndexChangedHandler);
        this.ComboBox.SelectionChangeCommitted +=
            new EventHandler(this.ComboBoxSelectionChangeCommittedHandler);
    }

```

Listing 355: Der Konstruktor (Forts.)

Nun wird die geschützte `Edit`-Methode der `DataGridTextBoxColumn`-Klasse überschrieben. Diese Methode wird automatisch dann aufgerufen, wenn die Spalte in den Editiermodus versetzt wird. Die überschriebene `Edit`-Methode ruft zuerst die geerbte `Edit`-Methode auf und merkt sich dann in den privaten Eigenschaften `currentRow` und `currencyManager` die aktuelle Zeile und das übergebene `CurrencyManager`-Objekt für die spätere Verwendung in den Methoden für das `SelectedIndexChanged`- und das `Leave`-Ereignis der `ComboBox`. Dann wird die `ComboBox` so initialisiert, dass diese an der Position und in der Größe der `TextBox` erscheint, und der Text der `TextBox` in den aktuellen Wert der `ComboBox` geschrieben. Damit beim Schreiben das `SelectedItemChanged`-Ereignis der `ComboBox` nicht abgearbeitet wird, setzt `Edit` die private Eigenschaft `dontHandleSelectedItemChanged` auf `true`. Diese Eigenschaft wird in der Behandlungsmethode des `SelectedItemChanged`-Ereignisses abgefragt. Dann wird nur noch die `TextBox` unsichtbar, die `ComboBox` sichtbar geschaltet, nach vorne geholt und der Fokus auf die `ComboBox` gesetzt.

```

protected override void Edit(CurrencyManager source, int rowNum,
    System.Drawing.Rectangle bounds, bool readonly, string instantText,
    bool cellIsVisible)
{
    base.Edit(source, rowNum, bounds, readonly, instantText,
        cellIsVisible);

    // Merken der aktuellen Zeile und des CurrencyManager-Objekts
    this.currentRow = rowNum;
    this.currencyManager = source;

    // Übergeben der Eigenschaften der TextBox der Spalte an die ComboBox
    this.ComboBox.Parent = this.TextBox.Parent;
    this.ComboBox.Location = this.TextBox.Location;
    this.ComboBox.Size = new Size(this.TextBox.Size.Width,
        this.ComboBox.Size.Height);
    // Abarbeitung des SelectedItemChanged-Ereignisses vermeiden
    this.dontHandleSelectedItemChanged = true;
    // Text übergeben
    this.ComboBox.SelectedValue = this.TextBox.Text;
    // Abarbeitung des SelectedItemChanged-Ereignisses wieder erlauben
    this.dontHandleSelectedItemChanged = false;

    // TextBox unsichtbar, ComboBox sichtbar schalten, die ComboBox nach
    // vorne holen und den Fokus darauf setzen

```

Listing 356: Überschreiben der `Edit`-Methode um die `ComboBox` zu initialisieren

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        this.TextBox.Visible = false;
        this.ComboBox.Visible = true;
        this.ComboBox.BringToFront();
        this.ComboBox.Focus();
    }

```

Listing 356: Überschreiben der Edit-Methode um die ComboBox zu initialisieren (Forts.)

Die `Commit`-Methode, die laut der (mageren) Dokumentation eine »Aufforderung zum Abschließen einer Bearbeitungsprozedur« ist, wird ebenfalls überschrieben. In dieser Methode wird der Wert der `ComboBox` an die `TextBox` übergeben, damit das `DataGrid` den ausgewählten Wert speichert.

`Commit` wird auf zwei Arten aufgerufen: Zum einen nachdem der Anwender in der `ComboBox` die Return-Taste betätigt hat. Dazu ist in der Klasse `DataGridComboBoxColumn` allerdings nichts weiter implementiert, da der Aufruf implizit von der `DataGridTextBoxColumn`-Klasse ausgeführt wird (die die Tastenbetätigungen abfängt). Der andere ebenfalls implizite Aufruf erfolgt nach dem Verlassen der `ComboBox`.

Hinweis

Die Eigenschaft `inEditMode`, die in `Commit` abgefragt und zurückgesetzt und in der Methode für das `BoxSelectionChangeCommitted`-Ereignis der `ComboBox` gesetzt wird, ist übrigens nur aus einem Grunde implementiert: `Commit` wird zum einen aus mir unklaren Gründen mehrfach aufgerufen, wenn der Anwender die `ComboBox` verlässt, aber deren Wert nicht geändert hat. In meinem Fall waren das immer drei Aufrufe. `inEditMode` verhindert damit schon einmal das mehrfache Übergeben des Textes. Eine echte Daseinsberechtigung hat diese Eigenschaft aber für den Fall, dass der Anwender den Wert der `ComboBox` nicht ändert und in eine andere Zeile wechselt. In diesem Fall bezieht sich der letzte Aufruf (d.h. das Argument `rowNum`) nämlich auf die neue Zeile. Würde `inEditMode` nicht abgefragt werden, würde `Commit` den Text der Spalte in dieser Zeile gleich noch mit ändern.

```

protected override bool Commit(CurrencyManager dataSource, int rowNum)
{
    if (this.inEditMode)
    {
        this.inEditMode = false;

        // Den Wert der ComboBox in die TextBox schreiben
        base.SetColumnValueAtRow(dataSource, rowNum,
            this.ComboBox.SelectedValue);
    }
    return true;
}

```

Listing 357: Überschreiben der Commit-Methode zur Übergabe des ausgewählten Eintrags an die TextBox

Die Ereignisbehandlungsmethode für das `SelectionChangeCommitted`-Ereignis der `ComboBox` (das aufgerufen wird, wenn die Eingabe des Anwenders in die `ComboBox`-Liste übernommen wird) spielt eine zentrale Rolle in der Klasse `DataGridViewComboBoxColumn`. Immer dann, wenn durch den Aufruf dieser Methode eine wirkliche Änderung des Zustands der `ComboBox` erkannt wird, wird die `DataGridView`-Spalte in den Editiermodus versetzt, damit diese Änderungen annehmen kann. Gleichzeitig wird die Eigenschaft `inEditMode` auf `true` gesetzt, damit die Methoden, die die Auswahl des Anwenders in die `TextBox` der Spalte zurückschreiben, ihre Arbeit machen:

```
private void ComboBoxSelectionChangeCommittedHandler(object sender,
    System.EventArgs e)
{
    this.inEditMode = true;
    base.ColumnStartedEditing((Control)sender);
}
```

Listing 358: Ereignisbehandlungsmethode für das `SelectionChangeCommitted`-Ereignis der `ComboBox`

In der Ereignisbehandlungsmethode für das `Leave`-Ereignis der `ComboBox` wird dafür gesorgt, dass die Spalte neu gezeichnet wird, wenn die Klasse sich im Editiermodus befindet. Die `ComboBox` wird dann wieder unsichtbar geschaltet. Das Verlassen der `ComboBox` führt übrigens wie bereits gesagt automatisch zum Aufruf von `Commit`.

```
private void ComboBoxLeaveHandler(object sender, System.EventArgs e)
{
    if (this.inEditMode)
        // Die Spalte auffordern, ihre TextBox neu zu zeichnen
        base.Invalidate();

    // ComboBox verstecken
    this.ComboBox.Hide();
}
```

Listing 359: Ereignisbehandlungsmethode für das `Leave`-Ereignis der `ComboBox`

Die Ereignisbehandlungsmethode für das `SelectedIndexChanged`-Ereignis der `ComboBox` ist schließlich nur deswegen implementiert, damit das eigene `ComboBoxValueChanged`-Ereignis aufgerufen werden kann:

```
private void ComboBoxSelectedIndexChangedHandler(object sender,
    System.EventArgs e)
{
    // Wenn der Index des selektierten Eintrags sich in der ComboBox
    // ändert: Das eigene ComboBoxValueChanged-Ereignis aufrufen
}
```

Listing 360: Ereignisbehandlungsmethode für das `SelectedIndexChanged`-Ereignis der `ComboBox`

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benut

Multi-
mediaAutom-
tionSons-
tiges

```

        if (this.dontHandleSelectedItemChanged == false)
        {
            if (this.ComboValueChangedEvent != null)
                this.ComboValueChangedEvent(this.currentRow,
                this.ComboBox.Text);
        }
    }
}

```

Listing 360: Ereignisbehandlungsmethode für das `SelectedIndexChanged`-Ereignis der `ComboBox`

194 Drag & Drop von Dateien und Ordnern

Für Anwendungen, die Eingaben von Datei- oder Ordnernamen erwarten, ist es sinnvoll, ein Drag & Drop vom Explorer aus zu ermöglichen. Dieses Problem ist einfach gelöst, aber nicht unbedingt so klar, dass das Rezept hier sinnlos wäre ☺.

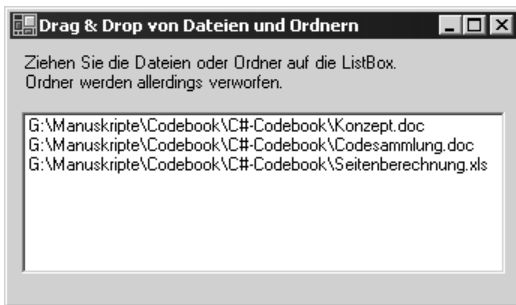


Abbildung 138: Beispielanwendung für das Drag & Drop von Dateien und Ordnern

In der Beispielanwendung sollen Dateien und Ordner auf die `ListBox` mit Namen *fileList* gezogen werden können. Dazu stellen Sie zunächst die Eigenschaft `AllowDrop` der `ListBox` auf `true`.

Im Ereignis `DragEnter` der `ListBox` überprüfen Sie über die `GetDataPresent`-Methode des im `DragEventArgs`-Argument übergebenen `IDataObject`-Objekts, ob Dateien oder Ordner gezogen werden. Dazu übergeben Sie die Konstante `DataFormats.FileDrop`. In diesem Fall setzen Sie den Effekt auf `DragDropEffects.Copy`, da Sie im Prinzip nur das »Kopieren« der Dateien und nicht das Verschieben zulassen (das Sie natürlich implementieren müssten):

```

private void fileList_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
        e.Effect = DragDropEffects.Copy;
    else
        e.Effect = DragDropEffects.None;
}

```

Listing 361: Setzen des Drag & Drop-Effekts

Im Ereignis `DragDrop` werten Sie nun die gezogenen Daten aus. Dazu rufen Sie die `GetData`-Methode des mit dem `DragEventArgs`-Objekt in dessen `Data`-Eigenschaft übergebenen `IDataObject`-Objekts aus. Am ersten Argument können Sie ein `Type`-Objekt übergeben (was für gezogene Dateien und Ordner aber nicht existiert). Alternativ können Sie auch einen `String` übergeben, der den Typ kennzeichnet. Und hier können Sie direkt `DataFormats.FileDrop` übergeben, weil diese Konstante den passenden `String` (»FileDrop«) verwaltet. Am zweiten Argument definieren Sie mit `false`, dass die Daten nicht in das angegebene Format konvertiert werden sollen, weil diese ja bereits in diesem Format vorliegen.

Sie erhalten dann ein `String`-Array mit den Datei- und/oder Ordnernamen. Alles Weitere bleibt dann Ihnen überlassen. Das Beispiel ermittelt über ein `FileInfo`-Objekt (aus dem Namensraum `System.IO`), ob es sich um eine Datei handelt, und fügt dieses Objekt dann einfach an die Liste an. Da die `ToString`-Methode eines `FileInfo`-Objekts den vollen Dateinamen zurückgibt, wird dieser in der Liste angezeigt. Der Vorteil dieser Vorgehensweise ist, dass Sie bei der Auswertung der Liste über die referenzierten `FileInfo`-Objekte die vollen Informationen zu den Dateien zur Verfügung haben.

```
private void fileList_DragDrop(object sender, DragEventArgs e)
{
    // Dateien aus den gezogenen Daten auslesen
    string[] filesNames = (string[])e.Data.GetData(DataFormats.FileDrop,
        false);
    foreach (string fileName in filesNames)
    {
        // FileInfo-Objekt erzeugen
        FileInfo fi = new FileInfo(fileName);
        if (fi.Exists)
        {
            // Wenn es sich nicht um einen Ordner handelt: FileInfo-Objekt der
            // Liste anfügen
            this.fileList.Items.Add(fi);
        }
    }
}
```

Listing 362: Auslesen der gezogenen Dateinamen

195 Ein Ordner-Dialog

Das .NET Framework bietet bereits ab der Version 1.0 die Möglichkeit, einen Ordner-Dialog anzuzeigen (wenn auch etwas versteckt). Die entsprechende Klasse heißt `FolderNameEditor` und wird im Namensraum `System.Windows.Forms.Design` verwaltet. Die Verwendung ist einfach, aber leider auch etwas eingeschränkt, da Sie keine Eigenschaften einstellen können:

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
FolderNameEditor browser = new FolderNameEditor();  
string folderPath = (string)browser.EditValue(null, null);  
if (folderPath != null)  
    Console.WriteLine(folderPath);  
else  
    Console.WriteLine("Abbruch");
```

Listing 363: Ein Ordner-Dialog über die Klasse *FolderNameEditor*

Die Argumente der *EditValue*-Methode werden für interne Zwecke verwendet, für die die *FolderNameEditor*-Klasse eigentlich auch vorgesehen ist.

Wie Sie dem Beispiel entnehmen können, wird ein Benutzer-Abbruch über die Rückgabe von *null* ausgewertet. Das Ergebnis des Aufrufs der *EditValue*-Methode zeigt Abbildung 139.



Abbildung 139: Ein Ordner-Dialog über die Klasse *FolderNameEditor*

Das .NET Framework 1.1 bietet über die Klasse *FolderBrowserDialog* (aus dem Namensraum *System.Windows.Forms*) eine ähnliche, aber erweiterte Möglichkeit. Hier können Sie über die Eigenschaft *SelectedPath* den voreingestellten Ordner einstellen. Neben normalen Ordnerangaben können Sie auch die speziellen Windows-Ordner über die Aufzählung *Environment.SpecialFolder* angeben. Die Eigenschaft *ShowNewFolderButton* bestimmt, ob der Schalter zum Erstellen eines neuen Ordners sichtbar ist. Über die Eigenschaft *Description* können Sie einen Text angeben, der über der Ordnerliste erscheint. Die *ShowDialog*-Methode öffnet den Dialog dann wie gewohnt:

```
FolderBrowserDialog fbd = new FolderBrowserDialog();
fbd.SelectedPath = "C:\\Programme";
// fbd.RootFolder = Environment.SpecialFolder.Personal;
fbd.ShowNewFolderButton = false;
fbd.Description = "Wählen Sie einen Ordner aus:";

if (fbd.ShowDialog() == DialogResult.OK)
    Console.WriteLine(fbd.SelectedPath);
else
    Console.WriteLine("Abbruch");
```

Listing 364: Ein Ordner-Dialog über die Klasse *FolderBrowserDialog*

ACHTUNG

Die Klasse *FolderBrowserDialog* hat aber in der Version 1.1 des .NET Framework ein massives Problem unter Windows ab der Version 2000: Die Auswahl von Ordnern mit einer Pfadlänge von mehr als 128 Zeichen führt in vielen Fällen zu Ausnahmen und eventuell auch zu Abstürzen. Manchmal wird bereits bei der Auswahl des Ordners eine nicht behandelte Ausnahme gemeldet (die auch nicht abfangbar ist), manchmal erscheint die Ausnahme erst beim Auslesen des Ordnersnamens. In einigen Fällen ist die Auswahl auch (scheinbar) erfolgreich. Das Problem liegt wahrscheinlich daran, dass der Programmierer dieser Klasse nur 260 Bytes für einen String-Puffer reserviert hat, der aber auf Systemen ab Windows 2000 aus 16-Bit-Unicode-Zeichen besteht). Von der Verwendung dieser Klasse kann ich im Moment nur dringend abraten, da das undefinierte Ausnahme-Verhalten darauf hindeutet, dass der Speicher des Programms unkontrolliert überschrieben wird. Für *FolderNameEditor* gilt das übrigens nicht.

Wenn Sie diese Einschränkungen vermeiden wollen, können Sie den originalen Windows-Ordner-Dialog entweder recht kompliziert über API-Aufrufe und die Integration von COM-Schnittstellen verwenden. Sie können aber auch einfach die fertige Klasse *ShellFolderBrowser* von *The Code Project* einsetzen, die Sie an der Adresse www.codeproject.com/cs/miscctrl/folder-browser.asp oder auf der Buch-CD finden.

Diese Klasse kapselt den Aufruf der API-Funktion *SHBrowseForFolder*, die leider nicht allzu einfach anzuwenden ist (weswegen ich für das Buch auf eine Erläuterung verzichte). Die Anwendung der Klasse ist aber recht einfach:

```
ShellFolderBrowser folderBrowser = new ShellFolderBrowser();
folderBrowser.BrowseFlags = BrowseFlags.ReturnOnlyFSDirs;
folderBrowser.Title = "Suchen Sie den Ordner:";
if (folderBrowser.ShowDialog())
    Console.WriteLine(folderBrowser.FolderPath);
else
    Console.WriteLine("Abbruch");
```

Listing 365: Ordnerdialog über die Klasse *ShellFolderBrowser*

Das einzig Komplexere an der Klasse sind die Flags, die Sie in die Eigenschaft `BrowseFlags` schreiben. Hier können Sie mit den Werten der Aufzählung `BrowseFlags` u.a. festlegen, dass der Dialog nur alle Drucker anzeigen soll (`BrowseForPrinter`), nur alle Computer anzeigen soll (`BrowseForComputer`), ein Eingabefeld zusätzlich anzeigt (`EditBox`) und/oder nur Dateisystemordner zurückgeben soll (`ReturnOnlyFSDirs`, bei allen anderen Ordnern ist dann der OK-Schalter deaktiviert). Schauen Sie sich das mit dem Quellcode von The Code Project mitgelieferte Beispiel an, das alle Varianten des Aufrufs implementiert.

Für unseren Zweck reicht der Aufruf in Listing 365 normalerweise aus, wie Abbildung 140 zeigt.



Abbildung 140: Ein Ordnerdialog über die Klasse `ShellFolderBrowser`

196 Hooking: Eine `MessageBox` mit definierten Schalterbeschriftungen versehen

Die in .NET über die Klasse `MessageBox` implementierte Standard-Windows-`MessageBox` besitzt leider nur vordefinierte Schalterbeschriftungen. In vielen Fällen passen diese aber nicht zu der jeweiligen Anwendung. Wenn Sie den Anwender beispielsweise fragen wollen, ob er beim Kopieren von Dateien die aktuelle Datei oder alle Dateien überschreiben will, können Sie mit den Default-Schaltern (JA – NEIN, JA – NEIN – ABBRECHEN, ABBRECHEN – IGNORIEREN – WIEDERHOLEN etc.) nicht viel anfangen. Ideal wäre eine `MessageBox` wie in Abbildung 141.

Um dieses Problem zu lösen können Sie ein Windows-Formular erzeugen, das den Text, die Schalter und das Icon je nach Aufruf einer `Show`-Methode erzeugt. Das macht aber sehr viel Arbeit, wenn Sie das Formular möglichst flexibel einsetzbar machen wollen.

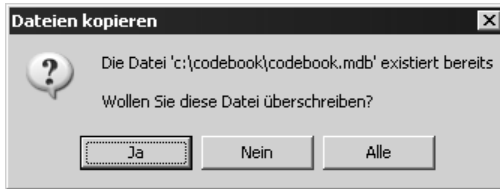


Abbildung 141: MessageBox mit speziellen Schalterbeschriftungen

Einfacher ist es, die originale Windows-MessageBox zu verändern. Über die API-Funktion `SetDlgItemText` können Sie die Beschriftungen einzelner Schalter im Fenster eines Dialoges (also auch einer MessageBox) ändern. Dazu übergeben Sie den Windows-Handle des Dialogs, eine Id für den Schalter und den neuen Text. Die Ids der Schalter einer MessageBox sind in Windows festgelegt. Der Wert 1 steht z.B. für einen OK-Schalter, 2 steht für einen Abbrechen-Schalter.

Das Problem ist nur, dass die MessageBox modal ist und Sie folglich nach dem Öffnen nicht an das Fenster herankommen. Ein weiteres Problem wäre auch im anderen Fall, dass Sie den Windows-Handle des MessageBox-Fensters herausfinden müssen.

Die Lösung dieser Probleme ist das so genannte *Hooking*. Hooking bedeutet, dass Sie eine spezielle (Hook-)Funktion so installieren, dass diese bestimmte Windows-Nachrichten abfängt. Wie Sie ja wahrscheinlich bereits wissen, basiert die Interaktion zwischen dem Anwender und den Programm-Fenstern in Windows auf Nachrichten, die Windows bei allen möglichen Ereignissen an die Fenster sendet. Fenster verarbeiten diese Nachrichten und reagieren entsprechend darauf. In modernen Programmiersystemen werden Windows-Nachrichten in Ereignisse umgewandelt. Wenn Sie z.B. für das Ereignis `MouseMove` eines Formulars eine Behandlungsmethode schreiben, reagieren Sie in Wirklichkeit auf die vom Fenster empfangene Windows-Nachricht `WM_MOUSEMOVE`.

Über einen Hook können Sie auf Windows-Nachrichten auch dann reagieren, wenn Sie ein Fenster nicht selbst programmiert haben. Sie installieren dazu über die API-Funktion `SetWindowsHookEx` eine spezielle Hook-Funktion. Dabei definieren Sie die Art des Hooks über festgelegte API-Konstanten. Ein Hook des Typs `WH_KEYBOARD` reagiert z.B. auf alle Tastatur-Nachrichten (womit Sie einen globalen Tastatur-Handler entwickeln könnten). Die verfügbaren Hook-Typkonstanten sind in der API-Dokumentation zu `SetWindowsHookEx` beschrieben.

Für unseren Zweck eignet sich ein *CBT-Hook*. Dieser primär für »Computer Based Training«-Programme entwickelte Hook wird bei verschiedenen Nachrichten wie dem Erstellen, Aktivieren oder Zerstören eines Fensters aufgerufen. Da das Aktivieren eines Fensters genau das ist, was wir benötigen, verwende ich einen solchen Hook. Innerhalb der Hook-Funktion erhalten Sie über das Argument `code` eine Information über die Unterart des Hook. Die Konstante `HCBT_ACTIVATE` steht für Nachrichten, die beim Aktivieren eines Fensters aufgerufen werden. Aus dem Argument `wParam` können Sie dann den Handle des gerade aktivierten Fensters auslesen um damit Windows-API-Funktionen wie `SetDlgItemText` aufzurufen.

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Hooks können Sie global (für alle Windows-Nachrichten) oder nur für einen Thread (den Haupt-Thread einer Anwendung) installieren. Globale Hooks können das System überwachen, sind aber recht gefährlich, wenn innerhalb der Hook-Funktion ein modaler Dialog aufgerufen wird (der das System dann lahmlegt) oder ein Fehler auftritt (der das System u. U. abschießt). Für unseren Zweck reicht aber ein weniger gefährlicher Thread-lokaler Hook. Um diesen zu erzeugen übergeben Sie `SetWindowsHookEx` im letzten Argument die Id des aktuellen Threads (die Sie über `System.AppDomain.GetCurrentThreadId` auslesen).

In der Hook-Funktion sollten Sie dafür Sorge tragen, dass Windows nach Ihrem Hook einen eventuell vorhandenen nächsten abarbeiten kann, was Sie durch den Aufruf der Funktion `CallNextHookEx` erreichen.

Ein einfaches Beispiel

Als erstes einfaches Beispiel erzeugt die folgende Konsolenanwendung eine `MessageBox` mit drei benutzerdefinierten Schaltern. Zunächst benötigen Sie eine Variable, die den Handle des Hooks verwaltet:

```
class Hooking
{
    private static IntPtr hHook;
```

Eine Aufzählung verwaltet die Konstanten der verschiedenen Hook-Typen (obwohl hier nur ein CBT-Hook verwendet wird):

```
public enum HookType: int
{
    WH_JOURNALRECORD = 0,
    WH_JOURNALPLAYBACK = 1,
    WH_KEYBOARD = 2,
    WH_GETMESSAGE = 3,
    WH_CALLWNDPROC = 4,
    WH_CBT = 5,
    WH_SYSMSGFILTER = 6,
    WH_MOUSE = 7,
    WH_HARDWARE = 8,
    WH_DEBUG = 9,
    WH_SHELL = 10,
    WH_FOREGROUNDIDLE = 11,
    WH_CALLWNDPROCRET = 12,
    WH_KEYBOARD_LL = 13,
    WH_MOUSE_LL = 14
}
```

Für CBT-Hooks benötigen Sie eine weitere Aufzählung, die die Nachrichten spezifiziert, bei denen ein solcher Hook aufgerufen werden kann:

```
private enum CBTHookAction: int
{
    HCBT_MOVESIZE = 0,
    HCBT_MINMAX = 1,
    HCBT_QS = 2,
    HCBT_CREATEWND = 3,
    HCBT_DESTROYWND = 4,
    HCBT_ACTIVATE = 5,
    HCBT_CLICKSKIPPED = 6,
    HCBT_KEYSKIPPED = 7,
    HCBT_SYSCOMMAND = 8,
    HCBT_SETFOCUS = 9
}
```

Zur Veränderung der Beschriftungen der MessageBox-Schalter über `SetDlgItemText` werden die einzelnen Schaltertypen über festgelegte Konstanten definiert, die Sie ebenfalls in einer Aufzählung deklarieren sollten:

```
private enum ButtonType
{
    IDOK = 1,
    IDCANCEL = 2,
    IDABORT = 3,
    IDRETRY = 4,
    IDIGNORE = 5,
    IDYES = 6,
    IDNO = 7
}
```

Da Sie in echten .NET-Programmen keine Funktionsadressen direkt übergeben können, benötigen Sie einen Delegate für die Hook-Funktion. Die Signatur ist dabei festgelegt, wobei Sie das eigentlich als `int` deklarierte erste Argument zur Vereinfachung auch mit dem kompatiblen Aufzählungstyp deklarieren können:

```
private delegate int HookProcDelegate(CBTHookAction nCode,
    IntPtr wParam, IntPtr lParam);
```

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Nun folgt die Deklaration der benötigten API-Funktionen:

```
[DllImport("user32.dll", CharSet=CharSet.Auto)]
private static extern IntPtr SetWindowsHookEx(HookType idHook,
    HookProcDelegate lpfn, IntPtr hMod, int dwThreadId);

[DllImport("user32.dll")]
private static extern int CallNextHookEx(IntPtr hhook,
    CBTHookAction code, IntPtr wParam, IntPtr lParam);

[DllImport("user32.dll")]
private static extern int UnhookWindowsHookEx(IntPtr hhook);

[DllImport("User32")]
static extern private bool SetDlgItemText(IntPtr hWnd,
    int nIDDlgItem, string lpString);
```

Die Hook-Funktion, die die eigentliche Arbeit übernimmt, muss dann der im Delegate festgelegten Signatur entsprechen:

```
private static int HookProc(CBTHookAction code, IntPtr wParam,
    IntPtr lParam)
{
    if (code == CBTHookAction.HCBT_ACTIVATE)
    {
        // Wenn das Fenster gerade aktiviert wird:
        // Schalter neu beschriften
        SetDlgItemText(wParam, (int)ButtonType.IDYES, "Jau");
        SetDlgItemText(wParam, (int)ButtonType.IDNO, "Nein");
        SetDlgItemText(wParam, (int)ButtonType.IDCANCEL,
            "Vielleicht");
    }

    // Zum nächsten Hook in der Hook-Kette wechseln
    return CallNextHookEx(hHook, code, wParam, lParam);
}
```

Die Hook-Funktion vergleicht zunächst, ob der CBT-Hook für das Aktivieren eines Fensters aufgerufen wurde. Da der Hook weiter unten nur für den aktuellen Thread installiert wird, kann die Funktion sicher sein, dass es sich dabei um das Fenster der MessageBox handelt. Über die Funktion `SetDlgItemText` wird dann für jeden der Schalter ein neuer Text definiert. Der Handle des MessageBox-Fensters wird dabei aus dem Argument `wParam` ausgelesen.

Wie bereits gesagt ist der Aufruf von `CallNextHookEx` zur Verzweigung in den nächsten Hook sehr wichtig. Dabei müssen Sie auch den Rückgabewert dieser Funktion, der vom verwendeten Hook abhängt, in der Hook-Funktion selbst zurückgeben. Nur so sichern Sie ab, dass ein Hook ohne die Gefahr eines Absturzes funktioniert.

In der `Main`-Funktion der Anwendung installieren Sie nun den Hook:

```
[STAThread]
static void Main(string[] args)
{
    // Hook installieren
    hHook = SetWindowsHookEx(HookType.WH_CBT,
        new HookProcDelegate(HookProc), IntPtr.Zero,
        (int)AppDomain.GetCurrentThreadId());
}
```

Die Installation erfolgt so, dass ein CBT-Hook erzeugt wird. Für die Hook-Funktion wird ein neuer Delegate erzeugt, der am zweiten Argument übergeben wird. Das dritte Argument wird in API-Programmen verwendet, wenn die Hook-Funktion innerhalb einer DLL gespeichert ist, und muss in unserem Fall auf `IntPtr.Zero` gesetzt werden. Am vierten Argument übergebe ich die Id des aktuellen Thread, damit der Hook nur für diesen installiert wird.

Dann kann die Standard-MessageBox so aufgerufen werden, dass diese auch drei Schalter besitzt:

```
MessageBox.Show("C# ist doch cool, oder?",
    Application.ProductName, MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question);
```

Schließlich müssen Sie den Hook auf jeden Fall wieder deinstallieren:

```
UnhookWindowsHookEx(hHook);
}
}
```

Und das Ergebnis ist überzeugend (Abbildung 142).

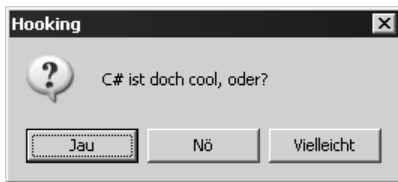


Abbildung 142: Das Ergebnis des Hookings der MessageBox

Eine Klasse für eine MessageBox mit benutzerdefinierten Schalterbeschriftungen

Um die Hooking-Komplexität zu verstecken und die Erzeugung einer MessageBox mit eigenen Schalterbeschriftungen zu erleichtern, habe ich eine Klasse `ExtMessageBox` entwickelt. Die `Show`-Methode dieser Klasse arbeitet ähnlich der gleichnamigen Methode der `MessageBox`-Klasse, nur dass diese Methode (aus Vereinfachungsgründen) nicht mehrfach überladen ist

und an Stelle des `buttons`-Arguments drei Strings für die einzelnen Schalterbeschriftungen erwartet. Übergeben Sie an Stelle einer Beschriftung `null`, wird der entsprechende Schalter nicht dargestellt. Die Rückgabe der Methode ist ein Wert vom neuen Typ `ExtDialogResult`, der mit seinen Konstanten `FirstButton`, `SecondButton` und `ThirdButton` den Schalter definiert, der betätigt wurde.

`Show` arbeitet auf der Basis des vorhergehenden Beispiels. Abhängig von den übergebenen Schalterbeschriftungen erzeugt diese Methode eine `MessageBox` mit einem Ok-, einem Ja- und einem Nein- oder einem Ja-, einem Nein- und einem Abbrechen-Schalter. Da die übergebenen Beschriftungen in privaten Eigenschaften gespeichert werden, kann die Hook-Funktion ermitteln, welche Schalter die `MessageBox` besitzt, um diese gezielt zu verändern. Nach dem Aufruf der `MessageBox` vergleicht `Show` die Rückgabe auch wieder abhängig von den definierten Beschriftungen mit den `DialogResult`-Konstanten und gibt einen entsprechenden `ExtDialogResult`-Wert zurück.

`Show` erfordert den Import der Namensräume `System`, `System.Windows.Forms` und `System.Runtime.InteropServices`.

```
public class ExtMessageBox
{
    /* Aufzählung für die Rückgabe der Show-Methode */
    public enum ExtDialogResult
    {
        FirstButton,
        SecondButton,
        ThirdButton
    }

    /* Eigenschaft für den Hook-Handle */
    private static IntPtr hHook;

    /* Eigenschaften, die die Beschriftungen der Schalter speichern und
     * in der Hook-Funktion abgefragt werden */
    private static string button1Caption;
    private static string button2Caption;
    private static string button3Caption;

    /* Aufzählung für die verschiedenen Hook-Typen */
    public enum HookType: int
    {
        WH_JOURNALRECORD = 0,
        WH_JOURNALPLAYBACK = 1,
        WH_KEYBOARD = 2,
        WH_GETMESSAGE = 3,
        WH_CALLWNDPROC = 4,
        WH_CBT = 5,
        WH_SYSMSGFILTER = 6,
        WH_MOUSE = 7,
```

Listing 366: Klasse für eine erweiterte `MessageBox` mit der Möglichkeit, die Beschriftungen der Schalter zu definieren

```

        WH_HARDWARE = 8,
        WH_DEBUG = 9,
        WH_SHELL = 10,
        WH_FOREGROUNDIDLE = 11,
        WH_CALLWNDPROCRET = 12,
        WH_KEYBOARD_LL = 13,
        WH_MOUSE_LL = 14
    }

    /* Aufzählung für die Aktionen, bei denen ein CBT-Hook aufgerufen
    * werden kann */
    private enum CBTHookAction : int
    {
        HCBT_MOVESIZE = 0,
        HCBT_MINMAX = 1,
        HCBT_QS = 2,
        HCBT_CREATEWND = 3,
        HCBT_DESTROYWND = 4,
        HCBT_ACTIVATE = 5,
        HCBT_CLICKSKIPPED = 6,
        HCBT_KEYSKIPPED = 7,
        HCBT_SYSCOMMAND = 8,
        HCBT_SETFOCUS = 9
    }

    /* Aufzählung für die MessageBox-Schalterttypen */
    private enum ButtonType
    {
        IDOK = 1,
        IDCANCEL = 2,
        IDABORT = 3,
        IDRETRY = 4,
        IDIGNORE = 5,
        IDYES = 6,
        IDNO = 7
    }

    /* Delegate für die Hook-Funktion */
    private delegate int HookProcDelegate(CBTHookAction nCode,
        IntPtr wParam, IntPtr lParam);

    /* Deklaration der benötigten API-Funktionen */
    [DllImport("user32.dll", CharSet=CharSet.Auto)]
    private static extern IntPtr SetWindowsHookEx(HookType idHook,
        HookProcDelegate lpfn, IntPtr hMod, int dwThreadId);

    [DllImport("user32.dll")]
    private static extern int CallNextHookEx(IntPtr hhook,
        CBTHookAction code, IntPtr wParam, IntPtr lParam);

```

Listing 366: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

[DllImport("user32.dll")]
private static extern int UnhookWindowsHookEx(IntPtr hhook);

[DllImport("User32")]
static extern private bool SetDlgItemText(IntPtr hWnd,
    int nIDDlgItem, string lpString);

/* Die Hook-Funktion */
private static int HookProc(CBTHookAction code, IntPtr wParam,
    IntPtr lParam)
{
    if (code < 0)
        // Wenn keine Hook-Aktion übergeben wurde, Windows
        // direkt zum nächsten Hook verzweigen lassen
        return CallNextHookEx(hHook, code, wParam, lParam);

    if (code == CBTHookAction.HCBT_ACTIVATE)
    {
        // Wenn das Fenster gerade aktiviert wird:
        // Schalter neu beschriften
        if (button3Caption != null)
        {
            SetDlgItemText(wParam, (int)ButtonType.IDYES,
                button1Caption);
            SetDlgItemText(wParam, (int)ButtonType.IDNO,
                button2Caption);
            SetDlgItemText(wParam, (int)ButtonType.IDCANCEL,
                button3Caption);
        }
        else if (button2Caption != null)
        {
            SetDlgItemText(wParam, (int)ButtonType.IDYES,
                button1Caption);
            SetDlgItemText(wParam, (int)ButtonType.IDNO,
                button2Caption);
        }
        else
        {
            SetDlgItemText(wParam, (int)ButtonType.IDOK,
                button1Caption);
        }
    }

    // Zum nächsten Hook in der Hook-Kette wechseln
    return CallNextHookEx(hHook, code, wParam, lParam);
}

/* Methode zum Aufruf der MessageBox */
public static ExtDialogResult Show(string text, string
    caption, string button1Caption, string button2Caption,

```

Listing 366: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

```

string button3Caption, MessageBoxIcon icon,
MessageBoxDefaultButton defaultButton)
{
    // Festlegen der als Basis verwendeten Schalter
    MessageBoxButtons buttons;
    if (button3Caption != null)
        buttons = MessageBoxButtons.YesNoCancel;
    else if (button2Caption != null)
        buttons = MessageBoxButtons.YesNo;
    else
        buttons = MessageBoxButtons.OK;

    // Schalterbeschriftungen an die privaten Eigenschaften
    // übergeben, damit diese in der Hook-Funktion zur
    // Verfügung stehen
    ExtMessageBox.button1Caption = button1Caption;
    ExtMessageBox.button2Caption = button2Caption;
    ExtMessageBox.button3Caption = button3Caption;

    // CBT-Hook nur für den aktuellen Thread installieren
    hHook = SetWindowsHookEx(HookType.WH_CBT,
        new HookProcDelegate(HookProc), IntPtr.Zero,
        (int)AppDomain.GetCurrentThreadId());

    // Standard-MessageBox aufrufen
    ExtDialogResult returnValue;
    switch(MessageBox.Show(text, caption, buttons, icon,
        defaultButton))
    {
        case DialogResult.OK:
            returnValue = ExtDialogResult.FirstButton;
            break;
        case DialogResult.Yes:
            returnValue = ExtDialogResult.FirstButton;
            break;
        case DialogResult.No:
            returnValue = ExtDialogResult.SecondButton;
            break;
        case DialogResult.Cancel:
            returnValue = ExtDialogResult.ThirdButton;
            break;
        default:
            returnValue = ExtDialogResult.FirstButton;
            break;
    }

    // Hook wieder deinstallieren
    UnhookWindowsHookEx(hHook);
}

```

Listing 366: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
        // Ergebnis zurückgeben
        return returnValue;
    }
}
```

Listing 366: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

Die Verwendung dieser Klasse ist dann sehr einfach:

```
switch(ExtMessageBox.Show(@"Die Datei 'c:\codebook\codebook.mdb' " +
    "existiert bereits\r\n\r\nWollen Sie diese Datei überschreiben?",
    "Dateien kopieren", "Ja", "Nein", "Alle",
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button1))
{
    case ExtDialogResult.FirstButton:
        Console.WriteLine("Datei überschreiben ...");
        break;
    case ExtDialogResult.SecondButton:
        Console.WriteLine("Datei nicht überschreiben ...");
        break;
    case ExtDialogResult.ThirdButton:
        Console.WriteLine("ALle Dateien überschreiben ...");
        break;
}
```

Listing 367: Anwendung der Klasse für eine erweiterte MessageBox

Benutzer, Gruppen und Sicherheit

197 Informationen zu den Benutzern eines Computers oder einer Domäne auflisten

Die auf einem Computer oder in einer Domäne registrierten Benutzer können Sie über WMI oder ADSI auslesen. Ich verwende in diesem und den folgenden Rezepten ADSI, da diese Technologie für die Arbeit mit Benutzer- und Gruppenkonten mehr Möglichkeiten bietet als WMI. So ist es z.B. über WMI (nach meinen Erfahrungen und Recherchen in Newsgroups) nicht möglich, Benutzerkonten zu erzeugen oder ein Benutzerkonto einer Gruppe hinzuzufügen, was mit ADSI kein Problem darstellt.

Um alle Benutzer eines Computers oder einer Domäne aufzulisten, können Sie den ADSI-Provider *WinNT* verwenden (der unabhängig vom Namen auch unter anderen Windows-Versionen zur Verfügung steht). Dieser Provider ermöglicht (im Gegensatz zum LDAP-Provider) den direkten Zugriff auf Computer im Netzwerk und bietet daneben (wie der LDAP-Provider) auch den Zugriff auf eine Domäne. Informationen zum WinNT-Provider finden Sie an der Adresse msdn.microsoft.com/library/en-us/netdir/adsi/adsi_winnt_provider.asp.

Die in Listing 370 beschriebene Methode `EnumUsers` setzt diesen Provider ein, um alle Benutzer eines Computers bzw. einer Domäne aufzulisten.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System`, `System.Collections` und `System.DirectoryServices` importieren. Zur Ausführung ist die Installation von ADSI auf dem Zielsystem erforderlich.

Da die wichtigsten Eigenschaften eines Benutzers ausgelesen werden sollen, habe ich zunächst eine Klasse `User` deklariert, deren Instanzen einen Benutzer repräsentieren sollen:

```
public class User
{
    public string Name;           // Benutzername
    public string FullName;       // der volle Name
    public string Description;     // eine Beschreibung
    public int MaxStorage;        // Maximaler Speicherplatz
    public string HomeDirectory;  // Benutzer-Home-Verzeichnis
    public DateTime LastLogin;    // letzter Login
}
```

Listing 368: Klasse für einen Benutzer

Hinweis

Um das Rezept nicht zu komplex zu gestalten habe ich nur die in meinen Augen wichtigsten Eigenschaften eines Benutzers in die Klasse aufgenommen. Im Rezept 202 finden Sie eine Lösung zum Auslesen aller Eigenschaften eines Benutzers.

Basics

Datum
Zeit

Anwen-
dungen

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

Um die Benutzer typsicher auflisten zu können, implementiert die Klasse `Users` eine Auflistung, die von der Basisklasse `CollectionBase` abgeleitet ist. `Users` implementiert lediglich den Indizierer und die `Add`-Methode. `Add` ist `internal` deklariert, um zu verhindern, dass von außen Benutzer hinzugefügt werden können (was allerdings voraussetzt, dass die hier beschriebenen Klassen in eine separate Assembly kompiliert werden):

```
public class Users: CollectionBase
{
    /* Indizierer */
    public User this[int index]
    {
        set {base.InnerList[index] = value;}
        get {return (User)base.InnerList[index];}
    }

    /* Add-Methode */
    internal void Add(User user)
    {
        base.InnerList.Add(user);
    }
}
```

Listing 369: Klasse zur Auflistung von User-Objekten

Die Methode `EnumUsers` fragt alle Benutzer eines Computers oder einer Domäne ab, deren Namen übergeben werden. Diese Methode erzeugt für jeden Benutzer ein `User`-Objekt, fügt dieses an eine Auflistung vom Typ `Users` an und gibt diese zurück.

`EnumUsers` erzeugt zum Auflisten der Benutzer ein `DirectoryEntry`-Objekt (aus dem Namensraum `System.DirectoryServices`), das als ADSI-Objekt den Computer bzw. die Domäne repräsentiert. Im Konstruktor wird der ADSI-Pfad übergeben, der zuvor je nach dem übergebenen Domänen- und Computernamen für den WinNT-Provider zusammengesetzt wurde. Wird ein Computernamen und/oder der Name einer Domäne übergeben, repräsentiert der ADSI-Pfad den angegebenen Computer. Wird nur ein Domänenname übergeben, steht der Pfad für die Domäne.

Der Pfad zur Domäne *Galaxy* sieht beispielsweise so aus:

```
WinNT://Galaxy,domain
```

Der Pfad zum Computer *Zaphod* sieht folgendermaßen aus:

```
WinNT://Zaphod,computer
```

Sind Computer und Domäne angegeben, enthält der Pfad beide Angaben:

```
WinNT://Galaxy/Zaphod,computer
```

Auf diese Weise können Sie die lokalen Benutzer eines Computers oder die globalen Benutzer einer Domäne auflisten. Für den Fall, dass an beiden Argumenten `null` übergeben wird, setzt `EnumUsers` den Namen des lokalen Computers in das Argument `machineName` ein.

Da der Zugriff auf Verzeichnisdienste normalerweise geschützt ist, übergibt `EnumUsers` am zweiten und dritten Argument des `DirectoryEntry`-Konstruktors einen Benutzernamen und dessen Passwort. Diesen Benutzer verwendet das `DirectoryEntry`-Objekt zur Authentifizierung gegenüber dem Verzeichnisdienst. Würden Sie diese beiden Argumente weglassen, würde das Objekt das Konto des aktuell in Windows eingeloggten Benutzers verwenden. Da dieses in der Praxis häufig nicht die erforderlichen Rechte besitzt, habe ich `EnumUsers` so entwickelt, dass der für ADSI zu verwendende Benutzer und dessen Passwort übergeben werden können. Sie können aber auch einfach `null` übergeben, wenn das Konto des aktuell eingeloggten Benutzers für die Authentifizierung eingesetzt werden soll.

Danach geht `EnumUsers` über die `Children`-Eigenschaft alle dem ermittelten `DirectoryEntry`-Objekt untergeordneten Objekte durch (die wieder über `DirectoryEntry`-Instanzen repräsentiert werden). Für jedes so ermittelte Objekt wird über dessen Eigenschaft `SchemaClassName` abgefragt, ob es sich um ein Objekt der Schema-Klasse `User` handelt (die ein Bestandteil des WinNT-Providers ist). Ist dies der Fall, erzeugt `EnumUsers` ein neues `User`-Objekt und initialisiert dies zunächst mit dem Namen des Benutzers (der aus der Eigenschaft `Name` des `DirectoryEntry`-Objekts ausgelesen wird). Die anderen Eigenschaften eines Benutzers sind in der generischen `DirectoryEntry`-Klasse nicht direkt bekannt und stehen über die `Properties`-Auflistung zur Verfügung, der Sie den Namen der Eigenschaft übergeben. Da diese Auflistung `object`-Instanzen zurückgibt, müssen diese noch konvertiert werden.

ADSI-Objekte weisen eine Besonderheit im Vergleich zu »normalen« Objekten auf: Ein ADSI-Objekt basiert nicht auf einer Klasse, sondern auf einem so genannten *Schema*. Dieses Schema legt fest, welche Eigenschaften das Objekt besitzen *muss* und welche es besitzen *kann*. Bei der `User`-Klasse sind alle Eigenschaften außer der immer vorhandenen Eigenschaft `Name` optional. Die Abfrage über die `Properties`-Auflistung führt im Falle einer nicht vorhandenen Eigenschaft zur Rückgabe von `null`. `EnumUsers` beachtet dies, indem die Abfrage der Eigenschaften in einer Ausnahmebehandlung erfolgt (die allerdings die Ausnahme verwirft).

```
public static Users EnumUsers(string domainName, string machineName,
    string bindUser, string bindPassword)
{
    // Users-Auflistung für das Ergebnis erzeugen
    Users users = new Users();

    // Gültigen Rechnernamen für den lokalen Computer erzeugen, falls
    // weder die Domäne noch der Rechnername übergeben wurden
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;

    // ADSI-Pfad für den WinNT-Provider zum Auslesen des Computers bzw.
    // der Domäne zusammenstellen
```

Listing 370: Methode zur Auflistung aller Benutzer eines Computers

```

string adsiPath = "WinNT://";
if (domainName != null && machineName != null)
    adsiPath += domainName + "/" + machineName + ",computer";
else if (machineName != null)
    adsiPath += machineName + ",computer";
else if (domainName != null)
    adsiPath += domainName + ",domain";

// DirectoryEntry-Objekt erzeugen
DirectoryEntry computerEntry = new DirectoryEntry(adsiPath, bindUser,
    bindPassword);

try
{
    // Alle dem Computer untergeordneten Objekte durchgehen
    foreach(DirectoryEntry de in computerEntry.Children)
    {
        // Überprüfen, ob es sich um ein User-Objekt handelt
        if (de.SchemaClassName.ToLower() == "user")
        {
            // Neues Benutzer-Objekt erzeugen und initialisieren
            User user = new User();
            user.Name = de.Name;
            try {user.FullName = (string)de.Properties[
                "FullName"].Value;} catch {}
            try {user.Description = (string)de.Properties[
                "Description"].Value;} catch {}
            try {user.HomeDirectory = (string)de.Properties[
                "HomeDirectory"].Value;} catch {}
            try {user.MaxStorage = (int)de.Properties[
                "MaxStorage"].Value;} catch {}
            try {user.LastLogin = (DateTime)de.Properties[
                "LastLogin"].Value;} catch {}

            // User-Objekt an die Auflistung anfügen
            users.Add(user);
        }
    }
}
finally
{
    // ADSI-Objekt freigeben
    computerEntry.Dispose();
}

// Benutzer-Auflistung zurückgeben
return users;
}

```

Listing 370: Methode zur Auflistung aller Benutzer eines Computers (Forts.)

```

C:\Codebook\Informationen zu den Benutzern eines Computers oder eine...
Name: ACTUser
Voller Name: Application Center TestKonto
Beschreibung: Konto, das zum Starten der Dienste Application
Center Test Broker und Application Center Test Controller v
erwendet wird
Letzter Login: 20.01.2003 01:36:59
Home-Verzeichnis:
Maximaler Speicherplatz: -1

Name: Administrator
Voller Name: Jürgen Bayer
Beschreibung: Vordefiniertes Konto für die Verwaltung des Co
mputers bzw. der Domäne
Letzter Login: 26.04.2003 18:50:19
Home-Verzeichnis:
Maximaler Speicherplatz: -1

Name: ASPNET
Voller Name: aspnet_wp account
Beschreibung: Account for running ASP.NET Worker process
Letzter Login: 01.01.0001 00:00:00
Home-Verzeichnis:
Maximaler Speicherplatz: -1

```

Abbildung 143: Das Beispielprogramm zur Auflistung aller Benutzer eines Computers oder einer Domäne in Aktion

Die Anwendung der `EnumUsers`-Methode zeigt das folgende Listing, das alle Benutzer des Computers *Zaphod* ermittelt und deren Daten an der Konsole ausgibt:

```

string domainName = null;
string machineName = "Zaphod";
string bindUser = null;
string bindPassword = null;

// Benutzer abfragen
try
{
    Users users = EnumUsers(domainName, machineName, bindUser, bindPassword);

    // Benutzer durchgehen und deren Daten ausgeben
    for (int i = 0; i < users.Count; i++)
    {
        Console.WriteLine();
        Console.WriteLine("Name: {0}", users[i].Name);
        Console.WriteLine("Voller Name: {0}", users[i].FullName);
        Console.WriteLine("Beschreibung: {0}", users[i].Description);
        Console.WriteLine("Letzter Login: {0}", users[i].LastLogin);
        Console.WriteLine("Home-Verzeichnis: {0}", users[i].HomeDirectory);
        Console.WriteLine("Maximaler Speicherplatz: {0}", users[i].MaxStorage);
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 371: Anwendung der `EnumUsers`-Methode

198 Überprüfen, ob ein Benutzerkonto existiert

Die Existenz eines Benutzers können Sie relativ einfach über ADSI überprüfen. Die Methode `UserExists` in Listing 372 überprüft, ob ein Benutzer auf einem Computer oder in einer Domäne existiert. Neben dem Benutzernamen erwartet diese Methode den Namen der Domäne und den des Computers. Um die Existenz eines Benutzers in einer Domäne zu überprüfen, übergeben Sie nur den Domänennamen und am Argument `machineName` `null`. Wenn Sie den Rechner- und/oder Domänennamen angeben, überprüft `UserExists` die Existenz auf einem bestimmten Computer. Für den Fall dass an den Argumenten `domainName` und `machineName` `null` übergeben wird, schreibt `UserExists` den Namen des lokalen Computers in das Argument `machineName`.

Die Argumente `bindUser` und `bindPassword` werden wie im Rezept 197 für die Authentifizierung gegenüber ADSI verwendet.

`UserExists` erzeugt zunächst ein `DirectoryEntry`-Objekt, das je nach den übergebenen Domänen- und Rechnernamen entweder den Computer oder die Domäne repräsentiert. Die Existenz des angegebenen Benutzers wird dann über den Aufruf der `Find`-Methode dieses Objekts überprüft. Dieser Methode wird der Name des gesuchten Objekts und dessen ADSI-Klasse übergeben. `Find` sucht im Container, von dem aus diese Methode aufgerufen wurde, und erzeugt eine Ausnahme vom Typ `COMException`, wenn kein passendes Objekt gefunden wurde. Diese Ausnahme nutzt `UserExists` um zu entscheiden, ob der Benutzer existiert.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.DirectoryServices` importieren.

```
public static bool UserExists(string userName, string domainName,
    string machineName, string bindUser, string bindPassword)
{
    // Gültigen Rechnernamen für den lokalen Computer erzeugen, falls
    // weder die Domäne noch der Rechnername übergeben wurden
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;

    // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
    string adsiPath = "WinNT://";
    if (domainName != null && machineName != null)
        adsiPath += domainName + "/" + machineName + ",computer";
    else if (machineName != null)
        adsiPath += machineName + ",computer";
    else if (domainName != null)
        adsiPath += domainName + ",domain";
    DirectoryEntry computerEntry = new DirectoryEntry(adsiPath, bindUser,
        bindPassword);

    try
    {
        DirectoryEntry userEntry = computerEntry.Children.Find(
```

Listing 372: Methode zur Überprüfung der Existenz eines Benutzerkontos

```

        userName, "user");
    return true;
}
catch
{
    return false;
}
}

```

Listing 372: Methode zur Überprüfung der Existenz eines Benutzerkontos (Forts.)

Listing 373 zeigt eine beispielhafte Anwendung der `UserExists`-Methode. Überprüft wird, ob der Benutzer *Administrator* auf dem lokalen Rechner existiert.

```

string domainName = null;
string machineName = null;
string bindUser = null;
string bindPassword = null;

if (UserExists("Administrator", domainName, machineName, bindUser,
    bindPassword))
    Console.WriteLine("Administrator existiert");
else
    Console.WriteLine("Administrator existiert nicht");

```

Listing 373: Beispielhafte Anwendung der UserExists-Methode

199 Benutzergruppen eines Computers oder einer Domäne auflisten

Um alle auf einem Computer oder in einer Domäne verwalteten Benutzergruppen auszulesen können Sie wie im Rezept 197 ADSI verwenden. Die im Folgenden beschriebene Methode `EnumGroups` basiert auf diesem Rezept. Ich habe den Quellcode lediglich ein wenig verändert. Die Klasse `Group` verwaltet nun die (wenigen) Eigenschaften einer Gruppe, die Klasse `Groups` ist eine Auflistung von `Group`-Objekten. Die Methode `EnumGroups` liest ähnlich der Methode `EnumUsers` nun alle Benutzergruppen ein, speichert deren Eigenschaften in einer `Group`-Instanz, fügt diese der `Groups`-Auflistung hinzu und gibt diese Auflistung zurück.

`EnumGroups` wird der Name der Domäne und/oder der Name des Computers übergeben. Wollen Sie die Gruppen einer Domäne abfragen, übergeben Sie lediglich den Domänennamen und tragen am Argument `machineName` `null` ein. Zur Abfrage der lokalen Gruppen eines Computers geben Sie den Computernamen und optional den Namen der Domäne an. Für die Domäne können Sie dann auch `null` übergeben, falls der abgefragte Computer in derselben Domäne verwaltet wird wie der anfragende. Über das Argument `bindUser` können Sie einen Benutzer angeben, dessen Konto für die ADSI-Authentifizierung verwendet werden soll. Dessen Passwort geben Sie dann am Argument `bindPassword` an. Alternativ können Sie am Argument `bindUser` `null` angeben, wenn das Konto des aktuell in Windows eingeloggten Benutzers für die Authentifizierung verwendet werden soll.

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System`, `System.Collections` und `System.DirectoryServices` importieren.

```

/* Klasse für eine Benutzer-Gruppe */
public class Group
{
    public string Name;           // Benutzername
    public string Description;    // eine Beschreibung
}

/* Klasse zur Auflistung von Group-Instanzen */
public class Groups: CollectionBase
{
    /* Indizierer */
    public Group this[int index]
    {
        set {base.InnerList[index] = value;}
        get {return (Group)base.InnerList[index];}
    }

    /* Add-Methode */
    internal void Add(Group group)
    {
        base.InnerList.Add(group);
    }
}

/* Methode zum Auflisten aller Gruppen eines Computers */
public static Groups EnumGroups(string domainName,
    string machineName, string bindGroup, string bindPassword)
{
    // Groups-Auflistung für das Ergebnis erzeugen
    Groups groups = new Groups();

    // Gültigen Rechnernamen für den lokalen Computer erzeugen, falls
    // weder die Domäne noch der Rechnername übergeben wurden
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;

    // DirectoryEntry-Objekt für den Computer oder die Domäne erzeugen
    string adsiPath = "WinNT://";
    if (domainName != null && machineName != null)
        adsiPath += domainName + "/" + machineName + ",computer";
    else if (machineName != null)
        adsiPath += machineName + ",computer";
    else if (domainName != null)
        adsiPath += domainName + ",domain";
    DirectoryEntry computerEntry = new DirectoryEntry(
        adsiPath, bindGroup, bindPassword);

```

Listing 374: Methode zum Auflisten der Benutzergruppen eines Computers

```

try
{
    // Alle dem Computer untergeordneten Objekte durchgehen
    foreach(DirectoryEntry de in computerEntry.Children)
    {
        // Überprüfen, ob es sich um ein Group-Objekt handelt
        if (de.SchemaClassName.ToLower() == "group")
        {
            // Neues Group-Objekt erzeugen und initialisieren
            Group group = new Group();
            group.Name = de.Name;
            try {group.Description = (string)de.Properties[
                "Description"].Value;} catch {}

            // Group-Objekt an die Auflistung anfügen
            groups.Add(group);
        }
    }
}
finally
{
    // ADSI-Objekt freigeben
    computerEntry.Dispose();
}

// Benutzer-Auflistung zurückgeben
return groups;
}

```

Listing 374: Methode zum Auflisten der Benutzergruppen eines Computers (Forts.)

Listing 375 zeigt eine Anwendung der `EnumGroups`-Methode, bei der die Daten aller lokalen Gruppen an der Konsole ausgegeben werden.

Das Programm erfordert den Import der Namensräume `System` und `System.Collections.Specialized`.

```

string domainName = null;
string machineName = null;
string bindUser = null;
string bindPassword = null;

// Gruppen abfragen
try
{
    Groups groups = EnumGroups(domainName, machineName,
        bindUser, bindPassword);

    // Benutzer durchgehen und deren Daten ausgeben
    for (int i = 0; i < groups.Count; i++)

```

Listing 375: Anwendung der EnumGroups-Methode

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    {
        Console.WriteLine();
        Console.WriteLine("Name: {0}", groups[i].Name);
        Console.WriteLine("Beschreibung: {0}", groups[i].Description);
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 375: Anwendung der EnumGroups-Methode (Forts.)

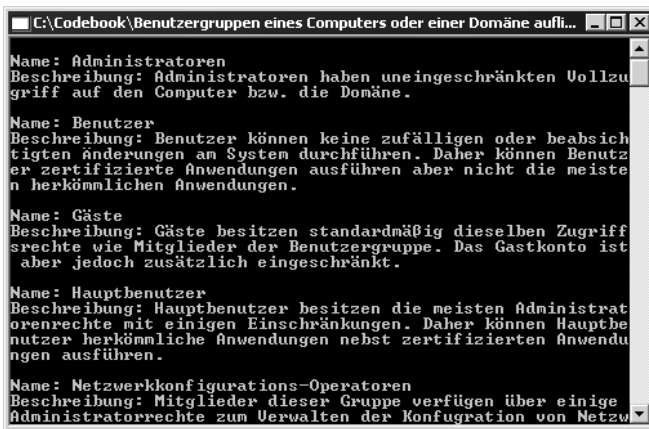


Abbildung 144: Das Beispielprogramm listet alle lokalen Gruppen auf

200 Benutzer ermitteln, die einer Gruppe angehören

Wenn Sie abfragen wollen, welche Benutzer einer Gruppe angehören, können Sie dazu ADSI verwenden. Über ein `DirectoryEntry`-Objekt können Sie ein ADSI-Objekt für eine Gruppe erzeugen. Dazu verwenden Sie idealerweise den ADSI-Provider *WinNT*. Das Erzeugen eines Objekts für die Gruppe *Administratoren* auf dem Computer *Zaphod* sieht z.B. so aus:

```

string adsiPath = "WinNT://Zaphod/Administratoren,group";
string bindUser = "Administrator";
string bindPassword = "bandit";
DirectoryEntry groupEntry = new DirectoryEntry(adsiPath, bindUser,
    bindPassword);

```

Im Beispiel habe ich wie in den vorhergehenden Rezepten bei der Erzeugung des `DirectoryEntry`-Objekts einen Benutzer (und dessen Passwort) angegeben, der für die Authentifizierung gegenüber dem Verzeichnisdienst verwendet werden soll. Würden Sie die letzten zwei Argumente weglassen, würde ADSI den aktuell in Windows eingeloggten Benutzer verwenden (der u. U. nicht über ausreichende Rechte verfügt).

Wollen Sie die Domäne (oder Arbeitsgruppe) des Computers mit einbeziehen (was das Auslesen des Gruppen-Objekts um einiges beschleunigt!), geben Sie diese vor dem Computernamen an:

```
string adsiPath = "WinNT://Galaxy/Zaphod/Administratoren,group";
```

Die vorstehenden Beispiele repräsentieren lokale Gruppen. Zum Auslesen einer globalen (Domänen-)Gruppe geben Sie keinen Computernamen an:

```
string adsiPath = "WinNT://Galaxy/Administratoren,group";
```

Das so erzeugte `DirectoryEntry`-Objekt ist leider nur ein generisches Objekt. Das tatsächlich referenzierte ADSI-Objekt implementiert u.a. die COM-Schnittstellen `IADs` (msdn.microsoft.com/library/en-us/netdir/adsi/iads.asp) und `IADsGroup` (msdn.microsoft.com/library/en-us/netdir/adsi/iadsgroup.asp), auf deren Methoden Sie über das `DirectoryEntry`-Objekt nicht direkt zugreifen können. Dazu verwenden Sie die Methode `Invoke`, der Sie den Namen der aufzurufenden `IADsGroup`-Methode und ein `object`-Array mit den Argumenten übergeben.

Die Mitglieder einer Gruppe erreichen Sie über die `IADsGroup`-Methode `Members`, die Sie zwar auch über `Invoke` aufrufen können. Diese Methode gibt aber eine Instanz einer Klasse zurück, die die COM-Schnittstelle `IADsMembers` implementiert. Da diese in Ihrem Programm normalerweise nicht bekannt ist, können Sie die Rückgabe nicht ohne weiteres auswerten.

Die Lösung dieses Problems ist recht einfach: Sie referenzieren dazu die COM-Typbibliothek, die die ADSI-Schnittstellen beschreibt. Diese Typbibliothek finden Sie unter dem Namen *activeds.tlb* im Windows-Systemverzeichnis. In Visual Studio .NET legen Sie dazu einen Verweis auf die *Active DS Type Library* an (Abbildung 145).

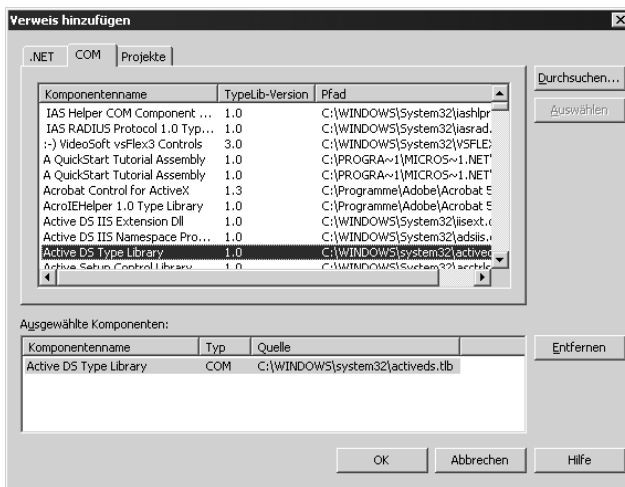


Abbildung 145: Anlegen eines Verweises auf die Active DS Type Library

Wenn Sie nun noch den Namensraum `ActiveDs` importieren, kennt Ihr Programm alle ADSI-Schnittstellen und damit auch die Schnittstelle `IADsMembers`. Dann können Sie `Invoke` problemlos ausführen:

```
IADsMembers adsMembers = (IADsMembers)groupEntry.Invoke("Members");
```

Da die Mitglieder eines `IADsGroup`-Objekts auch andere Objekte als `IADsUser`-Instanzen sein können, müssen Sie die `Members`-Auflistung noch filtern. Danach können Sie diese durchgehen und alle Gruppen-Benutzer auslesen.

Die Methode `EnumGroupMembers` in Listing 376 setzt diese Technik ein. Sie setzt zunächst an Hand der übergebenen Informationen zur Domäne, zum Computer und zur Gruppe den ADSI-Pfad für den WinNT-Provider zusammen und erzeugt mit diesem Pfad ein `DirectoryEntry`-Objekt. Die Argumente `bindUser` und `bindPassword` werden dabei für die Authentifizierung gegenüber ADSI übergeben. Über die `Invoke`-Methode des `DirectoryEntry`-Objekts wird dann die `Members`-Methode aufgerufen und die Rückgabe in einer Variablen vom Typ `IADsMembers` referenziert. Über die `Filter`-Eigenschaft dieses Objekts reduziert `EnumGroupMembers` die in der Auflistung referenzierten Objekte auf Instanzen der Schema-Klasse `User`. `User` ist die Klasse des WinNT-Providers, die Benutzerinformationen speichert und u.a. die Schnittstelle `IADsUser` implementiert. Danach geht `EnumGroupMembers` die ermittelte Auflistung in einer normalen `foreach`-Schleife durch und hängt die Namen der Benutzer an eine `StringCollection` an, die schließlich zurückgegeben wird.

`EnumGroupMembers` erfordert die Referenzierung der ADSI-Bibliothek *activeds.tlb* und den Import der Namensräume `System`, `System.DirectoryServices`, `System.Collections`, `System.Collections.Specialized` und `ActiveDs`.

```
public static StringCollection EnumGroupMembers(string domainName,
    string machineName, string groupName, string bindUser, string bindPassword)
{
    // StringCollection für die ermittelten Benutzernamen erzeugen
    StringCollection users = new StringCollection();

    // DirectoryEntry-Objekt für die Gruppe holen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string adsiPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        groupName + ".group";
    DirectoryEntry groupEntry = new DirectoryEntry(adsiPath, bindUser,
        bindPassword);

    try
    {
        // Member dieser Gruppe einlesen
```

Listing 376: Methode zum Auflisten aller Benutzer einer Gruppe

```

        IADsMembers adsMembers = (IADsMembers)groupEntry.Invoke("Members");

        // Member so filtern, dass nur User-Objekte übrig bleiben
        adsMembers.Filter = new object[] {"user"};

        // Benutzer durchgehen
        foreach (IADsUser adsUser in adsMembers)
        {
            users.Add(adsUser.Name);
        }
    }
    finally
    {
        // ADSI-Objekt freigeben
        groupEntry.Dispose();
    }

    // Die ermittelten Benutzer zurückgeben
    return users;
}

```

Listing 376: Methode zum Auflisten aller Benutzer einer Gruppe (Forts.)

Ausnahmen, die beim Auslesen der Gruppe entstehen können, werden innerhalb dieser Methode übrigens nicht abgefangen und somit einfach an den Aufrufer weitergereicht. Die Fehlermeldungen, die ADSI liefert, sind in unserem Fall recht aussagekräftig.

Das folgende Listing zeigt eine einfache Anwendung der `EnumGroupMembers`-Methode. Das Programm benötigt den Import der Namensräume `System` und `System.Collections.Specialized`.

```

string domainName = "Arbeitsgruppe";
string machineName = "Zaphod";
string groupName = "Administratoren";
string bindUser = null;
string bindPassword = null;

try
{
    StringCollection users = EnumGroupMembers(domainName, machineName,
        groupName, bindUser, bindPassword);
    Console.WriteLine("Benutzer der Gruppe '" + groupName + "':");
    for (int i = 0; i < users.Count; i++)
    {
        Console.WriteLine(users[i]);
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

Listing 377: Anwendung der Methode zum Auflisten aller Benutzer einer Gruppe

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

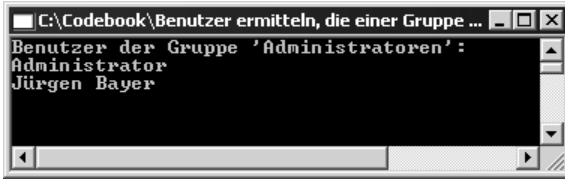


Abbildung 146: Das Beispielprogramm listet alle Benutzer der Gruppe Administratoren auf

201 Gruppen eines Benutzers abfragen

Wenn Sie ermitteln wollen, welchen Gruppen ein Benutzer angehört, können Sie wie in den vorhergehenden Rezepten ADSI verwenden.

Die Abfrage der Gruppen eines Benutzers ist ähnlich dem vorhergehenden Rezept etwas problematisch. Sie können zwar über den ADSI-Provider *WinNT* ein *User*-Objekt für den Benutzer referenzieren, das die ADSI-Schnittstelle *IADsUser* implementiert. Dessen Gruppen erreichen Sie aber über die *IADsUser*-Methode *Groups*, die ein Objekt zurückgibt, das die Schnittstelle *IADsMembers* implementiert. Die im generischen *DirectoryEntry*-Objekt nicht bekannte *Groups*-Methode können Sie zwar über *Invoke* aufrufen, die zurückgegebene *IADsMembers*-Instanz ist aber nicht in Ihrem Programm bekannt.

Als Lösung dieses Problems referenzieren Sie wieder die COM-Typbibliothek *activeds.tlb*, wie ich es im Rezept 200 gezeigt habe. Der Rest entspricht fast ebenso diesem Rezept, nur dass Sie hier statt der Benutzer einer Gruppe die Gruppen eines Benutzers abfragen.

Die im Folgenden beschriebene Methode *EnumUserGroups* setzt dies um. Übergeben wird, wie schon in den vorhergehenden Rezepten, zunächst ein Domänen- und ein Computernamen. Geben Sie nur den Domänenname an, sprechen Sie einen globalen (Domänen-)Benutzer an, geben Sie den Computernamen (und optional den Namen der Domäne) an, ermitteln Sie die Gruppen eines lokalen Benutzers. Der Name des Benutzers wird am Argument *userName* festgelegt. In den letzten Argumenten können Sie Informationen zu dem Benutzerkonto übergeben, das für die Authentifizierung gegenüber dem Verzeichnisdienst verwendet werden soll.

EnumUserGroups geht ähnlich *EnumGroupMembers* vor, nur dass nun ein ADSI-Objekt für den Benutzer erzeugt und dessen *Groups*-Auflistung durchgegangen wird:

Zum Kompilieren dieser Methode müssen Sie die ADSI-Bibliothek *activeds.tlb* referenzieren und die Namensräume *System*, *System.DirectoryServices*, *System.Collections*, *System.Collections.Specialized* und *ActiveDs* importieren.

```
public static StringCollection EnumUserGroups(string domainName,
    string machineName, string userName, string bindUser, string bindPassword)
{
    // StringCollection für die ermittelten Gruppennamen
    StringCollection groups = new StringCollection();
```

Listing 378: Methode zum Auflisten aller Gruppen eines Benutzers

```

// DirectoryEntry-Objekt für den Benutzer erzeugen
if (domainName == null && machineName == null)
    machineName = Environment.MachineName;
string adsiPath = "WinNT://" +
    (domainName != null ? domainName + "/" : "") +
    (machineName != null ? machineName + "/" : "") +
    userName + ",user";
DirectoryEntry userEntry = new DirectoryEntry(adsiPath,
    bindUser, bindPassword);

try
{
    // Gruppen dieses Benutzers einlesen
    IADsMembers adsMembers = (IADsMembers)userEntry.Invoke("Groups");

    // Member so filtern, dass nur Group-Objekte übrig bleiben
    adsMembers.Filter = new object[] { "group" };

    // Gruppen durchgehen
    foreach (IADsGroup adsGroup in adsMembers)
    {
        groups.Add(adsGroup.Name);
    }
}
finally
{
    // ADSI-Objekt freigeben
    userEntry.Dispose();
}

// Die ermittelten Gruppen zurückgeben
return groups;
}

```

Listing 378: Methode zum Auflisten aller Gruppen eines Benutzers (Forts.)

Listing 379 zeigt eine Anwendung der `EnumUserGroups`-Methode. Das Programm erfordert den Import der Namensräume `System` und `System.Collections.Specialized`.

```

string domainName = null;
string machineName = null;
string userName = "Administrator";
string bindUser = null;
string bindPassword = null;

try
{
    StringCollection groups = EnumUserGroups(
        domainName, machineName, userName, bindUser, bindPassword);
    Console.WriteLine("Gruppen des Benutzers '" + userName + "':");
}

```

Listing 379: Anwendung der `EnumUserGroups`-Methode

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
for (int i = 0; i < groups.Count; i++)
{
    Console.WriteLine(groups[i]);
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

Listing 379: Anwendung der `EnumUserGroups`-Methode (Forts.)

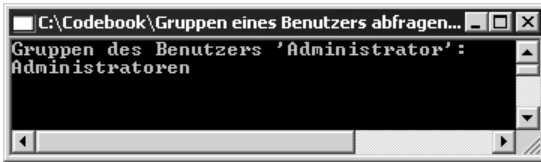


Abbildung 147: Auflisten der Gruppen des Benutzers »Administrator«

202 Eigenschaften eines Benutzers auslesen

Benutzerkonten besitzen einige Eigenschaften, die Sie über den Verzeichnisdienst *WinNT* abrufen können, wobei Sie wie in den vorhergehenden Rezepten ADSI verwenden. Die verfügbaren Methoden und Eigenschaften eines damit referenzierten `User`-Objekts beschreibt Microsoft an der Adresse msdn.microsoft.com/library/en-us/netdir/adsi/adsi_objects_of_winnt.asp.

Nicht allzu einfach ist das Herausfinden der Eigenschaften, die ein `User`-Objekt des WinNT-Providers besitzt. Wie bei ADSI üblich bestimmt eine Schema-Klasse die obligatorischen und optionalen Eigenschaften eines solchen Objekts. Obligatorische Eigenschaften müssen in Objekten implementiert werden, optionale nicht. Ein `User`-Objekt besteht mit Ausnahme der Eigenschaft `Name` ausschließlich aus optionalen Eigenschaften. Beim Herausfinden dieser Eigenschaften hilft die Microsoft-Dokumentation der Schemata des WinNT-Providers (msdn.microsoft.com/library/en-us/netdir/adsi/winnt_schema.asp).

In reinem ADSI würden Sie über spezielle *Get*- und *Put*-Methoden auf die Eigenschaften eines `User`-Objekts zugreifen. Wenn Sie ein `DirectoryEntry`-Objekt zum Zugriff auf ein solches Objekt verwenden, können Sie die Eigenschaften über dessen `Properties`-Auflistung lesen. Diese Auflistung gibt einfach `null` zurück, wenn Sie den Namen einer Eigenschaft übergeben, die nicht im Objekt existiert.

Die Klasse `User` in Listing 380 implementiert zunächst die möglichen Eigenschaften eines `User`-Objekts.

Zum Kompilieren dieser Klasse und des folgenden Programmcodes müssen Sie die Namensräume `System` und `System.DirectoryServices` importieren.

```

public class User
{
    // Direkte Eigenschaften
    public string Name;
    public string FullName;
    public string Description;
    public int UserFlags;
    public string Parameters;
    public int MaxStorage;
    public string HomeDirectory;
    public string HomeDirDrive;
    public string LoginScript;
    public string Profile;
    public int MinPasswordLength;
    public int MinPasswordAge;
    public int MaxPasswordAge;
    public int PasswordAge;
    public bool PasswordExpired;
    public int BadPasswordAttempts;
    public int PasswordHistoryLength;
    public DateTime LastLogin;
    public DateTime LastLogoff;
    public int MaxLogins;
    public string LoginWorkstations;
    public byte[] LoginHours;
    public DateTime AccountExpirationDate;
    public int RasPermissions;
    public int LockoutObservationInterval;
    public int PrimaryGroupID;
    public byte[] objectSid;
    // Eigenschaften, deren Wert sich aus der Eigenschaft UserFlags
    // berechnet (Eingeschränkte Auswahl!). Siehe
    // msdn.microsoft.com/library/en-us/netdir/adsi/ads_user_flag_enum.asp
    public bool AccountDisabled;
    public bool AccountLocked;
    public bool HomeDirRequired;
    public bool PasswordNotRequired;
    public bool PasswordCantChange;
    public bool PasswordNotExpires;
    public bool EncryptedPasswordAllowed;
    public bool LocalAccount;
    public bool NormalAccount;
    public bool SmartcardRequired;
}

```

Listing 380: Klasse mit den Eigenschaften für einen Benutzer

Die private Methode `GetADSIProperty` erleichtert das spätere Auslesen der optionalen Eigenschaften. Diese Methode gibt den Wert einer Eigenschaft des übergebenen `DirectoryEntry`-Objekts zurück, wenn diese gelesen werden konnte. Für den Fall, dass die `Properties`-Auflistung für die abzufragende Eigenschaft `null` speichert, gibt `GetADSIProperty` einen zum erwarteten Datentyp (der im letzten Argument übergeben wird) passenden Wert zurück.

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

private static object GetADSIProperty(DirectoryEntry entry,
    string propertyName, Type t)
{
    // Lesen der Eigenschaft
    object propertyValue = entry.Properties[propertyName].Value;

    if (propertyValue != null)
    {
        // Wenn ein Wert gespeichert ist: diesen zurückgeben
        // (boolesche Werte müssen konvertiert werden)
        if (t == typeof(bool))
            return (int)propertyValue == 0 ? false : true;
        else
            return propertyValue;
    }
    else
    {
        // Wenn kein Wert gespeichert ist: je nach dem übergebenen Typ
        // einen passenden Leerwert zurückgeben
        if (t == typeof(string))
            return null;
        else if (t == typeof(DateTime))
            return new DateTime(0);
        else if (t == typeof(int))
            return 0;
        else if (t == typeof(bool))
            return false;
        else
            return null;
    }
}

```

Listing 381: Methode zum fehlertoleranten Lesen von ADSI-Eigenschaften

Die Methode `GetUser` liest die Eigenschaften eines Benutzers aus und gibt eine Instanz der Klasse `User` zurück. Ähnlich den Methoden der vorhergehenden Rezepte erwartet diese Methode einen Domänennamen, den Namen des Computers, den Namen des Benutzers und Informationen zu dem Benutzerkonto, das für die Authentifizierung gegenüber dem WinNT-Provider verwendet werden soll. Um globale Benutzerkonten auszulesen, geben Sie nur den Domänennamen und am Argument `machineName` `null` an. Für lokale Benutzer geben Sie den Computernamen und optional auch den Domänennamen an (der ansonsten auch mit `null` belegt werden kann).

`GetUser` erzeugt zunächst ein `DirectoryEntry`-Objekt für den Benutzer und liest dann dessen Eigenschaften über die Methode `GetADSIProperty` ein. Die meisten Eigenschaften können direkt eingelesen werden. Einige Eigenschaften, wie z.B. die Information darüber, ob das Konto deaktiviert ist, werden allerdings im Feld `UserFlags` in Form eines Bitmusters gespeichert. Die dazu verwendeten Konstanten beschreibt Microsoft an der Adresse msdn.microsoft.com/library/en-us/netdir/adsi/ads_user_flag_enum.asp. `GetUser` fragt die wichtigsten dieser Flags ab und speichert das Ergebnis in entsprechenden booleschen Eigenschaften der `User`-Klasse.

Das erzeugte und mit den eingelesenen Eigenschaften initialisierte User-Objekt wird schließlich zurückgegeben.

```

public static User GetUser(string domainName, string machineName,
    string userName, string bindUser, string bindPassword)
{
    // Konstanten für die Flags eines Benutzers (Eingeschränkte Auswahl!)
    const int ADS_UF_ACCOUNTDISABLE = 0x0002;
    const int ADS_UF_LOCKOUT = 0x0010;
    const int ADS_UF_HOMEDIR_REQUIRED = 0x0008;
    const int ADS_UF_PASSWD_NOTREQD = 0x0020;
    const int ADS_UF_PASSWD_CANT_CHANGE = 0x0040;
    const int ADS_UF_DONT_EXPIRE_PASSWD = 0x10000;
    const int ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED = 0x0080;
    const int ADS_UF_TEMP_DUPLICATE_ACCOUNT = 0x0100;
    const int ADS_UF_NORMAL_ACCOUNT = 0x0200;
    const int ADS_UF_SMARTCARD_REQUIRED = 0x40000;

    // Gültigen Rechnernamen für den lokalen Computer erzeugen
    // falls weder die Domäne noch der Rechnername übergeben wurden
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;

    // DirectoryEntry-Objekt für den Benutzer erzeugen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string adsiPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        userName + ",user";
    DirectoryEntry userEntry = new DirectoryEntry(adsiPath,
        bindUser, bindPassword);

    // User-Objekt erzeugen
    User user = new User();

    try
    {
        // Eigenschaften dieses Benutzers einlesen
        user.Name = userName;
        user.Description = (string)GetADSIProperty(
            userEntry, "Description", typeof(string));
        user.FullName = (string)GetADSIProperty(
            userEntry, "FullName", typeof(string));
        user.UserFlags = (int)GetADSIProperty(
            userEntry, "UserFlags", typeof(int));
        user.Parameters = (string)GetADSIProperty(
            userEntry, "Parameters", typeof(string));
        user.MaxStorage = (int)GetADSIProperty(
            userEntry, "MaxStorage", typeof(int));
    }
}

```

Listing 382: Methode zum Einlesen der Eigenschaften eines Windows-Benutzers

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

user.HomeDirectory = (string)GetADSIProperty(
    userEntry, "HomeDirectory", typeof(string));
user.HomeDirDrive = (string)GetADSIProperty(
    userEntry, "HomeDirDrive", typeof(string));
user.LoginScript = (string)GetADSIProperty(
    userEntry, "LoginScript", typeof(string));
user.Profile = (string)GetADSIProperty(
    userEntry, "Profile", typeof(string));
user.MinPasswordLength = (int)GetADSIProperty(
    userEntry, "MinPasswordLength", typeof(int));
user.MinPasswordAge = (int)GetADSIProperty(
    userEntry, "MinPasswordAge", typeof(int));
user.MaxPasswordAge = (int)GetADSIProperty(
    userEntry, "MaxPasswordAge", typeof(int));
user.PasswordAge = (int)GetADSIProperty(
    userEntry, "PasswordAge", typeof(int));
user.PasswordExpired = (bool)GetADSIProperty(
    userEntry, "PasswordExpired", typeof(bool));
user.BadPasswordAttempts = (int)GetADSIProperty(
    userEntry, "BadPasswordAttempts", typeof(int));
user.PasswordHistoryLength = (int)GetADSIProperty(
    userEntry, "PasswordHistoryLength", typeof(int));
user.LastLogin = (DateTime)GetADSIProperty(
    userEntry, "LastLogin", typeof(DateTime));
user.LastLogoff = (DateTime)GetADSIProperty(
    userEntry, "LastLogoff", typeof(DateTime));
user.MaxLogins = (int)GetADSIProperty(
    userEntry, "MaxLogins", typeof(int));
user.LoginHours = (byte[])GetADSIProperty(
    userEntry, "LoginHours", typeof(byte[]));
user.RasPermissions = (int)GetADSIProperty(
    userEntry, "RasPermissions", typeof(int));
user.LockoutObservationInterval = (int)GetADSIProperty(
    userEntry, "LockoutObservationInterval", typeof(int));
user.PrimaryGroupID = (int)GetADSIProperty(
    userEntry, "PrimaryGroupID", typeof(int));
user.objectSid = (byte[])GetADSIProperty(
    userEntry, "objectSid", typeof(byte[]));
// Auswerten der User-Flags
if((user.UserFlags & ADS_UF_ACCOUNTDISABLE) != 0)
    user.AccountDisabled = true;
if((user.UserFlags & ADS_UF_DONT_EXPIRE_PASSWD) != 0)
    user.PasswordNotExpires = true;
if((user.UserFlags & ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED) != 0)
    user.EncryptedPasswordAllowed = true;
if((user.UserFlags & ADS_UF_HOMEDIR_REQUIRED) != 0)
    user.HomeDirRequired = true;
if((user.UserFlags & ADS_UF_LOCKOUT ) != 0)
    user.AccountLocked = true;
if((user.UserFlags & ADS_UF_NORMAL_ACCOUNT) != 0)
    user.NormalAccount = true;
if((user.UserFlags & ADS_UF_PASSWD_CANT_CHANGE) != 0)

```

Listing 382: Methode zum Einlesen der Eigenschaften eines Windows-Benutzers (Forts.)

```

        user.PasswordCantChange = true;
        if((user.UserFlags & ADS_UF_PASSWD_NOTREQD) != 0)
            user.PasswordNotRequired = true;
    }
    finally
    {
        // ADSI-Objekt freigeben
        userEntry.Dispose();
    }

    // Das User-Objekt zurückgeben
    return user;
}

```

Listing 382: Methode zum Einlesen der Eigenschaften eines Windows-Benutzers (Forts.)

Listing 383 zeigt eine Anwendung der `GetUser`-Methode. In diesem Beispiel werden nur die wichtigsten Daten an der Konsole ausgegeben.

```

string domainName = null;
string machineName = null;
string userName = "Administrator";
string bindUser = null;
string bindPassword = null;

// Benutzerdaten auslesen
User user = GetUser(domainName, machineName, userName, bindUser,
    bindPassword);

// Die wichtigsten Daten ausgeben
Console.WriteLine("Name: {0}", userName);
Console.WriteLine("Description: {0}", user.Description);
Console.WriteLine("FullName: {0}", user.FullName);
Console.WriteLine("AccountDisabled: {0}", user.AccountDisabled);
Console.WriteLine("AccountLocked: {0}", user.AccountLocked);
Console.WriteLine("HomeDirRequired: {0}", user.HomeDirRequired);
Console.WriteLine("PasswordNotRequired: {0}", user.PasswordNotRequired);
Console.WriteLine("PasswordCantChange: {0}", user.PasswordCantChange);
Console.WriteLine("PasswordNotExpires: {0}", user.PasswordNotExpires);
Console.WriteLine("EncryptedPasswordAllowed: {0}", user.EncryptedPasswordAllowed);
Console.WriteLine("LocalAccount: {0}", user.LocalAccount);
Console.WriteLine("NormalAccount: {0}", user.NormalAccount);
Console.WriteLine("MaxStorage: {0}", user.MaxStorage);
Console.WriteLine("HomeDirectory: {0}", user.HomeDirectory);
Console.WriteLine("HomeDirDrive: {0}", user.HomeDirDrive);
Console.WriteLine("LoginScript: {0}", user.LoginScript);
Console.WriteLine("Profile: {0}", user.Profile);
Console.WriteLine("PasswordExpired: {0}", user.PasswordExpired);

```

Listing 383: Anwendung der GetUser-Methode

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

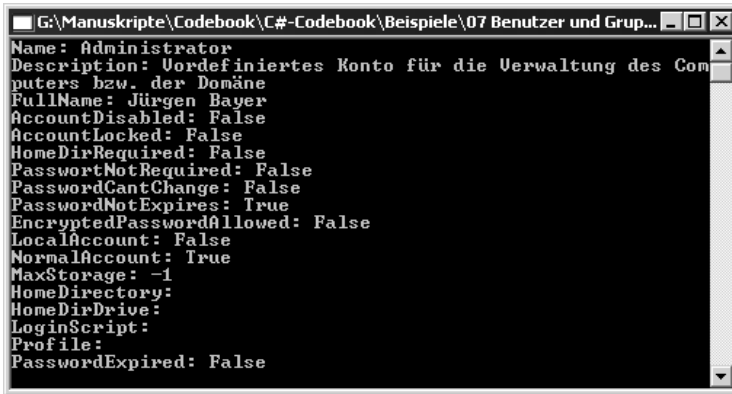


Abbildung 148: Das Beispielprogramm zeigt die wichtigsten Daten des Benutzers »Administrator« an

203 Gruppe anlegen

Benutzergruppen können Sie relativ problemlos über ADSI anlegen. Dazu erzeugen Sie zunächst ein `DirectoryEntry`-Objekt für den Computer, wobei Sie idealerweise den WinNT-Provider verwenden.

Eine `DirectoryEntry`-Instanz bietet über die `Add`-Methode der `Children`-Auflistung die Möglichkeit, untergeordnete Objekte zu erzeugen. Dazu geben Sie am ersten Argument den Namen des Objekts und am zweiten den Klassennamen an. `Add` gibt eine Referenz auf das erzeugte Objekt zurück. Über dessen `Properties`-Auflistung können Sie dann die Eigenschaften definieren. Da ADSI-Objekte immer im Arbeitsspeicher gecached werden, müssen Sie noch die Methode `CommitChanges` aufrufen um die Änderungen dauerhaft zu schreiben.

Die Methode `AddGroup` setzt diese Technik ein. Das Argument `groupName` bestimmt den Namen der Gruppe, das Argument `groupDescription` die Beschreibung. Die anderen Argumente dieser Methode entsprechen im Wesentlichen denen der vorhergehenden ADSI-Rezepte.

`AddGroup` erzeugt zunächst ein `DirectoryEntry`-Objekt für den Computer bzw. die Domäne und ruft dann die `Add`-Methode der `Children`-Auflistung auf um eine neue Gruppe zu erzeugen. Dieser Methode werden der Name des neuen Objekts und der Name der ADSI-Schema-Klasse `Group` übergeben. Über die zurückgegebene Referenz auf das erzeugte Objekt beschreibt `AddGroup` dann die Eigenschaft `Description`. Die dabei verwendete `Properties`-Auflistung gibt über ihren Indizierer eine Referenz auf eine weitere Auflistung vom Typ `PropertyValueCollection` zurück. Der Grund dafür, dass der Indizierer der `Properties`-Auflistung nicht einen einfachen Wert verwaltet, ist, dass Eigenschaften in ADSI auch aus mehreren Werten bestehen können (ähnlich einem Array). Deshalb muss `AddGroup` auch die `Add`-Methode der über `Properties` erreichten Werte-Auflistung aufrufen, um die Beschreibung der Gruppe am ersten Wert-Element der Eigenschaft `Description` abzulegen. Die Beachtung dieser Technik ist sehr wichtig, denn der Versuch, direkt in die Eigenschaft `Value` des vom Indizierer der `Properties`-Auflistung zurückgegebenen `PropertyValueCollection`-Objekts zu schreiben, führt zu der wenig aussagekräftigen Ausnahme »Unbekannter Fehler« beim Wegschreiben der geänderten Daten.

Da ADSI alle Objekte zunächst nur im Arbeitsspeicher verwaltet und nicht automatisch in den Verzeichnisdienst zurückschreibt, ruft `AddGroup` schließlich noch die `CommitChanges`-Methode des `DirectoryEntry`-Objekts auf.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.DirectoryServices` importieren.

```
public static void AddGroup(string domainName, string machineName,
    string groupName, string groupDescription, string bindUser,
    string bindPassword)
{
    // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string adsiPath = "WinNT://";
    if (domainName != null && machineName != null)
        adsiPath += domainName + "/" + machineName + ",computer";
    else if (machineName != null)
        adsiPath += machineName + ",computer";
    else if (domainName != null)
        adsiPath += domainName + ",domain";
    DirectoryEntry computerEntry = new DirectoryEntry(
        adsiPath, bindUser, bindPassword);

    try
    {
        // Gruppe hinzufügen und Eigenschaften definieren
        DirectoryEntry groupEntry =
            computerEntry.Children.Add(groupName, "group");
        groupEntry.Properties["description"].Add(groupDescription);
        groupEntry.CommitChanges();
    }
    finally
    {
        computerEntry.Dispose();
    }
}
```

Listing 384: Methode zum Hinzufügen einer Gruppe

Sofern bei der Ausführung dieser Methode keine Ausnahme eintritt, wurde die Gruppe erfolgreich erzeugt. Ausnahmen werden nicht abgefangen, da diese recht aussagekräftig sind und einfach an den Aufrufer weitergereicht werden können. Beim Anlegen einer lokalen Gruppe mit einem Namen, der bereits für eine Gruppe oder ein Benutzerkonto verwendet wird, erzeugt `AddUser` z. B. eine Ausnahme vom Typ `System.Runtime.InteropServices.COMException` mit der Meldung »Die angegebene lokale Gruppe ist bereits vorhanden« bzw. »Das Konto existiert bereits«.

204 Benutzer anlegen

Das Anlegen eines Benutzerkontos ist ähnlich dem Anlegen einer Gruppe im vorhergehenden Rezept.

Der in Listing 385 beschriebenen Methode `AddUser` werden neben den bereits in den anderen ADSI-Rezepten verwendeten Argumenten der Name des neuen Benutzers (`userName`), der volle Name (`fullName`), eine Beschreibung (`description`), ein Profilordner (`profile`), ein Login-Skript (`loginScript`), das Home-Verzeichnis (`homeDirectory`), das Passwort (`password`), eine Info darüber, ob das Passwort vom Benutzer geändert werden kann (`cantChangePassword`), und eine Info darüber, ob das Passwort nie abläuft (`passwordDontExpires`), übergeben. Diese Argumente entsprechen im Wesentlichen den Möglichkeiten, die Sie über die Windows-Benutzeradministration haben.

`AddUser` erzeugt je nach den übergebenen Domänen- und Computernamen zunächst ein `DirectoryEntry`-Objekt für den Computer oder die Domäne. Über die `Add`-Methode der `Children`-Auflistung wird dann ein neues `User`-Objekt erzeugt. Wie bereits beim Erzeugen einer Gruppe werden dazu der Name des neuen Objekts und der Name der Schema-Klasse für dieses Objekt übergeben. Über die Referenz auf das so erzeugte `DirectoryEntry`-Objekt für den neuen Benutzer schreibt `AddUser` dann die Eigenschaften. Dabei muss wieder wie im vorhergehenden Rezept beachtet werden, dass dazu die `Add`-Methode der vom Indizierer der `Properties`-Auflistung zurückgegebenen `PropertyValueCollection` aufgerufen werden muss.

Die Informationen darüber, ob das Passwort geändert werden kann und ob dieses niemals verfällt, werden in der Eigenschaft `UserFlags` des `User`-Objekts über festgelegte Konstanten verwaltet. `AddUser` setzt diese Konstanten deswegen je nach den übergebenen Werten in einer `int`-Variablen zusammen und schreibt diese in die Eigenschaft. Da es sich um ein normales Konto handeln soll, wird zusätzlich das Flag `ADS_UF_NORMAL_ACCOUNT` gesetzt. Die möglichen Flags finden Sie an der Adresse msdn.microsoft.com/library/en-us/netdir/ads/ads_user_flag_enum.asp.

Schließlich übergibt `AddUser` noch das Passwort, was über den Aufruf der nativen Methode `SetPassword` geschieht. Native Methoden werden über die `Invoke`-Methode aufgerufen. `AddUser` übergibt dieser Methode am ersten Argument den Namen der nativen Methode. Am zweiten Argument nimmt ein `object`-Array die zu übergebenden Argumente auf. In unserem Fall ist das einfach nur das Passwort.

Um die bisher lediglich im Arbeitsspeicher geänderten Daten in den Verzeichnisdienst zu schreiben, ruft `AddUser` dann noch die Methode `CommitChanges` auf.

`AddUser` benötigt den Import der Namensräume `System` und `System.DirectoryServices`.

```
public static void AddUser(string domainName, string machineName,
    string userName, string fullName, string description, string profile,
    string loginScript, string homeDirectory, string password,
    bool cantChangePassword, bool passwordDontExpires,
    string bindUser, string bindPassword)
{
```

Listing 385: Methode zum Hinzufügen eines Benutzers

```
// Konstanten für wichtige Flags des User-Objekts
const int ADS_UF_PASSWD_CANT_CHANGE = 0x0040;
const int ADS_UF_DONT_EXPIRE_PASSWD = 0x10000;
const int ADS_UF_NORMAL_ACCOUNT = 0x0200;

// DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
if (domainName == null && machineName == null)
    machineName = Environment.MachineName;
string adsiPath = "WinNT://";
if (domainName != null && machineName != null)
    adsiPath += domainName + "/" + machineName + ",computer";
else if (machineName != null)
    adsiPath += machineName + ",computer";
else if (domainName != null)
    adsiPath += domainName + ",domain";
DirectoryEntry computerEntry = new DirectoryEntry(
    adsiPath, bindUser, bindPassword);

try
{
    // Neues User-Objekt erzeugen
    DirectoryEntry userEntry = computerEntry.Children.Add(
        userName, "User");

    // Eigenschaften übergeben
    userEntry.Properties["FullName"].Add(fullName);
    userEntry.Properties["Description"].Add(description);
    userEntry.Properties["Profile"].Add(profile);
    userEntry.Properties["LoginScript"].Add(loginScript);
    userEntry.Properties["HomeDirectory"].Add(homeDirectory);

    // Flags definieren und übergeben
    int flags = ADS_UF_NORMAL_ACCOUNT;
    if (cantChangePassword)
        flags |= ADS_UF_PASSWD_CANT_CHANGE;
    if (passwordDontExpires)
        flags |= ADS_UF_DONT_EXPIRE_PASSWD;
    userEntry.Properties["UserFlags"].Add(flags);

    // Passwort setzen
    userEntry.Invoke("SetPassword", new object[] {password});

    // Geänderte Daten schreiben
    userEntry.CommitChanges();
}
finally
{
    computerEntry.Dispose();
}
```

Listing 385: Methode zum Hinzufügen eines Benutzers (Forts.)

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die beim Erzeugen eines User-Objekts möglichen Ausnahmen werden nicht abgefangen, da die Meldungen dieser Ausnahmen im Normalfall sehr aussagekräftig sind und deshalb einfach an den Aufrufer weitergereicht werden können. Wenn Sie z. B. einen Benutzer anlegen wollen, dessen Name bereits für eine Gruppe oder einen Benutzer verwendet wird, erhalten Sie eine Ausnahme vom Typ `System.Runtime.InteropServices.COMException` mit der Meldung »Die angegebene lokale Gruppe ist bereits vorhanden« bzw. »Das Konto existiert bereits«.

205 Eigenschaften eines Benutzers ändern

Wenn Sie die Eigenschaften eines Benutzers ändern wollen, können Sie dazu eine Methode verwenden, die dem Rezept 204 ähnlich ist. Im Unterschied zu dieser wird der Benutzer natürlich nicht erzeugt, sondern über ein `DirectoryEntry`-Objekt referenziert.

Die Methode `ChangeUser` in Listing 386 implementiert dies. Ich verzichte auf eine nähere Beschreibung, da die Arbeitsweise der Methode `CreateUser` sehr ähnlich ist. `ChangeUser` benötigt den Import der Namensräume `System` und `System.DirectoryServices`.

```
public static void ChangeUser(string domainName, string machineName,
    string userName, string fullName, string description, string profile,
    string loginScript, string homeDirectory, string password,
    bool cantChangePassword, bool passwordDontExpires, string bindUser,
    string bindPassword)
{
    // Konstanten für wichtige Flags des User-Objekts
    const int ADS_UF_PASSWD_CANT_CHANGE = 0x0040;
    const int ADS_UF_DONT_EXPIRE_PASSWD = 0x10000;
    const int ADS_UF_NORMAL_ACCOUNT = 0x0200;

    // DirectoryEntry-Objekt für den Benutzer erzeugen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string adsiPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        userName + ",user";
    DirectoryEntry userEntry = new DirectoryEntry(adsiPath,
        bindUser, bindPassword);

    try
    {
        // Eigenschaften übergeben
        userEntry.Properties["FullName"].Add(fullName);
        userEntry.Properties["Description"].Add(description);
        userEntry.Properties["Profile"].Add(profile);
        userEntry.Properties["LoginScript"].Add(loginScript);
        userEntry.Properties["HomeDirectory"].Add(homeDirectory);
```

Listing 386: Methode zum Ändern eines Benutzers

```

// Flags definieren und übergeben
int flags = ADS_UF_NORMAL_ACCOUNT;
if (cantChangePassword)
    flags |= ADS_UF_PASSWD_CANT_CHANGE;
if (passwordDontExpires)
    flags |= ADS_UF_DONT_EXPIRE_PASSWD;
userEntry.Properties["UserFlags"].Add(flags);

// Passwort setzen
userEntry.Invoke("SetPassword", new object[] {password});

// Geänderte Daten schreiben
userEntry.CommitChanges();
}
finally
{
    userEntry.Dispose();
}
}

```

Listing 386: Methode zum Ändern eines Benutzers (Forts.)

206 Benutzer einer Gruppe zuweisen

Über ADSI ist das Zuweisen eines Benutzers zu einer Gruppe recht einfach. Die Methode `AddUserToGroup` in Listing 387 macht genau dies. Die Argumente dieser Methode entsprechen im Wesentlichen denen der vorhergehenden Rezepte. Das Argument `groupName` bezeichnet die Gruppe, das Argument `userName` den Benutzer.

`AddUserToGroup` erzeugt zunächst ein `DirectoryEntry`-Objekt für die Gruppe, wobei der WinNT-Provider eingesetzt wird. Der Pfad zur Gruppe wird je nach den übergebenen Daten mit oder ohne Domäne bzw. Computernamen zusammengesetzt. Auf diese Weise können lokale und globale Gruppen angesprochen werden. Da beim Hinzufügen eines Benutzers dessen ADSI-Pfad benötigt wird, setzt `AddUserToGroup` diesen gleich auch entsprechend zusammen.

Danach ruft `AddUserToGroup` die native Methode `Add` des ADSI-Group-Objekts auf um den Benutzer anzufügen. Der Aufruf erfolgt über die `Invoke`-Methode des `DirectoryEntry`-Objekts, das die Gruppe repräsentiert. Dabei werden der Name der nativen Methode und ein `object`-Array mit den Argumenten übergeben. Zum Anfügen eines Benutzers erwartet `Add` die Übergabe eines ADSI-Pfads.

Der Aufruf der `Invoke`-Methode erfolgt in einer Ausnahmebehandlung, die Ausnahmen vom Typ `TargetInvocationException` abfängt. Der Grund dafür ist, dass `Invoke` bei Fehlern, die innerhalb von ADSI auftreten, eine Ausnahme vom Typ `TargetInvocationException` mit der nur wenig aussagekräftigen Fehlermeldung »Ein Aufrufziel hat einen Ausnahmefehler verursacht« erzeugt. Das ist z.B. dann der Fall, wenn ein Benutzer einer Gruppe angefügt werden soll, der dieser bereits angehört. Lediglich die innere Ausnahme, die bei meinen Versuchen vom Typ `COMException` war, enthält die (aussagekräftige) Beschreibung des eigentlichen Feh-

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

lers. Die Ausnahmebehandlung überprüft deshalb diese Existenz der inneren Ausnahme und gibt diese einfach weiter. Auf diese Weise kann der Aufrufer aufgetretene Fehler wesentlich einfacher auswerten.

AddUserToGroup **erfordert den Import der Namensräume** System, System.DirectoryServices und System.Reflection.

```
public static void AddUserToGroup(string domainName, string machineName,
    string userName, string groupName, string bindUser,
    string bindPassword)
{
    // DirectoryEntry-Objekt für die Gruppe holen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string groupPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        groupName + ".group";
    DirectoryEntry groupEntry = new DirectoryEntry(groupPath,
        bindUser, bindPassword);

    // Pfad zum Benutzer zusammensetzen
    string userPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        userName + ".user";

    // Benutzer der Gruppe anfügen
    try
    {
        groupEntry.Invoke("Add", new object[] {userPath});
    }
    // Abfangen der Ausnahmen, die bei Invoke auftreten können
    catch (TargetInvocationException ex)
    {
        // Die eigentliche Fehlermeldung steht häufig in der
        // inneren Ausnahme. Diese wird hier einfach weitergegeben
        if (ex.InnerException != null)
            throw ex.InnerException;
        else
            throw ex;
    }

    // Änderungen wegschreiben
    groupEntry.CommitChanges();
}
```

Listing 387: Methode zum Hinzufügen eines Benutzers zu einer Gruppe

207 Benutzer aus einer Gruppe entfernen

Das Entfernen eines Benutzers aus einer Gruppe entspricht fast komplett dem Hinzufügen eines Benutzers, wie ich es im Rezept 207 beschrieben habe. Der einzige Unterschied ist, dass Sie nun anstelle von `Add` die Methode `Remove` des `User`-Objekts aufrufen. Ich verzichte deswegen auf eine nähere Beschreibung der Methode `RemoveUserFromGroup`, die einen Benutzer aus einer Gruppe entfernt.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.DirectoryServices` und `System.Reflection` importieren.

```
public static void RemoveUserFromGroup(string domainName, string machineName,
    string userName, string groupName, string bindUser, string bindPassword)
{
    // DirectoryEntry-Objekt für die Gruppe erzeugen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string groupPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        groupName + ",group";
    DirectoryEntry groupEntry = new DirectoryEntry(groupPath,
        bindUser, bindPassword);

    // Pfad zum Benutzer zusammensetzen
    string userPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        userName + ",user";

    // Benutzer aus der Gruppe entfernen
    try
    {
        groupEntry.Invoke("Remove", new object[] {userPath});
    }
    // Abfangen der Ausnahmen, die bei Invoke auftreten können
    catch (TargetInvocationException ex)
    {
        // Die eigentliche Fehlermeldung steht häufig in der
        // inneren Ausnahme. Diese wird hier einfach weitergegeben
        if (ex.InnerException != null)
            throw ex.InnerException;
        else
            throw ex;
    }

    // Änderungen wegschreiben
    groupEntry.CommitChanges();
}
```

Listing 388: Methode zum Entfernen eines Benutzers aus einer Gruppe

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

208 Benutzer löschen

Zum Löschen eines Benutzers können Sie (wie bereits schon mehrfach ...) ADSI verwenden.

Die Methode `RemoveUser` löscht einen Benutzer, dessen Name am Argument `userName` übergeben wird. Sie erzeugt dazu abhängig von den Argumenten `domainName` und `computerName` ein `DirectoryEntry`-Objekt für den Computer bzw. die Domäne. Bei der Erzeugung dieses Objekts werden wie in den anderen ADSI-Rezepten Informationen zu dem Benutzerkonto übergeben, das für die Authentifizierung gegenüber dem Verzeichnisdienst verwendet werden soll. Damit kann die Methode auch ausgeführt werden, wenn der aktuell in Windows eingeloggte Benutzer nicht über ausreichende Rechte verfügt.

Über die `Remove`-Methode der `Children`-Auflistung des `DirectoryEntry`-Objekts wird der Benutzer später gelöscht. Da diese Methode ein `DirectoryEntry`-Objekt als Argument erwartet, erzeugt `RemoveUser` dieses zuvor für den angegebenen Benutzer. Ausnahmen werden nicht abgefangen, da diese sehr aussagekräftig sind und einfach an den Aufrufer weitergereicht werden können.

`RemoveUser` benötigt den Import der Namensräume `System`, `System.Reflection` und `System.DirectoryServices`.

```
public static void RemoveUser(string domainName, string machineName,
    string userName, string bindUser, string bindPassword)
{
    // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string adsiPath = "WinNT://";
    if (domainName != null && machineName != null)
        adsiPath += domainName + "/" +
            machineName + ",computer";
    else if (machineName != null)
        adsiPath += machineName + ",computer";
    else if (domainName != null)
        adsiPath += domainName + ",domain";
    DirectoryEntry computerEntry = new DirectoryEntry(
        adsiPath, bindUser, bindPassword);

    // DirectoryEntry-Objekt für den Benutzer erzeugen
    adsiPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        userName + ",user";
    DirectoryEntry userEntry = new DirectoryEntry(
        adsiPath, bindUser, bindPassword);

    // Benutzer vom Computer bzw. von der Domäne entfernen
    computerEntry.Children.Remove(userEntry);
}
```

Listing 389: Methode zum Löschen eines Benutzers

209 Gruppe löschen

Der Programmcode zum Löschen einer Gruppe entspricht in fast identischer Form dem Löschen eines Benutzers (Rezept 208). Die minimalen Unterschiede sind, dass der Gruppenname übergeben und statt dem `DirectoryEntry`-Objekt für den Benutzer eines für die Gruppe erzeugt wird. Deshalb verzichte ich auch auf eine weitere Beschreibung der Methode `RemoveGroup`.

Zum Kompilieren der `RemoveGroup`-Methode müssen Sie die Namensräume `System`, `System.Reflection` und `System.DirectoryServices` importieren.

```
public static void RemoveGroup(string domainName, string machineName,
    string groupName, string bindUser, string bindPassword)
{
    // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
    if (domainName == null && machineName == null)
        machineName = Environment.MachineName;
    string adsiPath = "WinNT://";
    if (domainName != null && machineName != null)
        adsiPath += domainName + "/" + machineName + ",computer";
    else if (machineName != null)
        adsiPath += machineName + ",computer";
    else if (domainName != null)
        adsiPath += domainName + ",domain";
    DirectoryEntry computerEntry = new DirectoryEntry(
        adsiPath, bindUser, bindPassword);

    // DirectoryEntry-Objekt für die Gruppe erzeugen
    adsiPath = "WinNT://" +
        (domainName != null ? domainName + "/" : "") +
        (machineName != null ? machineName + "/" : "") +
        groupName + ",group";
    DirectoryEntry groupEntry = new DirectoryEntry(
        adsiPath, bindUser, bindPassword);

    // Gruppe vom Computer bzw. von der Domäne entfernen
    computerEntry.Children.Remove(groupEntry);
}
```

Listing 390: Methode zum Löschen einer Gruppe

210 Den Namen des aktuellen Benutzers auslesen

Den Namen des aktuellen Benutzers erhalten Sie über die Eigenschaft `UserName` der Klasse `System.Environment`.

```
string currentUserName = System.Environment.UserName;
```

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Bei diesem Benutzer handelt es sich um den Benutzer, der dem Thread zugewiesen ist, von dem aus Sie den Programmcode aufrufen. Normalerweise ist das der Benutzer, der in Windows eingeloggt ist. Da Sie ein Programm aber auch im Kontext eines anderen Benutzers ausführen können (siehe Rezept 211), kann es sich auch um einen anderen Benutzer handeln. Eine Lösung, den Namen des in Windows eingeloggten Benutzers auch in solchen Programmen auszulesen, habe ich leider nicht ermitteln können. Die Windows-API-Funktion `GetUserName`, die sich als Lösung anbietet, liefert leider ebenfalls nur den Namen des Benutzers, der dem aktuellen Thread zugeordnet ist.

211 Ein Programm unter einem spezifischen Benutzerkonto ausführen

Normale Programme werden immer standardmäßig unter dem Konto des aktuell in Windows eingeloggten Benutzers ausgeführt. Lediglich Dienste können über den Dienste-Manager so konfiguriert werden, dass diese unter einem spezifischen Benutzerkonto laufen (normalerweise ist das das Konto *Lokales System*). Ein Programm greift immer mit den Rechten dieses Kontos auf Ressourcen (z.B. auf Dateien oder auf die Registry) zu. In geschützten Systemen, auf denen der Zugriff auf Ressourcen für die einzelnen Benutzer eingeschränkt ist, führt das immer dann zu Problemen, wenn ein Programm eine Ressource bearbeiten muss, für die das aktuell verwendete Benutzerkonto keine ausreichenden Rechte besitzt. So kann es z.B. sein, dass ein Programm Informationen in eine Protokolldatei schreiben muss, die der eingeloggte Benutzer nur lesen darf.

Dieses Problem können Sie lösen, indem Sie den Prozess einer Anwendung mit einem spezifischen Benutzerkonto personifizieren. Dazu rufen Sie einfach die `Impersonate`-Methode eines `WindowsIdentity`-Objekts auf, das einen Windows-Benutzer repräsentiert. Diese Methode gibt eine Referenz auf ein `WindowsImpersonationContext`-Objekt zurück, über dessen `Undo`-Methode Sie die Personifizierung später wieder rückgängig machen können. Dieser Teil der Problemlösung ist einfach.

Dummerweise bietet die `WindowsIdentity`-Klasse keine direkte Möglichkeit, eine Instanz für ein spezifisches Benutzerkonto zu erzeugen. Im Konstruktor können Sie lediglich ein Handle auf ein Windows-Benutzer-Token übergeben. Ein solches Token repräsentiert innerhalb von Windows die Rechte eines bestimmten Benutzers. Ein Benutzer-Token wird üblicherweise mit Windows-API-Funktionen verwendet. Und eine solche rufen Sie dann auch auf, um ein Token für ein Benutzerkonto zu erhalten.

Die Klasse `Logon` in Listing 391 setzt dies um. Die Methode `ImpersonateUser`, die aufgerufen werden kann um eine Anwendung mit einem spezifischen Benutzer zu personifizieren, ruft zunächst die API-Funktion `LogonUser` auf. Diese Funktion erzeugt für den Benutzer, dessen Benutzername, Domäne und Passwort an den ersten Argumenten übergeben werden, ein Benutzer-Token.

Am vierten Argument übergibt `ImpersonateUser` die Konstante `LOGON32_LOGON_NETWORK_CLEARTEXT` (die den Wert 8¹ besitzt). Damit wird erreicht, dass das erzeugte Token die Benutzerinformationen als Klartext zwischenspeichert. Bei Netzwerkzugriffen werden diese Informationen vom entfernten Rechner ausgelesen und für die dortige Authentifizierung verwendet. Nur wenn die Informationen auch vorhanden sind, besitzt ein Token die auf dem entfernten Rechner für das entsprechende Benutzerkonto vergebenen Rechte.

Das Argument `dwLogonProvider` der `LogonUser`-Funktion wird mit der Konstante `LOGON32_PROVIDER_DEFAULT` belegt, damit der voreingestellte Logon-Provider des Systems verwendet wird (alternativ könnten Sie unter Windows 2000 und XP auch den alten NT 4- oder NT 3.5-Provider verwenden). Im letzten Argument wird eine `int`-Variable übergeben, in die `LogonUser` den Handle des erzeugten Tokens schreibt.

Über die Auswertung der Rückgabe der Funktion überprüft `ImpersonateUser` dann, ob der Aufruf erfolgreich war. Im Fehlerfall liest die Methode den letzten API-Fehler aus, generiert über die API-Funktion `FormatMessage` eine dazu passende Fehlermeldung und wirft damit eine Ausnahme.

War `LogonUser` erfolgreich, erzeugt `ImpersonateUser` mit dem erhaltenen Token ein `WindowsPrincipal`-Objekt und personifiziert die aktuelle Anwendung im Anschluss über dessen `Impersonate`-Methode. `Impersonate` gibt eine Referenz auf ein `WindowsImpersonationContext`-Objekt zurück, über dessen `Undo`-Methode die Personifizierung rückgängig gemacht werden kann. Damit dies später möglich ist, speichert `ImpersonateUser` diese Referenz in der privaten, statischen Eigenschaft `winImpersonationContext` der Klasse.

Um das Rückgängigmachen der Personifizierung zu ermöglichen, enthält die Klasse `Logon` eine weitere Methode `UndoImpersonation`, die einfach nur überprüft, ob die Eigenschaft `winImpersonationContext` ein `WindowsImpersonationContext`-Objekt referenziert und im positiven Fall dessen `Undo`-Methode aufruft. Das Programm wird dann wieder unter dem Kontext des Benutzers ausgeführt, unter dem es vor dem Aufruf von `ImpersonateUser` ausgeführt wurde.

Zum Kompilieren dieser Klasse müssen Sie die Namensräume `System`, `System.Text`, `System.Runtime.InteropServices` und `System.Security.Principal` importieren

```
public class Logon
{
    /* Deklaration benötigter API-Funktionen und Konstanten */
    [DllImport("advapi32.dll", SetLastError=true)]
    private static extern int LogonUser(string lpszUsername,
```

Listing 391: Klasse mit Methoden zur Personifizierung von Anwendungen

1. In der Dokumentation der `Impersonate`-Methode wird der Wert von `LOGON32_LOGON_NETWORK_CLEARTEXT` fälschlicherweise mit 3 angegeben. Der originale Wert 8 ist in der C++-Headerdatei `winbase.h` definiert. Der Wert 3 steht hier für die Konstante `LOGON32_LOGON_NETWORK`. Diese Konstante ist laut der Dokumentation für High Performance Server zur Authentifizierung von Passwörtern im Klartext vorgesehen. Da das Benutzertoken die Benutzerinformationen nicht zwischenspeichert, wenn lediglich `LOGON32_LOGON_NETWORK` angegeben ist, besitzt dieses keinen Zugriff auf Netzwerk-Ressourcen.

```

        string lpzDomain, string lpzPassword, int dwLogonType,
        int dwLogonProvider, out int phToken);

[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, string lpSource,
    int dwMessageId, int dwLanguageId, StringBuilder lpBuffer, int nSize,
    string [] Arguments);

private const int LOGON32_LOGON_NETWORK_CLEARTEXT = 3;
private const int LOGON32_PROVIDER_DEFAULT = 0;
private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;

/* Eigenschaft, die ein bei einer Personifizierung erzeugtes
 * WindowsImpersonationContext-Objekt referenziert, über das die
 * Personifizierung wieder rückgängig gemacht werden kann */
private static WindowsImpersonationContext
    winImpersonationContext = null;

/* Methode zur Personifizierung der aktuellen Anwendung mit einem
 * speziellen Benutzerkonto. Funktioniert nur unter Windows NT, 2000, XP
 * und neueren Windows-Versionen */
public static void ImpersonateUser(string domain, string userName,
    string password)
{
    // Benutzer einloggen
    int userToken = 0;
    bool loggedOn = (LogonUser(userName, domain, password,
        LOGON32_LOGON_NETWORK_CLEARTEXT, LOGON32_PROVIDER_DEFAULT,
        out userToken) != 0);

    if (loggedOn == false)
    {
        // Fehler auslesen, in einen Text umwandeln und damit eine
        // Ausnahme werfen
        int apiError = Marshal.GetLastWin32Error();
        StringBuilder errorMessage = new StringBuilder(1024);
        FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, null, apiError,
            0, errorMessage, 1024, null);
        throw new Exception(errorMessage.ToString());
    }

    // WindowsIdentity mit diesem Token erzeugen
    WindowsIdentity identity = new WindowsIdentity((IntPtr)userToken);

    // Überprüfen, ob das erzeugte WindowsIdentity-Objekt den übergebenen
    // Benutzer repräsentiert
    string tokenUserName = null;
    if (userName.IndexOf("@") > 0)
    {
        // Benutzername im UPN-Format (user@DNS_domain_name)

```

Listing 391: Klasse mit Methoden zur Personifizierung von Anwendungen (Forts.)

```

        tokenUserName = userName.Replace('@', '\\');
    }
    else
    {
        // Normaler Benutzername
        tokenUserName = domain + "\\\" + userName;
    }

    // Prozess mit der ermittelten Windows-Identität personifizieren
    winImpersonationContext = identity.Impersonate();
}

/* Methode zum Rückgängigmachen der Personifizierung */
public static void UndoImpersonation()
{
    if (winImpersonationContext != null)
    {
        winImpersonationContext.Undo();
    }
}
}

```

Listing 391: Klasse mit Methoden zur Personifizierung von Anwendungen (Forts.)

Voraussetzung für die erfolgreiche Ausführung dieser Methoden ist:

- ▶ dass die Methoden unter Windows NT 4, Windows 2000, Windows XP oder einer neueren Windows-Version ausgeführt werden,
- ▶ dass der Prozess, von dem aus die Methoden aufgerufen werden, unter Windows 2000 (und wahrscheinlich auch unter NT, nicht aber unter XP) das Privileg `SE_TCB_NAME` (siehe unten) besitzt
- ▶ und dass das angegebene Benutzerkonto das Recht besitzt, sich interaktiv anzumelden.

Unter Windows 2000 ist es leider erforderlich, dass der Benutzer, unter dessen Konto das Programm ausgeführt wird, das Privileg `SE_TCB_NAME` besitzt. Dieses Privileg vergeben Sie über die lokalen Sicherheitseinstellungen (SYSTEMSTEUERUNG / VERWALTUNG / LOKALE SICHERHEITSRICHTLINIE / SICHERHEITSEINSTELLUNGEN / LOKALE RICHTLINIEN / ZUWEISEN VON BENUTZERRECHTEN). Dort wird dieses Privileg als `EINSETZEN ALS TEIL DES BETRIEBSSYSTEMS` bezeichnet (was eine etwas unklare Übersetzung für das originale »Act as part of the operating system« ist). Laut der Microsoft-Windows-2000-Dokumentation erlaubt dieses Privileg einem Prozess, sich als jeder Benutzer zu authentifizieren (was ja unser Ziel ist). Der Benutzer erhält damit scheinbar keine weiteren spezifischen Rechte (was in gesicherten Systemen fatal wäre). Informationen zu diesem Privileg finden Sie, wenn Sie bei *msdn.microsoft.com* nach »Act as part of the operating system« suchen (die URL der entsprechenden Seite sieht so aus, als würde sie nicht allzu lange bestehen bleiben ...).

Zum Testen der Methoden erzeugen Sie z.B. eine Textdatei (*demo.txt*) und entfernen Sie die Leserechte auf dieser Datei für einen speziellen Benutzer. Loggen Sie sich dann als dieser Benutzer ein und führen Sie das folgende Programm aus, wobei Sie allerdings beim Aufruf

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

von `ImpersonateUser` einen auf Ihrem System gültigen Kontonamen und dessen Passwort eintragen müssen. Die Textdatei *demo.txt* wird beim Einlesen in dem Ordner erwartet, in dem das Programm gespeichert ist.

Das Programm erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Windows.Forms` und `System.Security.Principal`.

```
// Variablen für die Personifizierung
string userName = "Ford";
string password = "Handtuch";

try
{
    // Den Namen des aktuellen Benutzers ausgeben
    WindowsIdentity identity = WindowsIdentity.GetCurrent();
    Console.WriteLine("Aktueller Benutzer: {0}", identity.Name);

    // Aktion ausführen, zu der das interaktive Konto keine Rechte
    // besitzt
    StreamReader sr = new StreamReader(
        Path.Combine(Application.StartupPath, "demo.txt"),
        Encoding.Default);
    string line = sr.ReadLine();
    Console.WriteLine(line);
    sr.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

try
{
    // Programm personifizieren
    Logon.ImpersonateUser(null, userName, password);

    // Den Namen des nun aktuellen Benutzers ausgeben
    WindowsIdentity identity = WindowsIdentity.GetCurrent();
    Console.WriteLine("Aktueller Benutzer: {0}", identity.Name);

    // Aktion ausführen, zu der das neue Konto nun (hoffentlich) Rechte
    // besitzt
    StreamReader sr = new StreamReader(
        Path.Combine(Application.StartupPath, "demo.txt"),
        Encoding.Default);
    string line = sr.ReadLine();
    Console.WriteLine(line);
    sr.Close();
}
```

Listing 392: Beispielhafte Anwendung der Klasse `Logon`

```
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

// Personifizierung rückgängig machen
Logon.UndoImpersonation();
```

Listing 392: Beispielhafte Anwendung der Klasse Logon (Forts.)

Die Methoden der Klasse `Logon` funktionierten in meinen Tests einwandfrei. Ein kleines Problem bleibt aber noch bestehen: `LogonUser` erzeugte bei meinen Versuchen unter Windows XP dummerweise auch dann ein Token, wenn ein nicht vorhandener Benutzername angegeben wurde. Dieses Token repräsentiert dann den Benutzer *Gast* und besitzt natürlich die sehr eingeschränkten Rechte dieses Benutzers.

Problematisch wird dies, wenn eine Anwendung `ImpersonateUser` einen nicht (mehr) existierenden Benutzernamen übergibt und der Zugriff auf Ressourcen dann zu Ausnahmen führt. Die Ursache dieses Fehlers zu finden kann in dem Fall u. U. recht problematisch sein.

Unter Windows 2000 trat dieses Problem übrigens nicht auf.

Um zu vermeiden, dass ein Programm fehlerhaft ausgeführt wird, wenn das angegebene Benutzerkonto nicht existiert, könnte `ImpersonateUser` das erzeugte `WindowsPrincipal`-Objekt daraufhin überprüfen, ob es die Benutzerinformationen speichert, die übergeben wurden. Im negativen Fall könnte die Methode dann eine Ausnahme generieren.

Dabei entstehen aber zu viele Unwägbarkeiten. Die Eigenschaft `Name` eines `WindowsPrincipal`-Objekts verwaltet den Namen in der Form »*Domäne\Benutzername*«. Übergeben werden können (zurzeit) aber ein normaler Benutzername mit oder ohne Domäne und ein Benutzername im UPN-Format (*Benutzer@DNS-Domäne*). Der Vergleich der übergebenen Benutzerinformationen mit den Informationen im Objekt wäre sehr schwierig und zudem recht unsicher.

Die andere Idee, eine Überprüfung auf den Benutzer *Gast*, ist leider auch nicht allzu sicher. Die Eigenschaft `IsGuest` eines `WindowsPrincipal`-Objekts gibt bedauerlicherweise bei den automatisch erzeugten *Gast*-Token `false` zurück. Und der Vergleich auf den Namen *Gast* würde das Programm auf Deutschland einschränken, da der *Gast*-Benutzer in anderen Ländern natürlich anders benannt ist.

Eine Lösung des Problems wäre, vor dem Erzeugen des Tokens zu überprüfen, ob der angegebene Benutzer überhaupt existiert. Dazu können Sie die Methode `UserExists` aus dem Rezept 198 verwenden. Und dann müsste eigentlich alles funktionieren 😊.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

212 Daten symmetrisch ver- und entschlüsseln

Häufig müssen geheime Daten in Konfigurationsdateien oder Datenbanken gespeichert, über das Internet versendet werden oder Ähnliches. Diese Daten sollten Sie normalerweise nicht unverschlüsselt speichern bzw. übertragen.

Das .NET Framework bietet mit den Klassen des Namensraums `System.Security.Cryptography` eine Vielzahl an Möglichkeiten, Daten zu verschlüsseln. Neben den wichtigen Verschlüsselungs-Algorithmen für eine symmetrische und asymmetrische Verschlüsselung werden auch Hashing-Verfahren, digitale Zertifikate und XML-Signaturen unterstützt.

Hinweis

Verschlüsselungs-Algorithmen setzen im Allgemeinen Schlüssel ein, deren Werte in der Verschlüsselungslogik berücksichtigt werden. Ein einfacher Algorithmus würde z.B. den Zeichencode jedes Zeichens eines zu verschlüsselnden Strings mit dem Zeichencode des korrespondierenden Schlüssel-Strings addieren und das Ergebnis speichern. Neben dem Verschlüsselungsalgorithmus bestimmt auch die Länge des Schlüssels die Qualität des Ergebnisses.

Symmetrische Verschlüsselungsverfahren setzen ein und denselben Schlüssel mit einer meist festgelegten Länge für das Ver- und das Entschlüsseln ein. Das Problem dabei ist die Weitergabe des Schlüssels: Gelangt der Schlüssel in falsche Hände, können die Daten problemlos entschlüsselt werden (sofern das Verfahren bekannt ist). Der Schlüssel muss also geheim gehalten werden (und wird deshalb als geheimer Schlüssel bezeichnet). Beispiele für symmetrische Verfahren sind RC2 (64 Bit Schlüsselgröße), DES (64 Bit), TripleDES (192 Bit), AES (256 Bit), IDEA (128 Bit) und CAST (128 oder 256 Bit).

Asymmetrische Verfahren wie RSA und DSA setzen unterschiedliche Schlüssel variabler Länge für das Ver- und das Entschlüsseln der Daten ein. Die Schlüssel stehen in einer mathematischen Beziehung, die meist auf Primzahlen basiert, und bilden ein Schlüssel-Paar. Einer der Schlüssel wird geheim gehalten und daher als privater Schlüssel bezeichnet. Der andere Schlüssel wird als öffentlicher Schlüssel bezeichnet und kann problemlos öffentlich, z.B. über E-Mail, versendet werden. Diese Verfahren werden deshalb auch als Public-Key-Verfahren bezeichnet. Ein Sender, der den öffentlichen Schlüssel des Empfängers kennt, kann Daten mit diesem Schlüssel verschlüsseln und versenden. Die Daten können nur mit dem anderen Schlüssel des Paares entschlüsselt werden. Da dieser privat ist, kann nur der Empfänger die Daten entschlüsseln.

Asymmetrische Verschlüsselungsverfahren sind zwar in Bezug auf die Schlüsselweitergabe sicherer als symmetrische, bieten aber nicht so viel Sicherheit gegen das ungewollte Entschlüsseln der Daten wie symmetrische Verfahren und sind zudem wesentlich langsamer. Daher wird in der Praxis häufig eine Kombination eingesetzt: Die eigentlichen Daten werden symmetrisch über einen zufällig erzeugten Schlüssel verschlüsselt. Dieser Schlüssel wird mit dem öffentlichen Schlüssel des Empfängers asymmetrisch verschlüsselt und mit den Daten versendet. Der Empfänger kann den symmetrischen Schlüssel über seinen privaten Schlüssel und mit diesem die Daten entschlüsseln. Solche so genannten hybriden Verfahren werden z.B. von PGP und GnuPG eingesetzt. Zusätzlich wird häufig noch ein Hashcode (siehe Rezept Daten mit Hashing-Verfahren verschlüsseln) über die zu sendenden Daten berechnet, der der Nachricht mitgesendet wird. Da Hashcodes nicht entschlüsselbar sind, kann der Empfänger über dasselbe Hashing-Verfahren über die entschlüsselten Daten einen neuen Hashcode berechnen und diesen mit dem gesendeten vergleichen. Sind die Hashcodes identisch, kann der Empfänger sicher sein, dass die Daten auf ihrem Weg nicht verändert wurden.

Bei symmetrischen Verschlüsselungsverfahren werden verschiedene Chiffrier-Modi (Cipher modes) unterschieden. Block-Chiffrierungen wie CBC (Cipher Block Chaining), CFB (Cipher Feedback) und ECB (Electronic Codebook) verschlüsseln einen festen Block von Daten in einen Block gleicher Länge. Stream-Chiffrierungen verschlüsseln einzelne Bit-Gruppen, typischerweise, indem diese mit dem Schlüssel bitweise mit XOR verknüpft werden.

Die Blockgröße von Block-Chiffrierungen können Sie unter in den .NET-Klassen einstellen. Die Arbeitsweise der von .NET unterstützten Modi werden sehr gut in der Dokumentation der `CipherMode`-Aufzählung erläutert. Bei allen Block-Chiffrierungen werden die Daten blockweise über den Schlüssel neu berechnet und im Ergebnis abgelegt. Der beliebte CBC-Modus nutzt für die Verschlüsselung zusätzlich den zuvor verschlüsselten Block. Weil für den ersten Block kein verschlüsselter Block zur Verfügung steht, arbeitet CBC mit einem so genannten Initialisierungsvektor, das ist ein Block beliebiger Daten, der von außen definiert werden kann. Wenn Sie Daten im (übrigens unter .NET voreingestellten) CBC-Modus ver- und entschlüsseln, müssen Sie also neben dem Schlüssel auch den Initialisierungsvektor angeben. Die Länge des Initialisierungsvektors muss der Blockgröße entsprechen.

Da der letzte Block meist nicht der definierten Blockgröße entspricht, wird dieser häufig mit Bytes aufgefüllt. Bei diesem so genannten Padding werden ebenfalls verschiedene Modi eingesetzt. PKCS7 speichert dazu für jedes Byte den Wert der Anzahl der insgesamt hinzuzufügenden Bytes, der andere Modus Zeros speichert einfach 0-Bytes. PKCS7 ist unter .NET die Voreinstellung.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

In diesem Rezept zeige ich, wie Sie eine symmetrische Verschlüsselung von Daten erreichen. Im Rezept 213 erfahren Sie, wie Sie aus Daten einen Hashcode erzeugen (der nicht mehr entschlüsselbar ist). Die asymmetrische Verschlüsselung behandelt dieses Buch aufgrund der recht hohen Komplexität nicht. Ein gutes Beispiel finden Sie an der Adresse www.codeproject.com/csharp/Cryptography.asp.

Für eine symmetrische Verschlüsselung stellt Ihnen der Namensraum `System.Security.Cryptography` einige abstrakte Basisklassen und einige davon abgeleitete konkrete Implementierungen zur Verfügung:

- ▶ `SymmetricAlgorithm`: Basisklasse für alle symmetrischen Verschlüsselungsverfahren,
- ▶ `DES`: abstrakte Klasse für das DES-Verfahren (Data Encryption Standard),
- ▶ `DESCryptoServiceProvider`: konkrete Klasse für DES,
- ▶ `TripleDES`: abstrakte Klasse für den TripleDES-Algorithmus,
- ▶ `TripleDESCryptoServiceProvider`: konkrete Klasse für TripleDES,
- ▶ `RC2`: abstrakte Klasse für den RC2-Algorithmus,
- ▶ `RC2CryptoServiceProvider`: konkrete Klasse für RC2,
- ▶ `Rijndael`: abstrakte Klasse für das Rijndael-Verfahren,
- ▶ `RijndaelManaged`: konkrete Klasse für Rijndael.

Die abstrakten Klassen können Sie für eigene Implementierungen der einzelnen Algorithmen verwenden (wenn Sie viel Zeit haben ...). Diese Klassen bieten keine Implementierung der Ver- oder Entschlüsselung, sondern definieren lediglich die Schnittstelle der jeweiligen konkreten Klasse. Mit diesen Klassen wollte Microsoft wohl eine Art Polymorphismus erreichen, der dann möglich ist, wenn andere Hersteller von Kryptographie-Klassen ihre Klassen von den abstrakten Klassen ableiten.

Hinweis

Über die Adresse www.gotdotnet.com/team/clr/samples/eula_clr_cryptosrc.aspx können Sie den Quellcode der abstrakten Kryptographie-Klassen des .NET Frameworks downloaden, nachdem Sie die Endanwender-Lizenz-Vereinbarung dafür bestätigt haben.

Die Klassen mit der Endung *Managed* sind komplett in verwaltetem Code implementiert. Klassen mit der Endung *ServiceProvider* sind allerdings lediglich Wrapper für Klassen des nativen Windows-Kryptographie-API (Crypto API).

Die konkreten Klassen implementieren die einzelnen Verschlüsselungs-Algorithmen. In ihrem Konstruktor erzeugen diese Klassen kryptografisch starke Werte für den Schlüssel und den Initialisierungsvektor, die dann für das Ver- und das Entschlüsseln verwendet werden können. Sie können diese aber auch mit eigenen Werten neu definieren.

Um Daten symmetrisch zu verschlüsseln, müssen Sie sich nur noch entscheiden, welchen Algorithmus Sie verwenden. Ich habe mich für dieses Rezept entschieden, Ihnen die Wahl

des Algorithmus zu überlassen und eine Klasse zu entwickeln, die Daten nach allen genannten Algorithmen ver- und entschlüsseln kann. Dieses Vorhaben wurde enorm dadurch erleichtert, dass alle Klassen für symmetrische Verschlüsselungen von der Basisklasse `SymmetricAlgorithm` abgeleitet sind und keine Klasse zu den geerbten Elementen wesentliche, spezifische Elemente hinzufügt. Die Arbeit an dieser Klasse war übrigens nicht allzu einfach und hat gut einen Tag gedauert, weil ich (wie in den meisten anderen Rezepten dieses Buchs auch) eigene Ideen verwirklichen wollte. Dafür ist die Klasse aber auch sehr flexibel einsetzbar 😊.

Um diese Klasse kompilieren zu können, müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Security.Cryptography` importieren.

Zur Auswahl des Algorithmus habe ich zunächst eine Aufzählung `SymmetricEncryptAlgorithm` deklariert, die Konstanten für die unterstützten Algorithmen zur Verfügung stellt:

```
public enum SymmetricEncryptAlgorithm
{
    DES,
    TripleDES,
    RC2,
    Rijndael
}
```

Listing 393: Aufzählung für den Verschlüsselungsalgorithmus

Instanzen der Klasse `SymmetricEncryptor` übernehmen die Ver- und Entschlüsselung. Die Klasse enthält zunächst eine private Eigenschaft vom Typ `SymmetricAlgorithm`, die die Instanz des Verschlüsselungsobjekts verwaltet, und eine Eigenschaft, die den Namen des aktuell gewählten Algorithmus speichert (und die in Fehlermeldungen verwendet wird). Dem Konstruktor wird am ersten Argument der zu verwendende Algorithmus übergeben. Der Konstruktor erzeugt eine dazu passende Instanz einer der verfügbaren Klassen für symmetrische Verschlüsselungsverfahren und speichert die Referenz in der Eigenschaft `SymmetricEncryptor`. Außerdem wird der Name des Verschlüsselungsverfahrens in der dafür vorgesehenen Eigenschaft abgelegt:

```
public class SymmetricEncryptor
{
    /* Eigenschaften zur Speicherung der Verschlüsselungs-Instanz und des
     * gewählten Algorithmus */
    private SymmetricAlgorithm encryptor;
    private string algorithmName;

    /* Konstruktor */
    public SymmetricEncryptor(SymmetricEncryptAlgorithm algorithm)
    {
        switch (algorithm)
```

Listing 394: Eigenschaften und der Konstruktor der Klasse `SymmetricEncryptor`

```

    {
        case SymmetricEncryptAlgorithm.DES:
            this.encryptor = new DESCryptoServiceProvider();
            this.algorithmName = "DES";
            break;
        case SymmetricEncryptAlgorithm.TripplEDE:
            this.encryptor = new TripleDESCryptoServiceProvider();
            this.algorithmName = "TripleDES";
            break;
        case SymmetricEncryptAlgorithm.RC2:
            this.encryptor = new RC2CryptoServiceProvider();
            this.algorithmName = "RC2";
            break;
        case SymmetricEncryptAlgorithm.Rijndael:
            this.encryptor = new RijndaelManaged();
            this.algorithmName = "Rijndael";
            break;
    }
}

```

Listing 394: Eigenschaften und der Konstruktor der Klasse SymmetricEncryptor (Forts.)

Nun folgen einige Eigenschaften, über die Sie den zu verwendenden Chiffrier-Modus, die Block-Größe und das Padding lesen und definieren können (wenn Sie wollen ...):

```

/* Eigenschaft für den Chiffrier-Modus */
public CipherMode CipherMode
{
    get {return this.encryptor.Mode;}
    set {this.encryptor.Mode = value;}
}

/* Eigenschaft für die Blockgröße */
public int BlockSize
{
    get {return this.encryptor.BlockSize;}
    set {this.encryptor.BlockSize = value;}
}

/* Eigenschaft für das Padding */
public PaddingMode PaddingMode
{
    get {return this.encryptor.Padding;}
    set {this.encryptor.Padding = value;}
}

```

Listing 395: Eigenschaften zur Definition des Chiffrier-Modus, der Blockgröße und des Padding

Der Konstruktor der verwendeten Verschlüsselungsklasse erzeugt automatisch einen Zufalls-Schlüssel. Damit Sie diesen lesen, aber auch nach der Erzeugung der Instanz mit einem eigenen Schlüssel überschreiben können, habe ich die Eigenschaft `Key` implementiert, die auf die

gleichnamige Eigenschaft im Verschlüsselungs-Objekt zugreift. Dabei musste ich das Problem lösen, dass der Schlüssel intern als Byte-Array verwaltet wird, ich aber einen String als Schlüssel verwenden wollte.

Das Problem dabei ist (wie sehr häufig bei Strings) die Codierung. Strings werden in .NET-Programmen in der 16-Bit-Unicode-Codierung gespeichert. Würde `Key` den übergebenen String über `System.Text.Encoding.Unicode.GetBytes` bzw. `GetString` decodieren bzw. codieren, würde das hohe Byte jedes Unicode-Zeichens mit in das Byte-Array übertragen werden. Da die meisten Strings nur Zeichen im Bereich bis 0x00FF (255) enthalten, wäre jedes zweite Byte in diesem Falle 0. Der Schlüssel wäre also eigentlich nur die Hälfte wert. Deshalb habe ich mich entschlossen, als Codierung die zum 8-Bit-Teil des Unicode-Zeichensatzes compatible Codierung ISO-8859-1 zu verwenden und damit nur das niedrige Byte jedes Zeichens als Schlüssel zu verwenden. So können Sie alle 8-Bit-Zeichen des Unicode-Zeichensatzes für den Schlüssel verwenden und besitzen damit auch alle Möglichkeiten, die der Schlüssel zur Verfügung stellt (eben 8 Bit pro Byte J).

Um sicherzustellen, dass nicht aus Versehen Schlüssel mit Zeichen übergeben werden, die größer sind als 0x00FF, prüft `Key` beim Schreiben jedes Zeichen daraufhin ab und generiert im Fehlerfall eine Ausnahme vom Typ `CryptographicException`.

Da die verschiedenen Verschlüsselungsverfahren mit unterschiedlichen Schlüsselgrößen arbeiten, überprüft `Key` beim Setzen des Schlüssels zudem über die `ValidKeySize`-Methode des Verschlüsselungs-Objekts, ob die Schlüssellänge in Ordnung ist, und generiert im Fehlerfall eine Ausnahme. Ich habe mich hierbei nicht auf die Ausnahme verlassen, die die `Key`-Eigenschaft des Verschlüsselungs-Objekts wirft, wenn der Schlüssel ungültig ist, weil diese nur die allgemeine Fehlermeldung ausgibt, dass der Schlüssel ungültig ist. Meine Ausnahme liefert zudem die Information, welche Schlüsselgrößen erlaubt sind. Dabei musste ich eine Eigenart beachten: Die dazu auszulesende Auflistung `LegalKeySizes` liefert bei den .NET-Verschlüsselungsklassen nur ein Element. Die Eigenschaften `MinSize` und `MaxSize` speichern für diese Algorithmen aber nicht einen gültigen Bereich (wie die Namen vermuten lassen), sondern in einigen Fällen gleiche und in anderen Fällen zwei verschiedene mögliche Werte (also z. B. 64 oder 128 Bit). Das ist leider etwas unlogisch, Microsoft hätte in den Fällen, in denen mehrere fest definierte Schlüsselgrößen unterstützt werden, auch mehrere Elemente speichern sollen, bei denen `MinSize` und `MaxSize` den gleichen Wert ausweisen. Die Ausnahmemeldung trennt die Werte deshalb lediglich mit einem Komma.

```
public string Key
{
    get
    {
        return Encoding.GetEncoding("ISO-8859-1").GetString(
            this.encryptor.Key);
    }

    set
    {

```

Listing 396: Eigenschaft zum Lesen und Setzen des Schlüssels

```

// Den übergebenen Schlüssel überprüfen
if (this.encryptor.ValidKeySize(value.Length * 8) == false)
{
    // Ungültiger Schlüssel: Ausnahme mit erweiterten Informationen
    // werfen
    string allowedKeySizes = null;
    for (int i = 0; i < this.encryptor.LegalKeySizes.Length; i++)
    {
        if (allowedKeySizes != null) allowedKeySizes += ", ";
        allowedKeySizes += this.encryptor.LegalKeySizes[i].MinSize +
            ", " + this.encryptor.LegalKeySizes[i].MaxSize;
    }
    throw new CryptographicException("Der übergebene Schlüssel " +
        "ist mit " + (value.Length * 8) + " Bit für den " +
        this.algorithmName + "-Algorithmus ungültig. Erlaubt " +
        "sind die folgenden Größen: " + allowedKeySizes + ".");
}

// Auf Unicode-Zeichen größer 0x00FF überprüfen
for (int i = 0; i < value.Length; i++)
{
    if ((int)value[i] > 255)
        throw new CryptographicException("Der übergebene " +
            "Schlüssel enthält mindestens ein Unicode-Zeichen, " +
            "das größer ist als 0x00FF (255): " + value[i] + " (" +
            (int)value[i] + "). Unterstützt werden lediglich " +
            "8-Bit-Unicode-Zeichen");
}

// Den Schlüssel setzen
this.encryptor.Key = Encoding.GetEncoding(
    "ISO-8859-1").GetBytes(value);
}
}

```

Listing 396: Eigenschaft zum Lesen und Setzen des Schlüssels (Forts.)

Wenn Sie Daten im CBC-Modus verschlüsseln (was die Voreinstellung ist!), benötigt das Verschlüsselungsobjekt einen Initialisierungsvektor aus Bytes, dessen Größe der aktuell eingestellten Blockgröße entspricht. Der Konstruktor der verwendeten Verschlüsselungsklassen erzeugt diesen automatisch. Ähnlich wie bei der `Key`-Eigenschaft müssen Sie diesen lesen und auch schreiben können. Die Eigenschaft `IV` erlaubt deswegen das Lesen und Schreiben des Initialisierungsvektors. Wie bei `Key` habe ich diese Eigenschaft so definiert, dass der 8-Bit-Bereich der Unicode-Zeichen verwendet wird. Der übergebene String wird deshalb auch wieder daraufhin überprüft, ob er Zeichen speichert, die größer sind als 0x00FF.

Da die Größe des Initialisierungsvektors der aktuellen Blockgröße entsprechen muss und im Fehlerfall eine Ausnahme erst beim Ver- oder Entschlüsseln generiert werden würde, überprüft `IV` die Größe des Initialisierungsvektors gegen die Eigenschaft `BlockSize` und wirft im negativen Fall eine aussagekräftige Ausnahme.

```
public string IV
{
    get
    {
        return Encoding.GetEncoding("ISO-8859-1").GetString(
            this.encryptor.IV);
    }
    set
    {
        // Den übergebenen Initialisierungsvektor überprüfen
        if ((value.Length * 8) != this.encryptor.BlockSize)
        {
            // Ungültiger Initialisierungsvektor: Ausnahme mit erweiterten
            // Informationen werfen
            throw new CryptographicException("Der übergebene " +
                "Initialisierungsvektor ist mit " + (value.Length * 8) +
                " Bit für den " + this.algorithmName + "-Algorithmus " +
                "ungültig. Die Länge des Initialisierungsvektors muss " +
                "der Blockgröße (aktuell " + this.encryptor.BlockSize +
                ") entsprechen");
        }

        // Auf Unicode-Zeichen größer 0x00FF überprüfen
        for (int i = 0; i < value.Length; i++)
        {
            if ((int)value[i] > 255)
            {
                throw new CryptographicException("Der übergebene " +
                    "Initialisierungsvektor enthält mindestens ein " +
                    "Unicode-Zeichen, das größer ist als 0x00FF (255): " +
                    value[i] + " (" + (int)value[i] + "). Unterstützt " +
                    "werden lediglich 8-Bit-Unicode-Zeichen");
            }
        }

        // Den Initialisierungsvektor setzen
        this.encryptor.IV = Encoding.GetEncoding(
            "ISO-8859-1").GetBytes(value);
    }
}
```

Listing 397: Eigenschaft zum Zugriff auf den Initialisierungsvektor

Als kleine Zugabe habe ich die Methoden zur Erzeugung eines neuen zufälligen Schlüssels und Initialisierungsvektors übernommen:

```
public void GenerateKey()
{
    this.encryptor.GenerateKey();
}
```

Listing 398: Methoden zur Erzeugung eines zufälligen Schlüssels und Initialisierungsvektors

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
public void GenerateIV()
{
    this.encryptor.GenerateIV();
}
```

Listing 398: Methoden zur Erzeugung eines zufälligen Schlüssels und Initialisierungsvektors (Forts.)

Nun folgt der interessante Teil: das Ver- und Entschlüsseln. Die erste Variante der Methode `Encrypt` verschlüsselt einen Stream in einen anderen Stream. `Encrypt` arbeitet mit Streams, weil die Verschlüsselungs-Objekte auch damit arbeiten und weil diese Methode dadurch enorm flexibel wird. So können Sie z.B. `FileStream`-Objekte verwenden, um Dateien zu verschlüsseln, oder Daten aus einem `MemoryStream`-Objekt in einen `NetworkStream` schreiben um diese über das Internet oder ein normales Netz zu versenden.

Zum Verschlüsseln erzeugt `Encrypt` eine Instanz der Klasse `CryptoStream`. Am ersten Argument wird der Stream übergeben, in den das `CryptoStream`-Objekt die verschlüsselten Daten schreibt. Am zweiten Argument übergibt `Encrypt` ein Objekt, das die Schnittstelle `ICryptoTransform` implementiert, über die die Verschlüsselung vorgenommen wird. Dieses Objekt wird über die Methode `CreateEncryptor` des aktuellen Verschlüsselungs-Objekts ermittelt. Am letzten Argument wird dann noch festgelegt, dass der `CryptoStream` in den Zielstream schreiben soll.

Danach werden die Daten des übergebenen Quell-Stream in den `Crypto-Stream` geschrieben, wobei das Schreiben blockweise erfolgt, damit der Arbeitsspeicher bei großen Streams nicht überlastet wird. Das `CryptoStream`-Objekt sorgt automatisch dafür, dass die geschriebenen Daten über das `ICryptoTransform`-Objekt verschlüsselt und in den Ziel-Stream geschrieben werden. Nach dem Schreiben wird über die `FlushFinalBlock`-Methode des `CryptoStreams` nur noch dafür gesorgt, dass die noch im Stream vorhandenen Restdaten verarbeitet werden.

Der `Crypto-Stream` wird übrigens explizit nicht geschlossen. Der Grund dafür ist, dass ein Aufruf der `Close`-Methode bewirkt, dass (nach eigenen empirischen Ermittlungen) auch der Ziel-Stream (der dem `Crypto-Stream` im Konstruktor übergeben wurde) geschlossen wird. `Encrypt` kann aber nicht voraussetzen, dass der Zielstream nicht mehr benötigt wird. Wird z.B. ein `MemoryStream` übergeben, ist es nach dem Verschlüsseln notwendig, den Stream auf die erste Position zu setzen und auszulesen.

```
public void Encrypt(Stream source, Stream dest)
{
    // CryptoStream zum Verschlüsseln erzeugen. Als Transformations-Objekt
    // wird das Verschlüssel-Objekt der aktuellen SymmetricAlgorithm-Instanz
    // übergeben
    CryptoStream cryptoStream = new CryptoStream(dest,
        this.encryptor.CreateEncryptor(), CryptoStreamMode.Write);

    // Die Rohdaten blockweise in den CryptoStream schreiben (der diese
```

Listing 399: Methode zum Verschlüsseln eines Streams

```

// verschlüsselt in den Ziel-Stream schreibt)
byte[] buffer = new byte[1024];
int bytesRead = 0;
do
{
    bytesRead = source.Read(buffer, 0, 1024);
    cryptoStream.Write(buffer, 0, bytesRead);
} while (bytesRead > 0);

// Den Zielstream aktualisieren und den Puffer löschen
cryptoStream.FlushFinalBlock();

// Der CryptoStream darf hier nicht geschlossen werden, da dieser den
// Ziel-Stream ansonsten ebenfalls schließt
}

```

Listing 399: Methode zum Verschlüsseln eines Streams (Forts.)

Die erste Variante der Methode `Decrypt` entschlüsselt einen Stream in einen anderen Stream. `Decrypt` erzeugt dazu wieder eine `CryptoStream`-Instanz, der aber am ersten Argument nun der Quell-Stream übergeben wird. Am zweiten Argument wird das `ICryptoTransform`-Objekt übergeben, das die `CreateDecryptor`-Methode des aktuellen Verschlüsselungs-Objekts zurückgibt. Das dritte Argument bestimmt mit `CryptoStreamMode.Read`, dass die Daten des am ersten Argument übergebenen Streams gelesen werden sollen.

Dann werden die Daten blockweise aus dem `Crypto-Stream` in einen `Byte-Puffer` gelesen (wieder um den Arbeitsspeicher nicht zu sehr zu belasten) und in den `Ergebnis-Stream` geschrieben. Beim Lesen der Daten werden diese automatisch über das `ICryptoTransform`-Objekt entschlüsselt.

Der `Crypto-Stream` darf wieder nicht geschlossen werden, da ansonsten auch der Quell-Stream geschlossen wird.

```

public void Decrypt(Stream source, Stream dest)
{
    // CryptoStream zum Entschlüsseln erzeugen. Als Transformations-
    // Objekt wird das Entschlüssel-Objekt der aktuellen
    // SymmetricAlgorithm-Instanz übergeben
    CryptoStream cryptoStream = new CryptoStream(source,
        this.encryptor.CreateDecryptor(), CryptoStreamMode.Read);

    // Daten blockweise einlesen
    int bytesRead = 0;
    byte[] buffer = new byte[1024];
    do
    {
        bytesRead = cryptoStream.Read(buffer, 0, 1024);
        dest.Write(buffer, 0, bytesRead);
    }
}

```

Listing 400: Methode zum Entschlüsseln eines Streams

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

    } while (bytesRead > 0);

    // Der CryptoStream darf hier nicht geschlossen werden, da
    // dieser den Quell-Stream ansonsten auch schließt
}

```

Listing 400: Methode zum Entschlüsseln eines Streams (Forts.)

Eigentlich wäre die Klasse nun komplett. Ich habe aber noch zwei Methoden zum Ver- und Entschlüsseln von Strings hinzugefügt, da die dazu notwendige Umwandlung in einen `MemoryStream` recht aufwändig ist. Beide Methoden erzeugen `MemoryStream`-Objekte und geben diese einfach an die Stream-Variante der entsprechenden Methode weiter.

```

/* Methode zum Verschlüsseln von Strings */
public string Encrypt(string source)
{
    // MemoryStreams für die Daten erzeugen
    MemoryStream sourceStream =
        new MemoryStream(Encoding.Unicode.GetBytes(source));
    MemoryStream destStream = new MemoryStream();

    // Daten verschlüsseln
    Encrypt(sourceStream, destStream);

    // Ergebnis auslesen
    destStream.Seek(0, SeekOrigin.Begin);
    byte[] encryptedBytes = destStream.ToArray();

    // Streams schließen
    sourceStream.Close();
    destStream.Close();

    // String zurückgeben
    return Encoding.Unicode.GetString(encryptedBytes);
}

/* Methode zum Entschlüsseln von Strings */
public string Decrypt(string source)
{
    // MemoryStreams für die Daten erzeugen
    MemoryStream sourceStream =
        new MemoryStream(Encoding.Unicode.GetBytes(source));
    MemoryStream destStream = new MemoryStream();

    // Daten entschlüsseln
    Decrypt(sourceStream, destStream);

    // Ergebnis auslesen
    destStream.Seek(0, SeekOrigin.Begin);
}

```

Listing 401: Methoden zum Ver- und Entschlüsseln von Strings

```

byte[] encryptedBytes = destStream.ToArray();

// Streams schließen
sourceStream.Close();
destStream.Close();

// String zurückgeben
return Encoding.Unicode.GetString(encryptedBytes);
}
}

```

Listing 401: Methoden zum Ver- und Entschlüsseln von Strings (Forts.)

Zum Ver- und Entschlüsseln erzeugen Sie dann eine Instanz der Klasse `SymmetricEncryptor`. Wenn Sie den Schlüssel und den Initialisierungsvektor nicht selbst definieren wollen, können Sie beim Verschlüsseln die zufällig erzeugten Werte aus den Eigenschaften `Key` und `IV` auslesen. Diese Strings sollten Sie auf jeden Fall irgendwo speichern, da Sie diese Werte beim Entschlüsseln wieder benötigen.

Listing 402 zeigt die Vorgehensweise am Beispiel. Das Beispiel ver- und entschlüsselt eine Datei nach dem Rijndael-Algorithmus und setzt dazu eigene Schlüssel- und Initialisierungsvektor-Werte ein. Auf eine Ausnahmebehandlung habe ich im Beispiel aus Übersichtsgründen verzichtet.

```

// SymmetricEncryptor-Instanz erzeugen und den eigenen Schlüssel und
// Initialisierungsvektor übergeben
SymmetricEncryptor se =
    new SymmetricEncryptor(SymmetricEncryptAlgorithm.Rijndael);
se.Key = "dfd0B3q38dsäfBq3#sfjbxya<xsd-:,?";
se.IV = "460?-B,-7,kerkh-";

// Eine Datei verschlüsseln
FileStream sourceStream = new FileStream("c:\Hitchhiker.txt", FileMode.Open,
    FileAccess.Read);
FileStream destStream = new FileStream("c:\Hitchhiker_Encrypted.txt",
    FileMode.Create, FileAccess.Write);
se.Encrypt(sourceStream, destStream);
sourceStream.Close();
destStream.Close();

// Die Datei wieder entschlüsseln
sourceStream = new FileStream("c:\Hitchhiker_Encrypted.txt", FileMode.Open,
    FileAccess.Read);
destStream = new FileStream("c:\Hitchhiker.txt", FileMode.Create,
    FileAccess.Write);
se.Decrypt(sourceStream, destStream);
sourceStream.Close();
destStream.Close();

```

Listing 402: Beispiel für die Anwendung der SymmetricEncryptor-Klasse

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

213 Daten mit Hashing-Verfahren verschlüsseln

Daten müssen in einigen Fällen so verschlüsselt werden, dass diese nicht mehr entschlüsselt werden können und in einer verkürzten Form dargestellt werden. Dazu werden so genannte Hashcodes verwendet, die über öffentlich bekannte Hashing-Verfahren errechnet werden. Ein Hashcode repräsentiert die Daten in einer verkleinerten Form mit einer in der Regel festgelegten Länge (z.B. 128 oder 256 Bit). Aus einem Hashcode können unter keinen Umständen die eigentlichen Daten zurückgerechnet werden. Wird auf dieselben Daten derselbe Hashing-Algorithmus mit derselben Hashcode-Länge angewendet, resultiert aber immer derselbe Hashcode. Die Wahrscheinlichkeit, dass unterschiedliche Daten zu demselben Hashcode führen, ist äußerst gering.

Eine Anwendung für Hashcodes ist die Speicherung von Passwörtern, die von Anwendern zur Authentifizierung eingegeben werden sollen, in einer Konfigurationsdatei, Datenbank o.Ä. Gespeichert werden nicht die Passwörter, sondern deren Hashcodes. Im Login-Formular wird zur Authentifizierung das vom Anwender eingegebene Passwort in den korrespondierenden Hashcode umgerechnet und mit dem gespeicherten verglichen. Sind beide Hashcodes identisch, ist das Passwort korrekt. Natürlich können Sie für einen solchen Zweck auch einen Verschlüsselungs-Algorithmus verwenden. Hashcodes sind aber einfacher zu errechnen und bieten die Sicherheit, dass die Daten nicht entschlüsselt werden können.

Eine andere Anwendung für Hashcodes sind komplexe Verschlüsselungsverfahren, bei denen aus den zu verschlüsselnden Daten ein Hashcode berechnet wird, der den verschlüsselten Daten beigelegt wird. Der Empfänger kann aus den entschlüsselten Daten wieder einen Hashcode berechnen und diesen mit dem mitgelieferten vergleichen. Sind beide Hashcodes gleich, wurden die Daten mit einer recht hohen Wahrscheinlichkeit nicht verändert.

Hashcodes erzeugen Sie über spezielle Hashing-Klassen, die das .NET Framework im Namensraum `System.Security.Cryptography` zur Verfügung stellt. Diese Klassen implementieren die verschiedenen von .NET unterstützten Algorithmen und ermöglichen die Erzeugung von Hashcodes aus Byte-Arrays oder Streams.

Wie bereits bei der Verschlüsselung existieren einige abstrakte Basisklassen wie z.B. `SHA1` oder `MD5`, die keine Implementierung beinhalten. Die Namen der davon abgeleiteten Klassen enden teilweise auf `Managed`, was darauf hinweist, dass die Implementierung in verwaltetem Code existiert. Namen mit der Endung `ServiceProvider` (z.B. bei der Klasse `SHA1CryptoServiceProvider`) weisen darauf hin, dass die Klassen lediglich Wrapper für entsprechende Klassen des Microsoft-Crypto-API sind. Da aber für diese Algorithmen oft auch verwaltete Varianten existieren, beschreibe ich nur diese. Die folgenden Klassen implementieren die einzelnen Algorithmen:

- ▶ `SHA1Managed`: SHA1 (Secure Hash Algorithm 1) mit 160 Bit Länge,
- ▶ `SHA256Managed`: SHA1 mit 256 Bit Länge,
- ▶ `SHA384Managed`: SHA1 mit 384 Bit Länge,
- ▶ `SHA512Managed`: SHA1 mit 512 Bit Länge,
- ▶ `MD5CryptoServiceProvider`: MD5 (Message Digest 5) mit 128 Bit Länge,

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

- HMACSHA1: SHA1 mit 160 Bit und Schlüssel,
- MACTripledES: MACTripledES mit 64 Bit Länge und Schlüssel.

Die verschiedenen Implementierungen des SHA1-Algorithmus verwenden eine unterschiedliche Hashcode-Größe. Ein größerer Hashcode, dessen Berechnung ein wenig mehr Zeit kostet, bringt eine größere Sicherheit gegenüber Doppelgängern. Die Größe des Hashcodes können Sie übrigens aus der Eigenschaft `HashSize` auslesen.

Die Verwendung dieser Klassen, die übrigens alle mehr oder weniger direkt von der Basisklasse `HashAlgorithm` abgeleitet sind, ist sehr einfach: Sie erzeugen eine Instanz und rufen zur Erzeugung des Hashcode die Methode `ComputeHash` auf. Dieser Methode können Sie ein `Byte-Array` oder einen `Stream` übergeben. Sie erhalten dann ein `Byte-Array` mit den Daten des Hashcode zurück.

Bei den Algorithmen, die mit einem Schlüssel arbeiten, können Sie zudem vor der Erzeugung des Hashcode in der Eigenschaft `Key` den Schlüssel definieren. Dieser Schlüssel wird bei jeder Erzeugung einer Instanz dieser Klassen per Zufall neu erzeugt. Für den Vergleich zweier Hashcodes müssen Sie den Schlüssel also zwischenspeichern und beim zweiten Ermitteln des Hashcodes setzen. Da der Schlüssel als `Byte-Array` verwaltet wird, ist dies leider mit etwas Arbeit verbunden.

Eine recht direkte Möglichkeit, einen Hashcode zu erzeugen, ist die Erzeugung einer Instanz einer Hash-Algorithmus-Klasse über `HashAlgorithm.Create`. Dieser Methode übergeben Sie einen String, der den zu verwendenden Algorithmus bezeichnet (z.B. »SHA1«, »SHA256« oder »MD5«. Über die zurückerhaltene `HashAlgorithm`-Referenz können Sie deren `ComputeHash`-Methode aufrufen.

Das folgende Beispiel erzeugt einen Hashcode für einen String:

```
string source = "Das ist ein Teststring zur Ermittlung eines Hashcodes";  
byte[] buffer = Encoding.Unicode.GetBytes(source);  
string hashCode = Encoding.Unicode.GetString(((HashAlgorithm)  
    CryptoConfig.CreateFromName("SHA1")).ComputeHash(buffer));
```

Listing 403: Direktes Erzeugen eines Hashcodes

Um die Arbeit mit Hashcodes zu erleichtern und um die Übergabe des Schlüssels für die Algorithmen zu erleichtern, die einen solchen verwenden, habe ich die Klasse `Hasher` entwickelt. Die Grundidee dieser Klasse ist dieselbe wie die der Klasse `SymmetricEncryptor` aus dem Rezept 212. `Hasher` ermöglicht die einfache Verwendung aller unterstützten Hashing-Algorithmen. Da Hashcodes häufig als String gespeichert werden, geben die Varianten der Methode `ComputeHash`, die den Hashcode erzeugen, einfach einen String in der zu Unicode kompatiblen 8-Bit-Codierung ISO-8859-1 zurück.

Zum Kompilieren der `Hasher`-Klasse müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Security.Cryptography` importieren.

Zunächst habe ich für die Definition des zu verwendenden Algorithmus die Aufzählung `HashAlgorithmEnum` implementiert (deren Name mit `Enum` endet, da bereits eine Klasse `HashAlgorithm` existiert):

```
public enum HashAlgorithmEnum
{
    SHA160,
    SHA256,
    SHA384,
    SHA512,
    MD5,
    HMACSHA1,
    MACTripleDES
}
```

Listing 404: Aufzählung für die unterstützten Hashing-Algorithmen

Im Konstruktor der `Hasher`-Klasse wird der zu verwendende Algorithmus übergeben und in der privaten Variablen `hashObject` gespeichert:

```
public class Hasher
{
    /* Eigenschaft zur Speicherung des Hash-Objekts */
    private HashAlgorithm hashObject;

    /* Konstruktor */
    public Hasher(HashAlgorithmEnum algorithm)
    {
        switch (algorithm)
        {
            case HashAlgorithmEnum.SHA160:
                this.hashObject = new SHA1Managed();
                break;
            case HashAlgorithmEnum.SHA256:
                this.hashObject = new SHA256Managed();
                break;
            case HashAlgorithmEnum.SHA384:
                this.hashObject = new SHA384Managed();
                break;
            case HashAlgorithmEnum.SHA512:
                this.hashObject = new SHA512Managed();
                break;
            case HashAlgorithmEnum.MD5:
                this.hashObject = new MD5CryptoServiceProvider();
                break;
            case HashAlgorithmEnum.HMACSHA1:
                this.hashObject = new HMACSHA1();
                break;
        }
    }
}
```

Listing 405: Der Konstruktor der Hasher-Klasse

```

        case HashAlgorithmEnum.MACTripleDES:
            this.hashObject = new MACTripleDES();
            break;
    }
}

```

Listing 405: Der Konstruktor der Hasher-Klasse (Forts.)

Damit für die Algorithmen, die Schlüssel verwenden (HMACSHA1, MACTripleDES), der Schlüssel definiert werden kann, besitzt die Klasse `Hasher` eine Eigenschaft `Key`, der Sie den Schlüssel als `String` übergeben können. Um sicherzustellen, dass der in einem `Byte-Array` verwaltete Schlüssel ausgelesen bzw. gesetzt werden kann, interpretieren beide Zugriffsmethoden von `Key` das aktuelle Hash-Objekt als Instanz der Klasse `KeyedHashAlgorithm`, von der alle Hash-Algorithmus-Klassen abgeleitet sind, die einen Schlüssel einsetzen. Ergibt diese Interpretation den Wert `null`, generieren `set` und `get` eine Ausnahme vom Typ `NotSupportedException`.

Die `set`-Methode überprüft den übergebenen Schlüssel zusätzlich darauf, ob dieser Zeichen mit einem Code größer `0x00FF` speichert. Damit wird sichergestellt, dass keine Unicode-Zeichen übergeben werden, die den 8-Bit-Bereich überschreiten. Als Codierung für den `String` verwende ich die zu Unicode kompatible ISO-8859-1-Codierung. Die Begründung für den Verzicht auf den kompletten Unicode-Zeichensatz liegt darin, dass die meisten `Strings` in unseren Breiten lediglich den 8-Bit-Bereich verwenden. Bei einer Unicode-Codierung wäre jedes zweite `Byte` des Schlüssels in diesem Fall 0, was diesen in der Qualität verschlechtern würde.

```

public string Key
{
    get
    {
        // Überprüfen, ob der Algorithmus einen Schlüssel erlaubt,
        // und Speichern des Schlüssels
        KeyedHashAlgorithm kha = this.hashObject as KeyedHashAlgorithm;
        if (kha != null)
            // Schlüssel in einen String umwandeln und zurückgeben
            return Encoding.GetEncoding("ISO-8859-1").GetString(kha.Key);
        else
            throw new NotSupportedException("Der aktuell verwendete " +
                "Hash-Algorithmus unterstützt keine Schlüssel");
    }

    set
    {
        // Auf Unicode-Zeichen größer 0x00FF überprüfen
        for (int i = 0; i < value.Length; i++)
        {
            if ((int)value[i] > 255)

```

Listing 406: Eigenschaft zum Lesen und Schreiben des Schlüssels

```

        throw new CryptographicException("Der übergebene " +
            "Schlüssel enthält mindestens ein Unicode-Zeichen, " +
            "das größer ist als 0x00FF (255): " + value[i] + " (" +
            (int)value[i] + "). Unterstützt werden lediglich " +
            "8-Bit-Unicode-Zeichen");
    }

    // Überprüfen, ob der Algorithmus einen Schlüssel erlaubt,
    // und Speichern des Schlüssel
    KeyedHashAlgorithm kha = this.hashObject as KeyedHashAlgorithm;
    if (kha != null)
        // Schlüssel in einen String umwandeln und zurückgeben
        kha.Key = Encoding.GetEncoding("ISO-8859-1").GetBytes(value);
    else
        throw new NotSupportedException("Der aktuell verwendete " +
            "Hash-Algorithmus unterstützt keine Schlüssel");
    }
}

```

Listing 406: Eigenschaft zum Lesen und Schreiben des Schlüssels (Forts.)

Nun folgen nur noch die wichtigen Methoden zum Erzeugen des Hashcodes. Die vier Varianten der Methode `ComputeHash` erwarten ein Byte-Array, einen Stream oder einen String als Quelle, erzeugen daraus einen Hashcode und geben diesen als ISO-8859-1-String zurück.

```

/* Methoden zum Erzeugen eines Hash aus einem Byte-Array */
public string ComputeHash(byte[] data)
{
    return Encoding.GetEncoding("ISO-8859-1").GetString(
        this.hashObject.ComputeHash(data));
}

public string ComputeHash(byte[] data, int offset, int count)
{
    return Encoding.GetEncoding("ISO-8859-1").GetString(
        this.hashObject.ComputeHash(data, offset, count));
}

/* Methode zum Erzeugen eines Hash aus einem Stream */
public string ComputeHash(Stream inputStream)
{
    return Encoding.GetEncoding("ISO-8859-1").GetString(
        this.hashObject.ComputeHash(inputStream));
}

/* Methode zum Erzeugen eines Hash aus einem String */
public string ComputeHash(string inputString)
{
    // Byte-Array aus dem String erzeugen und damit den Hashcode erzeugen

```

Listing 407: Methoden zur Erzeugung eines Hashcodes

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
        byte[] buffer = Encoding.Unicode.GetBytes(inputString);  
        return Encoding.GetEncoding("ISO-8859-1").GetString(  
            this.hashObject.ComputeHash(buffer));  
    }  
}
```

Listing 407: Methoden zur Erzeugung eines Hashcodes (Forts.)

Die Hasher-Klasse ist zwar etwas aufwändiger geworden, als ich ursprünglich vermutete. Die Anwendung dieser Klasse ist dafür aber sehr einfach. Das folgende Beispiel erzeugt aus einem String über das HMACSHA1-Verfahren (das mit einem Schlüssel arbeitet) einen Hashcode:

```
// Hasher erzeugen und Schlüssel übergeben  
Hasher hasher = new Hasher(HashAlgorithmEnum.HMACSHA1);  
hasher.Key = "l6ke08325ybnk y973235298`46t B9872598";  
  
// Den Hashcode für einen String ermitteln  
string source = "Das ist ein Teststring zur Ermittlung eines Hashcodes";  
string hashCode = hasher.ComputeHash(source);  
Console.WriteLine("Quellstring: {0}", source);  
Console.WriteLine("Hashcode: {0}", hashCode);
```

Listing 408: Erzeugen eines Hashcodes für einen String



Abbildung 150: HMACSHA1-Hashcode für einen String

Sie können aber genauso gut einen Hashcode für eine Datei erzeugen, indem Sie `ComputeHash` ein `FileStream`-Objekt übergeben oder ein `Byte`-Array als Quelle verwenden.

214 Wave-Dateien über das Windows-API abspielen

Da das .NET Framework zurzeit noch keinen Multimedia-Support bietet, können Sie Wave-Dateien (mit der Endung *.wav*) nur über externe Komponenten abspielen. Neben verschiedenen Techniken wie WMI und DirectX, die in den folgenden Rezepten beschrieben werden, können Sie dazu die Windows-API-Funktionen `PlaySound` und `sndPlaySound` verwenden. Die Verwendung der API-Funktionen bietet den Vorteil, dass Sie nicht darauf angewiesen sind, dass spezielle Treiber oder Komponenten auf dem Zielsystem installiert sind.

Beide Funktionen sind nahezu identisch, `PlaySound` bietet über ein zusätzliches Argument und spezielle Flags leicht erweiterte Möglichkeiten wie z.B. das Abspielen einer Wave-Datei, die als Ressource in einer externen ausführbaren Datei gespeichert ist. Für unsere Zwecke reicht die etwas einfachere Funktion `sndPlaySound` aber aus.

Die Methode `PlayWaveFile` in Listing 409 implementiert das Abspielen einer Wave-Datei über `sndPlaySound`. Neben dieser Funktion sind zunächst einige für den Aufruf wichtige Konstanten deklariert, die am zweiten Argument übergeben werden. Über diese Konstanten können Sie u.a. bestimmen, dass der Sound asynchron abgespielt werden soll (`SND_ASYNC`), dass er in einer Schleife wiederholt werden soll, bis `sndPlaySound` erneut aufgerufen wird (`SND_LOOP`), und dass der aktuell abgespielte Sound nicht durch einen weiteren Aufruf der Funktion (ohne Angabe des Dateinamens) gestoppt werden kann (`SND_NOSTOP`). Die einzelnen Konstanten können Sie über `|` miteinander kombinieren. Beim asynchronen Abspielen kehrt `sndPlaySound` direkt nach dem Aufruf wieder zurück und Ihre Anwendung kann weiterarbeiten, während die Wave-Datei im Hintergrund abgespielt wird. Durch einen weiteren Aufruf von `sndPlaySound` unter der Übergabe von `null` am ersten Argument können Sie das aktuelle Abspielen stoppen, sofern beim vorherigen Aufruf nicht die Konstante `SND_NOSTOP` mit angegeben war.

Die Konstante `SND_MEMORY` ermöglicht übrigens das Abspielen einer Sounddatei, die im Arbeitsspeicher gespeichert ist, und somit auch den Zugriff auf Wave-Dateien in Ressourcen (was in diesem Rezept aber nicht umgesetzt ist).

Die Methode `PlayWaveFile` in Listing 409 überprüft zunächst, ob die Datei eventuell nicht vorhanden ist, und generiert in diesem Fall eine Ausnahme. Ich habe diese Ausnahme integriert um bei der Übergabe eines ungültigen Dateinamens eine vernünftige Fehlermeldung zu erhalten.

Vor dem Aufruf von `sndPlaySound` setzt `PlayWaveFile` die Flags entsprechend den an den Argumenten `async` und `loop` übergebenen Werten zusammen. Dabei wird auch das Flag `SND_NODEFAULT` gesetzt, damit `sndPlaySound` nicht den Windows-Defaultsound abspielt, wenn eine nicht existierende Datei oder eine Datei in einem falschen Format übergeben wird.

`sndPlaySound` gibt Null zurück, wenn ein Fehler aufgetreten ist. In diesem Fall wirft `PlayWaveFile` eine Ausnahme mit einer Fehlermeldung, deren Text über `FormatMessage` ermittelt wird. Die dadurch erzeugte Fehlermeldung (z.B. bei einer nicht existenten Soundkarte) ist wesentlich sprechender als die Fehlernummer, die `Marshal.GetLastWin32Error` zurückgibt.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Text`, `System.IO` und `System.Runtime.InteropServices` einbinden.

```

/* Deklaration der benötigten API-Funktionen und -Konstanten */
[DllImport("winmm.dll", SetLastError=true)]
private extern static int sndPlaySound(string lpszSound, int fuSound);

private const int SND_ASYNC = 0x0001;
private const int SND_LOOP = 0x0008;
private const int SND_NOSTOP = 0x0010;
private const int SND_NODEFAULT = 0x0002;

[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
    int dwMessageId, int dwLanguageId, System.Text.StringBuilder lpBuffer,
    int nSize, string [] Arguments);

private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;

/* Methode zum Abspielen einer Wave-Datei */
public static void PlayWaveFile(string fileName, bool async, bool loop)
{
    // Überprüfen ob die Datei existiert
    if (fileName != null && File.Exists(fileName) == false)
        throw new FileNotFoundException("Die Datei '" + fileName +
            "' existiert nicht", fileName);

    // Flags zusammensetzen
    int flags = SND_NODEFAULT;
    if (async) flags |= SND_ASYNC;
    if (loop) flags |= SND_LOOP;

    // Datei abspielen
    if (sndPlaySound(fileName, flags) == 0)
    {
        // Fehler auswerten
        StringBuilder message = new StringBuilder(1024);
        string errorMessage = "Fehler " + Marshal.GetLastWin32Error() +
            " beim Abspielen der Wave-Datei '" + fileName + "'";
        if (FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, (IntPtr)0,
            Marshal.GetLastWin32Error(), 0, message, 1024, null) > 0)
            errorMessage += ": " + message.ToString();
        throw new Exception(errorMessage);
    }
}

```

Listing 409: Methode zum Abspielen einer Wave-Datei über `sndPlaySound`

Unter Windows 98 (und wahrscheinlich auch unter Windows Me) kann `sndPlaySound` nicht in einer Konsolenanwendung ausgeführt werden. Das liegt daran, dass die Konsole unter Windows 98 im 16-Bit-Modus läuft, während `sndPlaySound` eine 32-Bit-Funktion ist.

215 Multimedia-Dateien (Wave, MP3, Midi, AVI, MPEG etc.) über MCI abspielen

Das relativ alte MCI (Media Control Interface) ermöglicht den Zugriff auf alle Multimedia-Dienste, die vom Betriebssystem zur Verfügung gestellt werden. MCI ist in Form von verschiedenen API-Funktionen in der Datei `winmm.dll` bereits seit Windows 3.1 Teil des Betriebssystems.

An Stelle von MCI können Sie auch DirectX für das Abspielen von Multimedia-Dateien verwenden, wie ich es im folgenden Rezept beschreibe. MCI besitzt aber den Vorteil, dass Sie davon ausgehen können, dass eigentlich jeder Computer, auf dem das .NET Framework (in der Windows-Version) installiert ist, über die notwendigen MCI-Treiber verfügt. DirectX muss hingegen separat installiert werden.

MCI arbeitet mit Treibern, die den Zugriff auf spezielle Multimedia-Geräte (bzw. -Programme) so kapseln, dass immer dieselben MCI-Funktionen zur Steuerung der Geräte verwendet werden können. Für spezielle Multimedia-Formate wie z.B. das Apple-QuickTime-Format können Sie spezielle MCI-Treiber beziehen (meist reicht dazu eine Google-Suche aus). Oft sind MCI-Treiber aber auch in der Installation eines Multimedia-Programms (wie z.B. bei QuickTime) enthalten. Die Microsoft-Treiber werden hauptsächlich vom Windows Media Player installiert. Ältere Betriebssysteme besitzen natürlich per Voreinstellung nur ältere Treiber und können mit Multimediadateien neuerer Generation nichts anfangen. Windows 98 ist allerdings nach einer frischen Installation (mit dem Internet Explorer 6 und dem .NET Framework 1.0) bereits in der Lage die wichtigsten Formate (Wave, MP3, AVI, MPEG) abzuspielen.

Falls eine Aktualisierung der Treiber notwendig sein sollte, reicht wahrscheinlich die Installation der neuesten Version des Media Players aus. Ein separater Download der Microsoft-MCI-Treiber scheint auf der Microsoft-Website leider nicht verfügbar zu sein.

MCI-Geräte können Sie im Wesentlichen über zwei Funktionen steuern. Der Funktion `mciSendCommand` übergeben Sie dazu eine Struktur mit den Befehlsparametern. Diese Variante ist etwas umständlich. Die einfachere Variante über die Funktion `mciSendString` arbeitet mit einem Befehls-String und ist in der Anwendung wesentlich einfacher. Zur Arbeit mit MCI benötigen Sie dann nur noch die Funktion `mciGetErrorString` zum Auslesen aufgetretener Fehler und etwas Wissen um die Befehls-Strings. Der Vorgang bei der Arbeit mit MCI-Geräten ist immer derselbe: Das Gerät wird zuerst geöffnet, dann können Sie Befehle wie z.B. `Play` zum Abspielen oder `Stop` zum Stoppen aufrufen, zum Schluss wird das Gerät wieder geschlossen.

Problematisch ist dabei lediglich das Herausfinden der korrekten MCI-Befehlsstrings. Diese sind in der MCI-Dokumentation recht gut in Zusammenhang mit der `mciSendString`-Funk-

tion beschrieben (über den Link MULTIMEDIA COMMAND STRINGS). Die MCI-Dokumentation finden Sie in der Plattform-SDK-Dokumentation unter GRAPHICS AND MULTIMEDIA SERVICES / WINDOWS MULTIMEDIA / MULTIMEDIA AUDIO / MCI. Lassen Sie sich vom Namen des vorletzten Ordners nicht verwirren: Moderne Systeme liefern auch MCI-Treiber für Videodateien mit. Im Internet finden Sie die MCI-Referenz am besten, indem Sie bei *msdn.microsoft.com* nach »MCI« suchen. Die Adresse der MCI-Referenz bezieht sich anscheinend auf die Version und wird sich mit ziemlicher Sicherheit ändern. Aber vielleicht haben Sie Glück: *msdn.microsoft.com/library/en-us/multimed/mci_7vvt.asp*.

So, das reicht an Theorie. Nun folgt die Klasse `Mci`, die Ihnen die Arbeit mit MCI sehr erleichtert. Diese Klasse implementiert zwar lange nicht alle Möglichkeiten, die MCI bietet. Sie können aber über `Mci`-Instanzen Audio- und Video-Dateien abspielen. Trotzdem ist die Klasse bereits recht komplex. Aber die Arbeit (des Kopierens der Klasse aus dem Repository ...) lohnt sich: Sie erhalten damit eine sehr einfach anzuwendende Möglichkeit, Sound- und Videodateien abzuspielen.

Zum Kompilieren der Klasse müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Text`, `System.Runtime.InteropServices` und `System.Windows.Forms` importieren.

Zur Implementierung habe ich zunächst eine Klasse `MciException` für eigene Ausnahmen deklariert. `MciException` erwartet im Konstruktor den Kontext des Fehlers (also eine Info darüber, wann dieser entstanden ist) und die Fehlermeldung. Damit ist die Ausnahmemeldung sehr aussagekräftig, was auch das Debuggen von Fehlern erleichtert, die nur beim Anwender auftreten.

```
public class MciException: Exception
{
    public MciException(string context, string message):
        base("Fehler beim " + context + ": " + message)
    {
    }
}
```

Listing 410: Klasse für eigene Ausnahmen

Die Klasse `Mci` ist von `IDisposable` abgeleitet, damit der Aufruf der `Dispose`-Methode, die später implementiert wird, möglich ist. Am Anfang der Klasse habe ich die verwendeten API-Funktionen deklariert:

```
public class Mci: IDisposable
{
    /* Deklaration der MCI-Funktionen, -Konstanten und -Strukturen */
    [DllImport("winmm.dll", CharSet=CharSet.Auto)]
    private static extern int mciSendString(string lpstrCommand,
        StringBuilder lpstrReturnString, int uReturnLength,
```

Listing 411: Beginn der Klasse MCI mit der Deklaration der verwendeten API-Funktionen

```

        IntPtr hwndCallback);

[DllImport("winmm.dll", CharSet=CharSet.Auto)]
private static extern int mciGetErrorString(int dwError,
        StringBuilder lpstrBuffer, int uLength);

[DllImport("kernel32.dll", CharSet=CharSet.Auto)]
private static extern int GetShortPathName(string lpzLongPath,
        StringBuilder lpzShortPath, int cchBuffer);

```

Listing 411: Beginn der Klasse MCI mit der Deklaration der verwendeten API-Funktionen (Forts.)

MCI-Geräte werden über eine beim Öffnen automatisch vergebene Geräte-Id oder einen Alias angesprochen. Ich verwende einen Alias, der im Fall der `Mci`-Klasse einfach ein GUID-Wert ist. Da dieser Alias in verschiedenen Methoden benötigt wird, speichert MCI diesen in der privaten Eigenschaft `alias`.

Eine schreibgeschützte Eigenschaft `IsOpen` macht eine Aussage darüber, ob das MCI-Gerät geöffnet ist, und wird von den Methoden zum Öffnen und Schließen gesetzt.

Zur Vereinfachung der Ermittlung von MCI-Fehlern habe ich dann die Methode `GetMciError` geschrieben. Diese Methode ruft die Funktion `mciGetErrorString` mit dem übergebenen Fehlercode auf und gibt den ermittelten String zurück. `mciGetErrorString` gibt, wie viele API-Funktionen, die mit Strings arbeiten, die Anzahl der gelesenen Zeichen zurück. Bei einer Rückgabe von 0 (z.B. bei der Übergabe eines unbekannten Fehlercodes) gibt `GetMciError` lediglich eine generische Fehlermeldung zurück:

```

/* Eigenschaft für den MCI-Alias */
private string alias;

/* Eigenschaft, die angibt, ob gerade ein Gerät geöffnet ist */
private bool isOpen = false;
public bool IsOpen
{
    get {return this.isOpen;}
}

/* Methoden zum Auslesen von MCI-Fehlern */
private string GetMciError(int errorCode)
{
    StringBuilder errorMessage = new StringBuilder(255);
    if (mciGetErrorString(errorCode, errorMessage,
        errorMessage.Capacity) == 0)
        return "MCI-Fehler " + errorCode;
    else
        return errorMessage.ToString();
}

```

Listing 412: Private Eigenschaften und eine Methode zur Ermittlung eines MCI-Fehlers

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Nun folgen Methoden zum Öffnen eines MCI-Geräts (automatisch über den Dateinamen). Da MCI auch das Abspielen von Videodateien auf einem gegebenen Windows-Fenster (einem Formular, einem `PictureBox`-Steuerelement etc.) erlaubt, habe ich zwei Varianten der `Open`-Methode entwickelt. Die erste, die lediglich den Dateinamen der Multimedia-Datei übergeben bekommt, ruft dabei einfach die zweite auf. Diese zweite Variante, die neben dem Dateinamen eine `Control`-Referenz erwartet (womit Sie alle .NET-Steuerelemente und Formulare übergeben können), erzeugt zunächst einen eindeutigen Alias für das MCI-Gerät. Sie nutzt dazu einfach die `NewGuid`-Methode der `Guid`-Klasse, die eine weltweite eindeutige GUID erzeugt. Um die nicht sinnvollen Bindestriche aus der String-Repräsentation des GUID-Werts zu entfernen, ruft `Open` die `ToString`-Methode der erzeugten `Guid`-Instanz mit dem Formatzeichen »N« (numerische Formatierung) auf.

Dann überprüft `Open`, ob die angegebene Datei existiert, und generiert im Fehlerfall eine Ausnahme vom Typ `FileNotFoundException`. Damit wird verhindert, dass MCI beim Öffnen einer nicht vorhandenen Datei einen Fehler erzeugt und dieser in einer `MciException` resultiert.

Der im `Open`-Befehl später übergebene Dateinamen kann in Anführungszeichen gesetzt werden, damit MCI den Befehl korrekt interpretieren kann (MCI trennt die Befehlsparameter durch Leerzeichen). Nach Aussagen in einer Newsgroup hat MCI aber Probleme mit sehr langen Dateinamen, was ich unter Windows XP allerdings nicht nachvollziehen konnte. Zur Sicherheit setzt `Open` den langen Dateinamen über `GetShortPathName` in einen kurzen um und verzichtet im MCI-String auf die Anführungszeichen.

Dann wird der MCI-String zum Öffnen zusammengesetzt. Dieser sieht in unserem Fall beispielsweise zum Abspielen einer MP3-Datei folgendermaßen aus, wenn kein Besitzer-Fenster angegeben ist:

```
"open C:\Patience.mp3 type mpegvideo alias 9343e55ade624346bb9a0729425d0bb4"
```

Der Parameter `type` gibt den zu verwendenden Gerätetyp an. Laut der Dokumentation können Sie nur spezifische Gerätenamen wie z.B. `cdaudio` (CD-ROM), `waveaudio` (Wave-Dateien) oder `digitalvideo` (Videodateien) einsetzen. Etwas problematisch ist dabei die notwendige Unterscheidung der verschiedenen Typen.

Der undokumentierte Typ `mpegvideo` (der weder in der Dokumentation noch auf der Microsoft Website zu finden ist) spricht hingegen den MPEGVideo-Treiber (*mciqtz.drv*) an, der in der Lage ist, alle wichtigen Audio- und Videoformate abzuspielen. Und das erleichtert die Arbeit ungemein.

Falls am zweiten Argument eine `Control`-Referenz übergeben wurde, hängt `Open` noch den Zusatz `parent Handle style child` mit dem Handle des Objekts an den MCI-Befehlsstring an, damit ein Video auf der Fläche des Controls ausgegeben wird.

Mit dem korrekt zusammengesetzten MCI-String ruft `Open` dann die `mciSendString`-Funktion auf. Da keine Daten zurück erwartet werden (einige MCI-Befehle lesen Statusdaten oder allgemeine Informationen zum Gerät aus), wird am zweiten Argument (das eine `StringBuilder`-Instanz erwartet) `null` und am dritten `0` übergeben. Das letzte Argument nimmt die Adresse einer Callback-Funktion auf, die von MCI bei festzulegenden Statusänderungen auf-

gerufen wird. Dieses interessante Feature (z.B. für eine Fortschrittmeldung) habe ich aus Platzgründen aber nicht implementiert. `Open` übergibt hier als `IntPtr.Zero`.

Tritt bei der Ausführung von `mciSendString` ein Fehler auf, wird dieser über `GetMciError` in eine Fehlermeldung umgesetzt und an eine neue `MciException` übergeben.

Positions- und Längenangaben in MCI-Geräten verwenden ein bestimmtes Zeitformat. Bei den meisten Geräten sind das Millisekunden. Es kann aber bei spezifischen Geräten vorkommen, dass das Zeitformat ein anderes ist, wie z.B. eine Angabe in Stunden, Minuten und Sekunden. Um sicherzustellen, dass möglichst immer Millisekunden verwendet werden, stellt `Open` schließlich noch das aktuelle Zeitformat über den `set`-Befehl ein.

```

public void Open(string fileName)
{
    Open(fileName, null);
}

public void Open(string fileName, Control owner)
{
    // Alias für das MCI-Gerät erzeugen
    this.alias = Guid.NewGuid().ToString("N");

    // Überprüfen, ob die Datei existiert
    if (File.Exists(fileName) == false)
        throw new FileNotFoundException("Die Datei '" + fileName +
            "' existiert nicht", fileName);

    // Den kurzen Dateinamen ermitteln
    StringBuilder shortPath = new StringBuilder(261);
    if (GetShortPathName(fileName, shortPath, shortPath.Capacity) == 0)
        throw new MciException("Auslesen des kurzen Dateinamens für '" +
            fileName + "'", "Windows-Fehler " + Marshal.GetLastWin32Error());

    // MCI-Befehlsstring zum Öffnen zusammensetzen
    string mciString = "open " + shortPath.ToString() +
        " type mpegvideo alias " + this.alias;

    if (owner != null)
        mciString += " parent " + (int)owner.Handle + " style child";

    // MCI-Gerät öffnen
    int result = mciSendString(mciString, null, 0, IntPtr.Zero);
    if (result != 0)
        throw new MciException("Öffnen des MCI-Geräts", GetMciError(result));

    this.isOpen = true;

    // Das Zeitformat für Längen- und Positionsangaben explizit
    // auf Millisekunden setzen
    mciString = "set " + this.alias + " time format ms" ;

```

Listing 413: Methoden zum Öffnen eines MCI-Geräts

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        result = mciSendString(mciString, null, 0, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Setzen des Zeitformats",
                                   GetMciError(result));
    }

```

Listing 413: Methoden zum Öffnen eines MCI-Geräts (Forts.)

Nun folgt zunächst eine Eigenschaft `Length`, die über den MCI-Befehlsstring »status Alias length« die Abspiellänge der Datei (im aktuellen Zeitformat) ermittelt. Dazu wird natürlich wieder `mciSendString` aufgerufen, dieses Mal aber mit einem ausreichend groß dimensionierten `StringBuilder`-Objekt, in das `mciSendString` das Ergebnis schreibt. Tritt beim Aufruf kein Fehler auf, wird das Ergebnis über `int.Parse` in einen `int`-Wert umgewandelt. Ich verwende explizit die `Parse`-Methode des `int`-Typs, da die Länge für spezifische Gerätetypen laut der Dokumentation trotz der expliziten Definition in `Open` bei einigen wenigen Medientypen auch in einer nicht numerischen Form zurückgegeben werden kann. Eine Konvertierung mit `Convert.ToInt32` würde in diesem Fall fehlschlagen.

```

public int Length
{
    get
    {
        StringBuilder buffer = new StringBuilder(261);
        int result = mciSendString("status " + this.alias + " length",
                                   buffer, buffer.Capacity, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Lesen der Länge", GetMciError(result));

        return int.Parse(buffer.ToString());
    }
}

```

Listing 414: Eigenschaft zur Ermittlung der Abspiellänge

Die nun folgende `Play`-Methode zum Abspielen der geöffneten Datei ist in drei Varianten implementiert, wobei die erste und die zweite jeweils die dritte aufrufen. Diese erwartet eine Startposition (in der aktuellen Längeneinheit, die wir ja auf Millisekunden gesetzt haben), eine Länge und eine Info darüber, ob der Abspielvorgang am Ende wiederholt werden soll. Der eigentliche Aufruf des MCI-Befehls entspricht dann dem beim Öffnen, nur dass hier natürlich ein anderer Befehlsstring verwendet wird. Die ersten beiden Varianten der Methode fragen über die `Length`-Eigenschaft einfach die aktuelle Länge der Datei ab um die Abspiellänge zu ermitteln.

```

public void Play(bool repeat)
{
    Play(0, this.Length, repeat);
}

public void Play(int from, bool repeat)
{
    Play(from, this.Length - from, repeat);
}

public void Play(int from, int to, bool repeat)
{
    string mciString = "play " + this.alias + " from " + from + " to " + to;
    if (repeat)
        mciString += " repeat";
    int result = mciSendString(mciString, null, 0, IntPtr.Zero);
    if (result != 0)
        throw new MciException("Aufruf von Play", GetMciError(result));

    this.isPlaying = true;
}

```

Listing 415: Methoden zum Abspielen der Multimedia-Datei

Die Eigenschaft `Volume` ermöglicht das Abfragen und Setzen der aktuellen Lautstärke. Die Abfrage erfolgt über den MCI-Befehl `status`, das Setzen über den Befehl `setaudio`. Die in meinen Versuchen möglichen Werte reichten von 0 bis 1000, leider ist dies aber nicht dokumentiert. Bei einem Wert von 1000 entspricht die Lautstärke der, die aktuell für den Soundtreiber eingestellt ist. Bei einem ungültigen Wert generiert `mciSendString` einen Fehler, der wie in allen anderen Methoden dieser Klasse auch in eine Ausnahme umgesetzt wird.

```

public int Volume
{
    get
    {
        StringBuilder buffer = new StringBuilder(261);
        int result = mciSendString("status " + this.alias + " volume",
            buffer, buffer.Capacity, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Lesen von Volume", GetMciError(result));
        return int.Parse(buffer.ToString());
    }
    set
    {
        int result = mciSendString("setaudio " + this.alias + " volume to " +
            value, null, 0, IntPtr.Zero);
    }
}

```

Listing 416: Eigenschaft für die Lautstärke

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        if (result != 0)
            throw new MciException("Aufruf von SetAudio",
                                   GetMciError(result));
    }
}

```

Listing 416: Eigenschaft für die Lautstärke (Forts.)

Über die Eigenschaft `Position` können Sie die aktuelle Position im Abspiel-Stream lesen und setzen. Die Arbeitsweise dieser Eigenschaft ist ähnlich der von `Volume`. Nach dem Setzen der neuen Position ruft `Position` den `Play`-Befehl auf, da der `Seek`-Befehl dazu führt, dass der Abspielvorgang angehalten wird. So können Sie auch mitten im Abspielen vor- und zurückspulen. Als Nebeneffekt bewirkt dies, dass der Abspielvorgang auch dann beginnt, wenn aktuell noch keine `Play`-Methode aufgerufen wurde, aber die Position gesetzt wird.

```

public int Position
{
    get
    {
        StringBuilder buffer = new StringBuilder(261);
        int result = mciSendString("status " + this.alias + " position",
                                   buffer, buffer.Capacity, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Lesen von Position", GetMciError(result));
        return int.Parse(buffer.ToString());
    }
    set
    {
        int result = mciSendString("seek " + this.alias + " to " + value,
                                   null, 0, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Aufruf von Seek", GetMciError(result));
        result = mciSendString("play " + this.alias, null, 0, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Aufruf von Play", GetMciError(result));
    }
}

```

Listing 417: Eigenschaft für die Position des aktuellen Abspielvorgangs

Nun folgt die Methode `SetRectangle`, über die Sie die Position eines auf einem Formular oder Steuerelement abgespielten Videos bestimmen können. Diese Methode ist z.B. dann interessant, wenn Sie die Größe des Videos bei der Größenänderung eines Formulars anpassen wollen. Übergeben werden die *x*- und *y*-Position, die Breite und die Höhe. `SetRectangle` ruft zur Definition der Position und der Größe den MCI-Befehl »*put Alias window at x y Breite Höhe*« auf:

```

public void SetRectangle(int x, int y, int width, int height)
{
    string mciString = "put " + this.alias + " window at " + x + " " + y +
        " " + width + " " + height;
    int result = mciSendString(mciString, null, 0, IntPtr.Zero);
    if (result != 0)
        throw new MciException("Aufruf von Put Window at",
            GetMciError(result));
}

```

Listing 418: Methode zur Definition der Position und der Größe des Videos bei einer Ausgabe auf einem Formular oder Steuerelement

Schließlich folgen nur noch eine Methode `Stop` zum Stoppen des aktuellen Abspielvorgangs, eine Methode `Close` zum Schließen des MCI-Geräts, die `Dispose`-Methode (der `IDisposable`-Schnittstelle) und der Destruktor. `Stop` und `Close` rufen die entsprechenden MCI-Befehle auf, wobei `Close` noch die Eigenschaft `isOpen` zurücksetzt. `Dispose` und der Destruktor rufen einfach nur `Close` auf:

```

/* Methode zum Stoppen */
public void Stop()
{
    string mciString = "stop " + this.alias;
    int result = mciSendString(mciString, null, 0, IntPtr.Zero);
    if (result != 0)
        throw new MciException("Aufruf von Stop", GetMciError(result));
}

/* Methode zum Schließen */
public void Close()
{
    if (this.isOpen)
    {
        string mciString = "close " + this.alias;
        int result = mciSendString(mciString, null, 0, IntPtr.Zero);
        if (result != 0)
            throw new MciException("Aufruf von Close", GetMciError(result));

        this.isOpen = false;
    }
}

/* Dispose-Methode */
public void Dispose()
{
    this.Close();
}

/* Destruktor */

```

Listing 419: Methoden zum Stoppen, zum Schließen, die Dispose-Methode und der Destruktor

```

~Mci()
{
    this.Close();
}
}

```

Listing 419: Methoden zum Stoppen, zum Schließen, die Dispose-Methode und der Destruktor (Forts.)

Den Destruktor und die `Dispose`-Methode habe ich zur Sicherheit implementiert, damit beim Ungültigwerden einer `Mci`-Instanz auf jeden Fall alle MCI-Ressourcen freigegeben werden. `Dispose` wird, wie Sie ja sicherlich wissen, automatisch aufgerufen, wenn die Instanz innerhalb einer `using`-Anweisung erzeugt wird:

```

using (Mci mci = new Mci())
{
    mci.Open("H:\\Videos\\axe.mpeg");
    mci.Play(false);
    MessageBox.Show("Playing ...");
}

```

Wenn Sie Objekte in einer `using`-Anweisung instanzieren, ist sichergestellt, dass diese beim Ungültigwerden immer die `Dispose`-Methode aufrufen und die Ressourcen unabhängig vom Garbage Collector freigeben.

Ein kleines Beispielprogramm (Abbildung 151) soll die Anwendung der `Mci`-Klasse erläutern. Das Programm soll Audio- und Videodateien abspielen.



Abbildung 151: Abspielen eines Mpeg-Videos in einem PictureBox-Steuerelement

Das Formular erzeugt im Load-Ereignis eine Instanz der Mci-Klasse, was ich hier nicht darstelle. Die Referenz wird in der privaten Eigenschaft `mci` verwaltet. In den im Folgenden beschriebenen Methoden verzichte ich im Buch aus Platz- und Übersichtsgründen auch auf die Ausnahmebehandlung (die Sie im Beispiel auf der CD aber vorfinden).

In der Ereignismethode für das Click-Ereignis des ÖFFNEN-Schalters wird das MCI-Gerät geöffnet. Ist die obere CheckBox eingeschaltet, wird als Handle für die Ausgabe eines Videos das Handle eines PictureBox-Steuerelements übergeben, das auf dem Formular angelegt ist. In diesem Fall erfolgt eine Videoausgabe auf dem Steuerelement. Wird kein Handle übergeben, erzeugt MCI für die Ausgabe ein eigenes Fenster. Sounddateien werden natürlich nicht in einem Fenster ausgegeben. Die Übergabe eines Handles führt aber hier nicht zu einem Fehler.

```
private void openButton_Click(object sender, System.EventArgs e)
{
    // MCI-Gerät öffnen
    if (this.openVideoInFormCheckBox.Checked)
        mci.Open(fileNameTextBox.Text, this.pictureBox.Handle);
    else
        mci.Open(fileNameTextBox.Text);

    // Länge und aktuelle Lautstärke auslesen
    infoLabel.Text = "Länge: " + mci.Length + " ms";
    this.volumeTextBox.Text = mci.Volume.ToString();
}
```

Listing 420: Öffnen des MCI-Geräts mit der angegebenen Datei

Die Methode für den ABSPIELEN-Schalter ruft lediglich die Play-Methode auf, wobei der Zustand der zweiten Checkbox am `repeat`-Argument übergeben wird:

```
private void playButton_Click(object sender, System.EventArgs e)
{
    mci.Play(this.repeatCheckBox.Checked);
}
```

Listing 421: Abspielen der geöffneten Multimedia-Datei

Die Methoden für den LAUTSTÄRKE- und den VORSPULEN-Schalter lesen bzw. schreiben die Eigenschaften `Volume` und `Position`:

```
private void volumeButton_Click(object sender, System.EventArgs e)
{
    mci.Volume = Convert.ToInt32(this.volumeTextBox.Text);
}

private void forwardButton_Click(object sender, System.EventArgs e)
```

Listing 422: Einstellen der Lautstärke und der Position

```
{  
    int position = mci.Position;  
    this.infoLabel.Text = "Aktuelle Position: " + position;  
    mci.Position = position + 10000;  
}
```

Listing 422: Einstellen der Lautstärke und der Position (Forts.)

Die Methoden des STOP- und des SCHLIEßEN-Schalters rufen die Methoden `Stop` und `Close` auf:

```
private void stopButton_Click(object sender, System.EventArgs e)  
{  
    mci.Stop();  
}  
  
private void closeButton_Click(object sender, System.EventArgs e)  
{  
    mci.Close();  
}
```

Listing 423: Stoppen und Schließen des aktuellen Abspielvorgangs

Die Methode für das `Resize`-Ereignis des Formulars ruft `SetRectangle` auf, um das Video an die Größe der `PictureBox` anzupassen (die im Beispiel über die `Anchor`-Eigenschaft automatisch an die Größe des Formulars angepasst wird). Dabei fragt die Methode zuvor ab, ob das MCI-Gerät überhaupt geöffnet ist, um einen Fehler beim Aufruf des MCI-Befehls bei einem nicht geöffneten Gerät zu vermeiden (das könnten die anderen Methoden dieses Beispiels natürlich auch so implementieren):

```
private void StartForm_Resize(object sender, System.EventArgs e)  
{  
    if (mci.IsOpen)  
    {  
        mci.SetRectangle(0, 0, this.pictureBox.Width, this.pictureBox.Height);  
    }  
}
```

Listing 424: Anpassen der Videogröße im Resize-Ereignis

Im `Closing`-Ereignis des Formulars wird die `Close`-Methode noch einmal aufgerufen:

```
private void StartForm_Closing(object sender,  
    System.ComponentModel.CancelEventArgs e)  
{  
    mci.Close();  
}
```

Listing 425: Schließen des MCI-Geräts im Closing-Ereignis

Dieses Schließen ist deswegen wichtig, weil MCI das MCI-Gerät scheinbar automatisch schließt, wenn der aufrufende Thread beendet wird. Wird die `Close`-Methode zuvor nicht aufgerufen, führt der spätere Aufruf des Destruktors über den Garbage Collector zu dem Fehler »Gerät ist nicht geöffnet oder wird vom MCI nicht erkannt«. Dieses Problem konnte ich bisher nicht lösen.

Die `Mci`-Klasse kann durchaus noch erweitert werden. Die folgenden Erweiterungen wären sinnvoll (und machbar):

- ▶ Regelmäßige Benachrichtigung über die aktuelle Position in einem Delegate,
- ▶ Separate Einstellung der Lautstärke für den linken und den rechten Kanal,
- ▶ Ein- und Ausschalten der Soundwiedergabe,
- ▶ Pausieren und Weiter-Ausführen der Wiedergabe,
- ▶ Einstellen der Bass- und der Höhenwiedergabe,
- ▶ Wiedergabe von CD-Audio-Tracks.

Drei bekannte Probleme sind in der Buch-Variante nicht gelöst:

- ▶ Wird eine neue Datei geöffnet, während eine andere abgespielt wird, werden die Dateien zwar parallel abgespielt (was ein interessantes Feature ist, so können Sie z.B. neben einem Video eine MP3-Datei oder zwei MP3-Dateien gleichzeitig abspielen), da aber nur ein Alias verwaltet wird, kann nur die letzte bearbeitet (z.B. geschlossen) werden,
- ▶ MCI funktionierte bei meinen Versuchen nicht in einer Konsolenanwendung. Obwohl kein Fehler erzeugt wurde, spielte MCI Multimedia-Dateien einfach nicht ab,
- ▶ Ist keine Soundkarte installiert, resultiert dies in der wenig aussagekräftigen Ausnahme. Auf meinem Testsystem meldete MCI den Fehler »MMSYSTEM277 Bei der Initialisierung von MCI trat ein Fehler auf. Starten Sie Windows neu«. Sinnvoll wäre es, in der `Open`-Methode zu überprüfen, ob auf dem System eine Soundkarte installiert ist und im negativen Fall eine deutlichere Ausnahme zu werfen. Diese Überprüfung können Sie über das Rezept 153 vornehmen.

216 Multimedia-Dateien (Wave, MP3, Midi, AVI, MPEG etc.) über DirectX abspielen

DirectX bietet einen hervorragenden Multimedia-Support. Schon in DirectX-Versionen vor Version 9 können Sie über eine COM-Komponente auf eine sehr einfache Weise Multimedia- und 3D-Grafikdateien abspielen, verändern, erzeugen etc. Ab Version 9 bietet Microsoft den Zugang zu DirectX-Features sogar über komplett verwaltete Klassen, die sich nahtlos in das .NET Framework integrieren und eine enorme Performance versprechen.

Ein Nachteil der Verwendung von DirectX ist, dass auf den Systemen, auf denen Ihre Anwendung ausgeführt werden soll, dieselbe oder eine höhere Version von DirectX installiert sein muss als die, unter der Sie entwickeln (weswegen ich mir im vorigen Rezept noch die Mühe gemacht habe, wichtige Features von MCI in einer Klasse zu kapseln).

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

In diesem Rezept beschreibe ich die Anwendung der verwalteten Klassen der Version 9 für das Abspielen von Multimedia-Dateien. Um diese zur Verfügung zu haben, müssen Sie das DirectX-9-SDK installieren, das Sie an der Adresse msdn.microsoft.com/downloads/list/directx.asp finden. Der Download des kompletten SDK umfasst etwa 220 MB. Sie können allerdings auch nur die Teile des SDK downloaden, die Sie benötigen. Bei der Installation müssen Sie zwischen der Debug- und der Retail-Version wählen. Die größere Debug-Version (ca. 500 MB auf der Festplatte) enthält zusätzliche Informationen, die beim Debuggen von DirectX-Anwendungen hilfreich sein können.

Hinweis

Das in diesem Rezept beschriebene Abspielen von Multimedia-Dateien ist nur ein kleiner Teil von DirectX. Die anderen Bereiche wie Direct3D, DirectDraw und DirectInput werden in diesem Buch nicht behandelt. Schauen Sie sich dazu die Beispiele an, die Sie über den Eintrag `DIRECTX SAMPLE BROWSER` im Startmenüordner des DirectX-SDK erreichen. Auf der DirectX-Seite von *C# Corner* (www.c-sharpcorner.com/directx.asp) finden Sie einige Beispiele zur Anwendung verschiedener DirectX-Techniken.

Wenn Sie die Installation von Hand vornehmen wollen bzw. müssen, können Sie auf dem Zielrechner einfach nicht nur die DirectX-Laufzeitversion, sondern das SDK installieren. Idealer wäre allerdings, nur die Assemblies zu installieren. Dazu kopieren Sie die DLL-Dateien aus dem Ordner `C:\WINDOWS\Microsoft.NET\Managed DirectX\v4.09.00.0900` (die Versionsnummer kann natürlich differieren) in das Anwendungsverzeichnis. Für das Abspielen von Multimediadateien reichen die Assemblies `Microsoft.DirectX.dll` und `Microsoft.DirectX.AudioVideoPlayback.dll` aus.

ACHTUNG

Zur Ausführung eines Programms, das die verwalteten Klassen von DirectX 9 verwendet, müssen die entsprechenden Assemblies auf dem Rechner vorhanden sein. Diese Assemblies sind im GAC registriert, weswegen Visual Studio die Dateien bei der Erstellung der Anwendung nicht in das Anwendungsverzeichnis kopiert. Ein Visual-Studio-Setup-Projekt erkennt aber die Abhängigkeiten und integriert die notwendigen Dateien in das Setup.

Die Dokumentation der verwalteten DirectX-Klassen ist leider etwas versteckt, wenn Sie kein Visual Studio .NET besitzen (die Dokumentation im Ordner `C:\DXSDK\Doc\DirectX9` bezieht sich auf die Programmierung mit C++). In Visual Studio integriert sich die Dokumentation vollständig in die Hilfe. Leider fehlt die automatische Integration in die .NET Framework-Dokumentation. Sie können die DirectX-Dokumentation aber über die Windows-Befehlszeile aufrufen, indem Sie den folgenden Befehl eingeben:

```
dexplore /helpcol ms-help://MS.DirectX9.1033
```

Zum Abspielen von Audio- und Videodateien referenzieren Sie die `Microsoft.DirectX.AudioVideoPlayback.dll` und importieren den Namensraum `Microsoft.DirectX.AudioVideoPlayback`. Über eine Instanz der Klasse `Audio` können Sie Audiodateien abspielen, über Instanzen

der Klasse `Video` Videodateien. Die Anwendung dieser Klassen ist sehr einfach und nahezu identisch. Ich zeige deshalb nur das Abspielen einer Videodatei.

Beim Erzeugen der `Video`-Instanz übergeben Sie den Dateinamen und können am zweiten Argument bestimmen, ob die Datei sofort abgespielt werden soll. Alternativ können Sie eine Instanz über die statischen Methoden `FromFile` oder `FromUrl` erzeugen, wobei letztere erlaubt, eine Videodatei aus dem Internet zu laden.

Vor dem Abspielen können Sie über die Eigenschaft `Owner` bestimmen, dass das Video auf einem Steuerelement oder Formular abgespielt werden soll. Geben Sie den `Owner` nicht an, wird das Video in einem separaten Fenster abgespielt.

Angenommen, Sie besitzen ein Formular mit einem `PictureBox`-Steuerelement mit Namen `videoBox` und eine MPEG-Datei mit Namen `Tuborg.mpeg` im Ordner der Anwendung, dann können Sie im `Load`-Ereignis des Formulars ein Video folgendermaßen in der `PictureBox` abspielen:

```
private Video video = null;

private void StartForm_Load(object sender, EventArgs e)
{
    // Video-Instanz erzeugen
    string fileName = Path.Combine(Application.StartupPath, "Tuborg.mpeg");
    this.video = new Video(fileName, false);
    // Alternativ über FromFile oder FromUrl
    // this.video = Video.FromUrl(@"http://www.abc.com/videos/tuborg.mpeg");

    // Owner definieren und abspielen
    this.video.Owner = this.videoBox;
    this.video.Play();
}
```

Listing 426: Abspielen eines Videos über DirectX

Dieses Beispiel setzt voraus, dass Sie die Assembly `Microsoft.DirectX.AudioVideoPlayback.dll` referenzieren und die Namensräume `Microsoft.DirectX.AudioVideoPlayback`, `Microsoft.Windows.Forms` und `System.IO` importieren.

Die weitere Arbeit mit den `Audio`- oder `Video`-Instanzen ist über verschiedene Methoden und Eigenschaften sehr einfach. Ich zeige deshalb hier nur exemplarisch die wichtigen Elemente der Klassen:

```
// Abspielen pausieren
this.video.Pause();

// Vollbildmodus einschalten
this.video.Fullscreen = true;
```

Listing 427: Aufruf der wichtigsten Methoden und Eigenschaften der Video-Klasse

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```
// Weiter abspielen
this.video.Play();

// Lautstärke über die dem Video-Objekt zugeordnete Audio-Instanz einstellen
// (die volle Lautstärke entspricht dem Wert Null)
this.video.Audio.Volume = -500;

// Position (in Sekunden) abfragen
double position = this.video.CurrentPosition;

// Zu einer neuen Position relativ von der aktuellen aus wechseln
// (funktionierte in meinen Tests leider nicht: Es wurde immer
// absolut vom Anfang aus positioniert)
this.video.SeekCurrentPosition(10, SeekPositionFlags.RelativePositioning);

// Abspielen stoppen
this.video.Stop();
```

Listing 427: Aufruf der wichtigsten Methoden und Eigenschaften der Video-Klasse (Forts.)

217 Bild in Byte-Array umwandeln

Für einige spezielle Aufgaben wie z.B. das Speichern eines Bildes in einer Datenbank muss das in einem Bitmap-, Icon- oder Metafile-Objekt gespeicherte Bild in ein Byte-Array umgewandelt werden. Die Klassen dieser Objekte besitzen leider keine Methode `CopyTo` o. Ä., die diese Aufgabe erledigt. Die `Save`-Methode der gemeinsamen Basisklasse `Image` ist jedoch in der Lage, die Daten in einen Stream zu schreiben. Wenn Sie am ersten Argument einen `MemoryStream` übergeben, können Sie aus diesem nach dem Aufruf von `Save` ein Byte-Array erzeugen, wie es die Methode `Image2Byte` in Listing 428 zeigt. Am zweiten Argument der `Save`-Methode wird das Format übergeben, in dem die »Datei« geschrieben werden soll (Gif, Jpeg etc.). Um dieses nicht festzulegen erwartet `Image2Byte` das Format am zweiten Argument.

`Image2Byte` benötigt die Referenzierung der Assembly `System.Drawing.dll` und die Einbindung der Namensräume `System`, `System.IO`, `System.Drawing` und `System.Drawing.Imaging`.

```
public static byte[] Image2Byte(Image image, ImageFormat format)
{
    // MemoryStream erzeugen und das Bild in diesen schreiben
    MemoryStream imageStream = new MemoryStream();
    image.Save(imageStream, format);
    imageStream.Flush();

    // MemoryStream in ein Byte-Array schreiben und dieses zurückgeben
    return imageStream.ToArray();
}
```

Listing 428: Methode zum Schreiben eines Image-Objekts in ein Byte-Array

218 Byte-Array in Bitmap umwandeln

Wenn Sie Bilder in einer Datenbank speichern, erhalten Sie beim Lesen ein Byte-Array mit den Daten des Bildes zurück. Aus diesen Bytes muss häufig ein Bitmap-Objekt erzeugt werden, das dann z. B. in einer PictureBox dargestellt werden kann.

Die Bitmap-Klasse (aus dem Namensraum `System.Drawing`) bietet keine direkte Möglichkeit, aus einem Byte-Array ein Bitmap-Objekt zu erzeugen. Dazu müssen Sie, ähnlich wie bei der umgekehrten Umwandlung, den Umweg über ein `MemoryStream`-Objekt gehen.

Die Methode `Byte2Bitmap` in Listing 429 macht genau das. Um das Bitmap-Objekt zu erzeugen, liest `Byte2Bitmap` das Byte-Array in eine neue `MemoryStream`-Instanz ein und übergibt diese an den Konstruktor der Bitmap-Klasse. Um den Stream korrekt schließen zu können (und damit Speicher freizugeben) wird das neue Bitmap-Objekt über eine lokale Referenz verwaltet, die am Ende zurückgegeben wird.

`Byte2Bitmap` erfordert die Referenzierung der Assembly `System.Drawing.dll` und das Importieren der Namensräume `System`, `System.Drawing` und `System.IO`.

```
public static Bitmap Byte2Bitmap(byte[] imageBytes)
{
    // MemoryStream erzeugen, das Bild in diesen schreiben und
    // den Stream zurückgeben
    MemoryStream imageStream = new MemoryStream(imageBytes);
    Bitmap bitmap = new Bitmap(imageStream);
    imageStream.Close();
    return bitmap;
}
```

Listing 429: Methode zum Erzeugen eines Bitmap-Objekts aus einem Byte-Array

219 Bitmap aus der Zwischenablage auslesen

Ein Bild, das in der Zwischenablage gespeichert ist, können Sie auslesen, indem Sie die `GetData`-Methode der `Clipboard`-Klasse (aus dem Namensraum `System.Windows.Forms`) aufrufen und als Typ den Typ der `Bitmap`-Klasse übergeben. Zuvor sollten Sie überprüfen, ob überhaupt ein Bitmap-Bild in der Zwischenablage gespeichert ist, wozu Sie einen Vergleich der Daten auf `null` und die `GetDataPresent`-Methode verwenden können.

Die Methode `GetBitmapFromClipboard` geht auf diese Weise vor um ein in der Zwischenablage gespeichertes Bitmap-Bild auszulesen. Enthält die Zwischenablage keine Daten oder kein Bitmap-Bild, gibt die Methode `null` zurück.

Zum Kompilieren dieser Methode müssen Sie die Assemblies `System.Windows.Forms.dll` und `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Windows.Forms` und `System.Drawing` einbinden.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

public static Bitmap GetBitmapFromClipboard()
{
    // Die Zwischenablagendaten auslesen und überprüfen
    IDataObject clipboardData = Clipboard.GetDataObject();
    if (clipboardData != null)
    {
        // Überprüfen, ob ein Bitmap gespeichert ist
        if (clipboardData.GetDataPresent(typeof(Bitmap)))
        {
            return (Bitmap)clipboardData.GetData(typeof(Bitmap));
        }
    }

    // null zurückgeben, falls kein Bitmap in der Zwischenablage
    // gespeichert ist
    return null;
}

```

Listing 430: Methode zum Auslesen eines Bitmap-Objekts aus der Zwischenablage

220 Screenshot des Bildschirms und eines Formulars erstellen

Einen Screenshot des Bildschirms bzw. eines Formulars können Sie nicht über .NET-Klassen erzeugen. Dazu müssen Sie Windows-API-Funktionen verwenden.

Über die Funktion `CreateDC` können Sie den Device Context (DC) des Bildschirms ermitteln. Ein DC repräsentiert im Windows-API die Zeichenoberfläche eines Fensters, Bitmaps oder eines Geräts wie dem Drucker. Über die Funktion `BitBlt` können Sie die Farbinformationen eines DC in einen anderen kopieren. Als Ziel-DC verwenden Sie dann den DC eines neu erzeugten Bitmap-Objekts.

Die Methode `Screenshot` in Listing 432 setzt `BitBlt` ein. Zum Kompilieren dieser Methode müssen Sie die Assemblies `System.Drawing.dll` und `System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.Drawing`, `System.Windows.Forms` und `System.Runtime.InteropServices` importieren.

Zur Umsetzung sind zunächst einige API-Deklarationen notwendig:

```

[DllImport("gdi32.dll", SetLastError=true)]
private static extern int BitBlt(IntPtr hdcDest, int nXDest, int nYDest,
    int nWidth, int nHeight, IntPtr hdcSrc, int nXSrc, int nYSrc,
    int dwRop);

[DllImport("gdi32.dll", SetLastError=true)]
private static extern IntPtr CreateDC(string lpszDriver, string lpszDevice,
    string lpszOutput, IntPtr lpInitData);

private static int SRCCOPY = 0x00CC0020;

```

Listing 431: Deklaration der benötigten API-Funktionen und Konstanten

Screenshot ermittelt zunächst über CreateDC den DC des Bildschirms. Am ersten Argument wird dazu der String »DISPLAY« übergeben. Mit diesem DC erzeugt Screenshot ein Graphics-Objekt, das beim nachfolgenden Erzeugen eines Bitmap-Objekts am letzten Argument angegeben wird und das die Auflösung des Bilds bestimmt. Das Bitmap-Objekt wird dann in der Größe des Bildschirms erzeugt und ist das Ziel für den späteren BitBlt-Aufruf. Um den DC dieses Objekts zu erhalten, erzeugt Screenshot ein neues Graphics-Objekt und gibt das Bitmap-Objekt am Argument des Konstruktors an. Der DC wird dann über die Methode GetHdc dieses Objekts ausgelesen.

Vor dem Aufruf von BitBlt muss der DC des Bildschirms erneut ermittelt werden, da dieser anscheinend beim Aufruf der FromHdc-Methode der Graphics-Klasse freigegeben wurde (ohne dieses erneute Auslesen schlägt BitBlt ohne Fehlermeldung fehl und die Freigabe des DC führt zu einem Fehler).

Dann wird das Bild über BitBlt kopiert. Am letzten Argument übergibt Screenshot die Konstante SRCCOPY, die dafür sorgt, dass das Quellbild alle Farbinformationen des Ziels überschreibt. Über andere Konstanten, die in der Referenz erläutert werden, können Sie die Farbinformationen des Ziels auch auf verschiedene Weise mit denen der Quelle vermischen, aber das ist für unsere Lösung nicht interessant. Schließlich müssen die DCs noch über die ReleaseHdc-Methode des jeweiligen Graphics-Objekts freigegeben werden, da diese Freigabe nicht automatisch über den Garbage Collector erfolgt.

```
public static Bitmap Screenshot()
{
    // Device Context für den Bildschirm ermitteln und damit ein
    // Graphics-Objekt erzeugen
    IntPtr screenDC = CreateDC("DISPLAY", null, null, (IntPtr)null);
    Graphics screenGraphics = Graphics.FromHdc(screenDC);

    // Bitmap mit den Ausmaßen des Bildschirms und der Auflösung des
    // Graphics-Objekts erzeugen
    Bitmap bitmap = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
        Screen.PrimaryScreen.Bounds.Height, screenGraphics);

    // Zweites Graphics-Objekt aus dem noch leeren Bitmap erzeugen um den
    // DC des Bitmap-Objekts auslesen zu können
    Graphics bitmapGraphics = Graphics.FromImage(bitmap);
    IntPtr bitmapDC = bitmapGraphics.GetHdc();

    // DC des Bildschirms noch einmal ermitteln
    screenDC = screenGraphics.GetHdc();

    // Über BitBlt das über den Bildschirm-DC repräsentierte Bild in das
    // über den Bitmap-DC repräsentierte Bild kopieren
    if (BitBlt(bitmapDC, 0, 0, Screen.PrimaryScreen.Bounds.Width,
        Screen.PrimaryScreen.Bounds.Height, screenDC, 0, 0, SRCCOPY) == 0)
    {
        bitmapGraphics.ReleaseHdc(bitmapDC);
    }
}
```

Listing 432: Methode zum Erzeugen eines Screenshots des Bildschirms

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        screenGraphics.ReleaseHdc(screenDC);
        throw new Exception("API-Fehler " + Marshal.GetLastWin32Error() +
            " beim Aufruf von BitBlt");
    }

    // Die DCs freigeben und das Bild zurückgeben
    bitmapGraphics.ReleaseHdc(bitmapDC);
    screenGraphics.ReleaseHdc(screenDC);
    return bitmap;
}

```

Listing 432: Methode zum Erzeugen eines Screenshots des Bildschirms (Forts.)

Die zweite Variante der Methode `Screenshot`, die am ersten Argument eine Referenz auf ein Formular erwartet, erzeugt einen Screenshot dieses Formulars. Die Arbeitsweise dieser Variante ist nahezu identisch zur ersten. Der Unterschied ist, dass kein DC für den Bildschirm erzeugt wird, sondern der des `Graphics`-Objekts verwendet wird, das die `CreateGraphics`-Methode des Formulars erzeugt. Da sich dieses Objekt auf den Client-Bereich des Formulars bezieht (also den Bereich innen, ohne Titelleiste und Ränder) übergibt `Screenshot` beim Aufruf von `BitBlt` als Breite und Höhe des Bildes die Innenmaße des Formulars.

```

public static Bitmap Screenshot(Form form)
{
    // Graphics-Objekt für das Formular erzeugen und ein neues Bitmap-Objekt
    // und Graphics-Objekt für das Ergebnis erzeugen
    Graphics formGraphics = form.CreateGraphics();
    Bitmap bitmap = new Bitmap(form.ClientRectangle.Width,
        form.ClientRectangle.Height, formGraphics);
    Graphics bitmapGraphics = Graphics.FromImage(bitmap);

    // Die DCs auslesen
    IntPtr formDC = formGraphics.GetHdc();
    IntPtr bitmapDC = bitmapGraphics.GetHdc();

    // Die Grafik des Formulars in das Ziel-Bitmap kopieren
    if (BitBlt(bitmapDC, 0, 0, form.ClientRectangle.Width,
        form.ClientRectangle.Height, formDC, 0, 0, SRCCOPY) == 0)
    {
        formGraphics.ReleaseHdc(formDC);
        bitmapGraphics.ReleaseHdc(bitmapDC);
        throw new Exception("API-Fehler " + Marshal.GetLastWin32Error() +
            " beim Aufruf von BitBlt");
    }

    // DCs freigeben und Bitmap zurückgeben
    formGraphics.ReleaseHdc(formDC);
    bitmapGraphics.ReleaseHdc(bitmapDC);
    return bitmap;
}

```

Listing 433: Methode für die Erzeugung des Screenshots eines Formulars

Abbildung 152 zeigt eine Beispielanwendung, die gerade den Screenshot des Bildschirms in ein PictureBox-Steuerelement geschrieben hat.

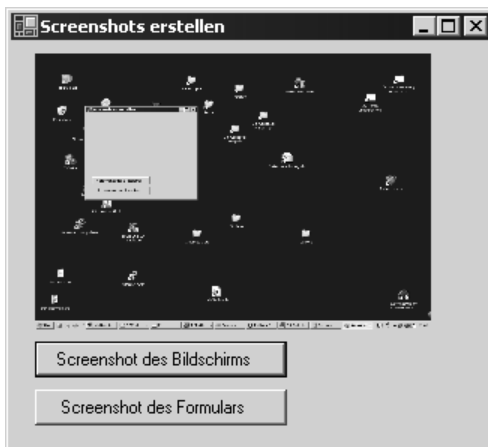


Abbildung 152: Screenshot des aktuellen Bildschirms

221 Bilder skalieren

Bilder können Sie skalieren, indem Sie mit einem Objekt einer von `Image` abgeleiteten Klasse (`Bitmap`, `Icon`, `Metafile`) ein neues Objekt erzeugen. Dazu übergeben Sie das Quell-Objekt am ersten Argument des Konstruktors. Am zweiten Argument übergeben Sie dann die neue Breite und am dritten die neue Höhe. Listing 434 zeigt dies am Beispiel. Eine eingeleseene `Bitmap`-Datei wird auf 50 Prozent der Größe skaliert und wieder unter einem neuen Namen abgespeichert.

```
string fileName = @"c:\\Hitchhiker.bmp";
Bitmap bitmap1 = (Bitmap)Bitmap.FromFile(fileName);

fileName = @"c:\\Hitchhiker_Scaled.bmp";
Bitmap bitmap2 = new Bitmap(bitmap1, (int)(bitmap1.Width * 0.5),
    (int)(bitmap1.Height * 0.5));
bitmap2.Save(fileName);
```

Listing 434: Skalieren eines Bildes auf eine Größe von 50 Prozent

Die Qualität der Skalierung ist größtenteils erstaunlich gut und hängt davon ab, welches Bildformat Sie skalieren, welche Qualität das Bild aufweist und wie stark Sie hoch- oder herunterskalieren. Beim Skalieren wird eine Interpolation der Bildpunkte durchgeführt, welche das ist, konnte ich aber nicht ermitteln, da das Ergebnis des einfachen Skalierens mit keinem der Ergebnisse der möglichen Interpolationen übereinstimmte. Die Interpolation ist nicht immer optimal und kann in vielen Fällen verbessert werden.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Sie können die Interpolation selbst bestimmen, indem Sie das Bild selbst skalieren. Dazu erstellen Sie ein neues `Bitmap`-Objekt in der neuen Größe und erzeugen über `Graphics.FromImage` dazu ein `Graphics`-Objekt. Über dessen `DrawImage`-Methode zeichnen Sie später das originale `Bitmap` auf das neue, wobei Sie das Quell- und das Ziel-Rechteck in Form von `Rectangle`-Objekten angeben. Die Rechtecke entsprechen beim Skalieren dem Rechteck des Quell- bzw. des Ziel-Bitmaps.

Vor dem Zeichnen bestimmen Sie die Interpolation. Dazu stehen Ihnen die folgenden Eigenschaften des `Graphics`-Objekts zur Verfügung:

- ▶ **InterpolationMode:** Bestimmt den Modus der Interpolation mit den Werten der `InterpolationMode`-Aufzählung. Über die Konstante `HighQualityBicubic` erreichen Sie (laut der Hilfe in Photoshop) die beste Qualität. Die bikubische Interpolation ist die langsamste, aber genaueste Methode, über die die gleichmäßigsten Tonabstufungen erzeugt werden. `HighQualityBilinear` führt zu einer etwas verminderten, aber dennoch sehr guten Qualität. Für andere, leider nicht näher dokumentierte Qualitätsstufen stehen Ihnen zudem die Konstanten `Bicubic`, `Bilinear`, `NearestNeighbor`, `Default`, `Invalid`, `Low` und `High` zur Verfügung.
- ▶ **PixelOffsetMode:** Diese leider nur sehr schlecht dokumentierte Eigenschaft gibt den Pixeloffset mit den Werten der `PixelOffsetMode`-Aufzählung an. Mit `HighQuality` erreichen Sie die beste Qualität, aber auch die schlechteste Geschwindigkeit. Daneben stehen die Konstanten `HighSpeed`, `Half`, `None`, `Default` und `Invalid` zur Verfügung.
- ▶ **SmoothingMode:** Bestimmt über die Werte der `SmoothingMode`-Aufzählung, ob und wie auf Linien und Kurven eine Glättung angewendet wird. Über die Konstante `HighQuality` erreichen Sie die beste Qualität. Daneben können Sie die Konstanten `HighSpeed`, `AntiAlias`, `None`, `Default` und `Invalid` verwenden.

Da die Einstellung der genannten Eigenschaften aufgrund der Vielfalt der Möglichkeiten recht schwierig festzulegen ist, habe ich die Methode `ScaleBitmap` in Listing 435 so geschrieben, dass die Werte für die Interpolation übergeben werden. `ScaleBitmap` erzeugt aus dem am ersten Argument übergebenen `Bitmap`-Objekt mit den am zweiten und dritten Argument übergebenen Breiten- und Höhenangaben und den Informationen zur Interpolation ein neues, skaliertes `Bitmap`-Objekt und gibt eine Referenz darauf zurück.

Zum Kompilieren dieser und der folgenden Methode müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Drawing2D` importieren.

```
public static Bitmap ScaleBitmap(Bitmap source, double scaleFactor,
    InterpolationMode interpolationMode, PixelOffsetMode pixelOffsetMode,
    SmoothingMode smoothingMode)
{
    // Bitmap in der neu berechneten Größe erstellen
    int width = (int)(source.Width * scaleFactor);
    int height = (int)(source.Height * scaleFactor);
```

Listing 435: Methode zum Skalieren eines Bildes

```

    Bitmap result = new Bitmap(width, height);

    // Graphics-Objekt für das Bitmap erzeugen und den
    // Interpolier-Modus, Pixeloffset-Modus und den
    // Smoothing-Modus einstellen
    Graphics g = Graphics.FromImage(result);
    g.InterpolationMode = interpolationMode;
    g.PixelOffsetMode = pixelOffsetMode;
    g.SmoothingMode = smoothingMode;

    // Bild von der Quelle auf das Ziel übertragen und dabei skalieren
    g.DrawImage(source, new Rectangle(0, 0, width, height),
        new Rectangle(0, 0, source.Width, source.Height),
        GraphicsUnit.Pixel);
    g.Dispose();

    return result;
}

```

Listing 435: Methode zum Skalieren eines Bildes (Forts.)

Eine zweite Variante dieser Methode erwartet statt der neuen Breite und Höhe einen Skalierungsfaktor (und ruft die erste Variante auf):

```

public static Bitmap ScaleBitmap(Bitmap source, double scaleFactor,
    InterpolationMode interpolationMode, PixelOffsetMode pixelOffsetMode,
    SmoothingMode smoothingMode)
{
    // Neue Breite und Höhe berechnen und damit die erste Variante aufrufen
    int width = (int)(source.Width * scaleFactor);
    int height = (int)(source.Height * scaleFactor);
    return ScaleBitmap(source, width, height, interpolationMode,
        pixelOffsetMode, smoothingMode);
}

```

Listing 436: Zweite Variante zum Skalieren eines Bildes

Bei der Anwendung der Skalierungsmethoden sollten Sie ein wenig mit den verfügbaren Einstellungen experimentieren. Bei meinen Versuchen hat beim Herunterskalieren eines hochqualitativen JPEG-Photos die Interpolation `HighQualityBicubic` das beste Ergebnis erbracht. Das Skalieren über ein neues `Bitmap`-Objekt, das ich am Anfang dieses Rezepts erläutert habe, führte hingegen zu einem deutlich schlechteren Ergebnis (was aber beim Skalieren von GIF- oder `Bitmap`-Bildern nicht unbedingt der Fall war). Die Einstellung des `PixelOffsetMode` und des `SmoothingMode` spielten dabei übrigens kaum eine erkennbare Rolle. Die Größe der erzeugten Datei hängt auch nur wenig von den gewählten Einstellungen ab. Im Test ergab das hochwertige Skalieren eines JPEG-Bildes über `ScaleBitmap` z.B. eine kleinere Größe als beim Skalieren mit einem `Bitmap`-Objekt.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

222 Thumbnails aus Bildern erzeugen

Thumbnails sind verkleinerte Darstellungen eines Bildes in einer meist festgelegten Größe. Thumbnails werden vorwiegend auf Webseiten dargestellt, sodass der Anwender auf das Thumbnail klicken und damit das eigentliche Bild öffnen kann. Aber die Erzeugung von Thumbnails ist natürlich auch für C#-Windowsanwendungen interessant. Eine Idee, die ich demnächst verwirklichen werden (sofern ich zwischen dem Schreiben von Büchern, meinen sonstigen Aufgaben und den ganzen Urlauben noch Zeit finde ☺) ist eine C#-Anwendung, die für alle Bilder eines Ordners eine Photoalbum-Webseite erstellt (Natürlich: Programme dieser Art gibt es schon einige, aber diese machen einfach nicht das, was ich will). Dazu benötige ich auf jeden Fall auch Thumbnails der Bilder.

Thumbnails können Sie über die statische Methode `CreateThumbnail` der `Bitmap`-Klasse erzeugen. Am ersten Argument übergeben Sie die neue Breite, am zweiten die neue Höhe. Das dritte Argument, das einen Delegate erwartet, wird zurzeit in GDI+ nicht verwendet. Entgegen der Dokumentation können Sie hier einfach `null` übergeben. Am vierten Argument, das ebenfalls nicht verwendet wird, übergeben Sie `IntPtr.Zero`.

Das Beispiel in Listing 437 liest eine JPEG-Datei ein und schreibt das Thumbnail dieser Datei in eine `PictureBox`. Das Beispiel benötigt die Referenzierung der Assembly `System.Drawing.dll` und den Import der Namensräume `System` und `System.Drawing`.

```
string fileName = @"C:\\Les Crosets1.jpg";
Bitmap bitmap = new Bitmap(fileName);
this.pictureBox1.Image = bitmap.GetThumbnailImage(this.pictureBox1.Width,
    this.pictureBox1.Height, null, IntPtr.Zero);
```

Listing 437: Erzeugen eines Thumbnails

Die Qualität des Thumbnails ist recht gut, kann aber noch verbessert werden. Dazu können Sie einfach die `ScaleBitmap`-Methode aus dem Rezept 211 verwenden:

```
string fileName = @"C:\\Les Crosets1.jpg";
Bitmap bitmap = new Bitmap(fileName);
this.pictureBox2.Image = ScaleBitmap(bitmap, this.pictureBox2.Width,
    this.pictureBox2.Height, InterpolationMode.HighQualityBicubic,
    PixelOffsetMode.HighQuality, SmoothingMode.HighQuality);
```

Listing 438: Erzeugen eines Thumbnails in einer hohen Qualität

Abbildung 153 zeigt eine Windowsanwendung, bei der zwei hochwertige JPEG-Dateien einmal über `CreateThumbnail` (oben) und dann über `ScaleBitmap` (unten) als Thumbnails dargestellt werden. Die bessere Qualität der unteren Thumbnails können Sie allerdings wahrscheinlich im Buch nicht erkennen.

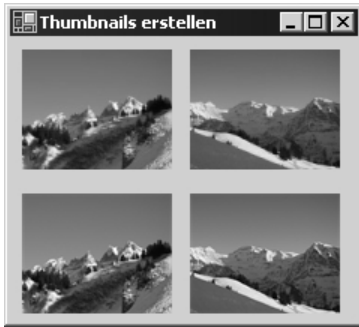


Abbildung 153: Thumbnails von zwei JPEG-Dateien

223 Bilder konvertieren

Das Konvertieren von Bildern ist eigentlich viel zu einfach, als dass dazu ein eigenes Rezept notwendig wäre. Falls Sie aber nicht wissen, wie es geht: Lesen Sie das Bild in ein `Bitmap`-Objekt ein und speichern Sie dieses über die `Save`-Methode ab, indem Sie am ersten Argument den Dateinamen und am zweiten Argument das neue Format über einen Wert der `ImageFormat`-Aufzählung angeben.

Das Beispiel in Listing 439 erfordert, dass Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
// JPEG-Datei in Bitmap einlesen
string fileName = @"C:\Les Crosets.jpg";
Bitmap bitmap = new Bitmap(fileName);

// Als Bitmap abspeichern
fileName = @"C:\Les Crosets.bmp";
bitmap.Save(fileName, ImageFormat.Bmp);
```

Listing 439: Konvertieren eines JPEG-Bildes in ein Bitmap-Bild

Beim Speichern können Sie auch die Qualität des Bildes beeinflussen, sofern GDI+ die dafür notwendigen Codierungsparameter unterstützt. Vergleichen Sie dazu das Rezept 224. Beim Konvertieren müssen Sie darauf achten, dass die Dateigröße entsprechend dem gewählten Format stark variieren kann. Eine Bitmap-Datei ist z.B. wesentlich größer als eine qualitativ gleichwertige JPEG-Datei. Aber das ist ja schon fast Allgemeinwissen 😊.

224 (JPEG-)Bilder mit definierter Qualität speichern

Einige Bildformate wie GIF und JPEG erlauben spezifische Einstellungen, die die Qualität des Bildes beeinflussen. Bei JPEG-Bildern wird die Qualität in Prozent angegeben, GIF ermöglicht Einstellungen wie den Dither-Modus und den Lossy (Verlust). Beim Speichern von `Bitmap`-Objekten können Sie die Qualität des Bildes nur begrenzt und scheinbar nicht für

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

alle Formate beeinflussen. Leider verlässt .NET an dieser Stelle auch die ansonsten so klare Struktur und wird ein wenig undurchsichtig.

Die grundsätzliche Vorgehensweise erläutert die Methode `SaveAsJpeg` in Listing 440, die ein `Bitmap`-Objekt als JPEG-Bild mit einer gegebenen Qualität speichert.

Dazu ermittelt `SaveAsJpeg` zunächst das für JPEG-Bilder in GDI+ vordefinierte `ImageCodecInfo`-Objekt. Ein solches Objekt verwaltet Informationen darüber, wie die zu erzeugende Datei gespeichert werden muss. Das Ermitteln dieses Objekts ist leider ein wenig aufwändig, weil die Methode alle verfügbaren `ImageCodecInfo`-Objekte durchgehen und deren MIME¹-Typ vergleichen muss. Das `ImageCodecInfo`-Objekt wird lediglich deswegen benötigt, weil die Variante der `Save`-Methode, der ein `EncoderParameters`-Objekt übergeben werden kann, nicht die Angabe des Bildformats über ein `ImageFormat`-Argument erlaubt.

Ein `EncoderParameters`-Objekt, das beim Speichern am letzten Argument übergeben wird, nimmt dann einzelne `EncoderParameter`-Objekte auf, die die Codierung der Dateien bestimmen. Ein `EncoderParameters`-Objekt ist eine Liste solcher Parameter. Bei der Erzeugung geben Sie im Konstruktor die Anzahl der zu definierenden Parameter an. Dann können Sie einzelne `EncoderParameter`-Instanzen an die Liste anhängen, die Sie über die Eigenschaft `Param` erreichen. Bei der Erzeugung eines `EncoderParameter`-Objekts geben Sie am ersten Argument ein `Encoder`-Objekt an, das die Codierungsinformationen speichert. Die `Encoder`-Klasse liefert dazu einige statische Eigenschaften, die die einzelnen Codierungs-Möglichkeiten repräsentieren. Dabei tritt das erste Problem auf: Für die einzelnen Formate unterstützt GDI+ nur eine Auswahl der verfügbaren Codierungen. Beim JPEG-Format sind das z.B. die Codierungen `Encoding.Transformation` (Drehen und Spiegeln des Bildes), `Encoding.Quality` (Einstellen der Qualität), `Encoding.LuminanceTable` und `Encoding.ChrominanceTable`. Leider ist nicht dokumentiert, welches Format welche Codierungsparameter unterstützt. In dem Beispiel zu diesem Rezept und im Repository finden Sie aber eine Hilfsmethode `GetAvailableEncoderNames`, die die Namen der für einen MIME-Typ verfügbaren Codierungsparameter auflistet. Eigenartigerweise werden nach dieser Methode für die Formate *Bitmap*, *GIF* und *PNG* keine Codierungsparameter unterstützt. Zur Vervollständigung finden Sie im Repository auch die Hilfsmethode `GetAvailableMimeTypeTypes`, die die von GDI+ unterstützten MIME-Typen auslistet und zurückgibt.

Am zweiten Argument des `EncoderParameter`-Konstruktors übergeben Sie dann den Wert des Parameters. Bei wenigen Parametern wie `Encoding.Quality` sind Zahlwerte erlaubt. Dabei müssen Sie auch ein wenig ausprobieren. Das zweite Argument des Konstruktors erlaubt u.a. `byte`- und `long`-Werte. Übergeben Sie bei einem `Encoding.Quality`-Parameter einen `byte`-Wert (was eigentlich passen würde, denn die JPEG-Qualität geht nur bis 100 Prozent), resultiert unerklärlicherweise eine Ausnahme mit der Meldung »Ungültiger Parameter verwendet«. Die Konvertierung in einen `long`-Wert bringt dann aber die Lösung dieses Problems.

1. Ein MIME-Typ (Multipurpose Internet Mail Extensions), der ursprünglich für das Versenden von spezifischen Daten in E-Mails vorgesehen war, bezeichnet einen Medientyp und wird hauptsächlich im Internet verwendet, damit der Empfänger weiß, wie die Daten ausgewertet werden müssen. Dazu existieren vordefinierte Bezeichnungen wie z.B. `text/html` oder `image/jpeg`.

Andere Parameter wie z.B. `Encoding.Transformation` werden als Wert der `EncoderValue`-Aufzählung übergeben, die alle verfügbaren speziellen Werte auflistet (was die Ermittlung der zu einem Parameter passenden Werte etwas erschwert). Dieser Wert muss in einen `long`-Wert konvertiert werden, da das Argument `value` keinen `EncoderValue`-Typ erlaubt.

Ich habe in `SaveAsJpeg` auf die speziellen Codierungsparameter verzichtet und stelle nur die Qualität ein. Das Drehen und Spiegeln eines `Bitmap`-Objekts, das über den Parameter `Encoding.Transformation` für JPEG-Dateien möglich ist, beschreibt das Rezept 225 unabhängig vom Bildformat.

Mit dem anfangs ermittelten `ImageCodecInfo`-Objekt und den Codierungsparametern wird das Bild dann über die `Save`-Methode des `Bitmap`-Objekts gespeichert.

```
public static void SaveAsJpeg(Bitmap bitmap, string fileName, byte quality)
{
    // Das in GDI+ enthaltene ImageCodecInfo-Objekt für JPEG-Bilder ermitteln
    ImageCodecInfo imageCodec = null;
    ImageCodecInfo[] imageCodecs = ImageCodecInfo.GetImageEncoders();
    for (int i = 0; i < imageCodecs.Length; i++)
    {
        if (imageCodecs[i].MimeType=="image/jpeg")
        {
            imageCodec = imageCodecs[i];
            break;
        }
    }

    if (imageCodec == null)
        throw new Exception("Das ImageCodecInfo-Objekt für den " +
            "Mimetyt image/jpeg kann nicht lokalisiert werden");

    // Den wichtigsten der für JPEG unterstützten Encoding-Parameter definieren
    EncoderParameters encoderParams = new EncoderParameters(1);
    encoderParams.Param[0] = new EncoderParameter(Encoder.Quality,
        (long)quality);

    // Bitmap speichern
    bitmap.Save(fileName, imageCodec, encoderParams);
}
```

Listing 440: Methode zum Speichern eines Bitmap-Objekts als JPEG-Datei mit definierter Qualität

Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System.Drawing` und `System.Drawing.Imaging` importieren.

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Die Anwendung ist dann ganz einfach:

```
// JPEG-Datei in Bitmap einlesen
string fileName = @"C:\Les Crosets.jpg";
Bitmap bitmap = new Bitmap(fileName);

// Als JPEG mit mittlerer Qualität speichern
fileName = @"C:\Les Crosets neu.jpg";
SaveAsJpeg(bitmap, fileName, 50);
```

Listing 441: Anwendung der SaveAsJpeg-Methode

Dabei ist natürlich klar, dass es nichts bringt, wenn Sie Bilder in einer höheren Qualität speichern, als diese im Ursprungszustand besaßen 😊.

225 Bilder drehen und spiegeln

Wenn Sie Bilder in 90-Grad-Schritten drehen oder an der X- und Y-Achse spiegeln wollen, können Sie dazu die `RotateFlip`-Methode eines `Image`-Objekts verwenden. Dieser Methode übergeben Sie einen Wert der Aufzählung `RotateFlipType`, der die Drehung bzw. das Spiegeln bestimmt. Listing 442 zeigt, wie Sie das in einer `PictureBox` verwaltete Bild um 90 Grad nach rechts drehen. Das Beispiel benötigt die Referenzierung der Assembly `System.Drawing.dll` und die Einbindung der Namensräume `System` und `System.Drawing`.

```
Image image = pictureBox1.Image;
image.RotateFlip(RotateFlipType.Rotate90FlipNone);
pictureBox1.Refresh();
```

Listing 442: Drehen eines Bildes um 90 Grad nach rechts

`RotateFlipType` stellt einige Konstanten zur Verfügung:

- ▶ `Rotate90FlipNone`, `Rotate180FlipNone`, `Rotate270FlipNone`: Drehen um 90, 180 bzw. 270 Grad nach rechts ohne zu spiegeln,
- ▶ `RotateNoneFlipX`, `RotateNoneFlipY`, `RotateNoneFlipXY`: Spiegeln um die X-, Y- bzw. die X- und die Y-Achse,
- ▶ Kombinationen aus einer der möglichen Drehungen und einer Spiegelung in der Form `Rotate90FlipX`, `Rotate90FlipY`, `Rotate90FlipXY`, `Rotate180FlipX` etc.

Die Qualität der Drehung bzw. Spiegelung ist sehr gut und verliert auch nach mehreren Drehungen bzw. Spiegelungen nicht an Wert.

Ewas aufwändiger ist das Drehen eines Bildes um andere Werte als 90, 180 und 270 Grad. Dazu können Sie mehrere Wege gehen. Ein Weg, der auch immer wieder in Newsgroups beschrieben wird, ist die Verwendung eines `Graphics`-Objekts und die Transformation der Zeichenmatrix.

Hinweis

Bevor ich diesen Weg beschreibe, gebe ich einen wichtigen Hinweis: Die .NET-Dokumentation der GDI+-Klassen ist teilweise sehr mager ausgefallen und beschreibt nicht wirklich, was einzelne Klassen und Methoden bewirken. Die eigentliche und häufig aussagekräftigere Dokumentation finden Sie im GDI+-Bereich der Plattform-SDK-Dokumentation. Interessante Artikel zu wichtigen Techniken finden Sie unter `GRAPHICS AND MULTIMEDIA SERVICES / GDI+ / USING GDI+ .`

Eine Zeichenmatrix definiert die Matrix, auf die sich Positionsangaben beim Zeichnen beziehen. Normalerweise entspricht die Zeichenmatrix der Zeichenfläche. Sie können die Zeichenmatrix aber auch so transformieren, dass Zeichenausgaben verschoben und verdreht erscheinen (und eine andere Skalierung verwenden, aber das ist hier unwichtig).

Die Vorgehensweise ist in der Methode `RotateImage` in Listing 443 implementiert. Bevor Sie sich diese Methode näher anschauen, muss ich zugeben, dass es eine bessere Lösung gibt. Das Problem meiner Lösung ist, dass die Qualität des Bildes unter der Verdrehung ein wenig leidet, was allerdings erst nach einigen nachfolgenden Drehungen sichtbar wird. Die bessere Lösung finden Sie an der Adresse www.codeproject.com/csharp/rotateimage.asp.

`RotateImage` berechnet zunächst (mit ein wenig Geometrie) die neue Größe des `Bitmap`-Objekts entsprechend dem angegebenen Winkel, wenn am Argument `resizeBitmap true` übergeben wurde. Damit wird erreicht, dass beim Drehen keine Ecken abgeschnitten werden.

Dann erzeugt `RotateImage` mit den neuen Maßen ein `Bitmap`-Objekt und für dieses ein `Graphics`-Objekt. Das `Graphics`-Objekt wird mit der am Argument `fillColor` übergebenen Farbe gefüllt, damit die überstehenden Ränder des neuen `Bitmap` an die Hintergrundfarbe des alten angepasst werden können.

Nun kommt der etwas schwierige (und in der .NET-Dokumentation leider sehr spärlich dokumentierte) Teil: Ein `Graphics`-Objekt verwaltet eine Instanz der Klasse `Matrix`. Dieses Objekt, das Sie über die `Transform`-Eigenschaft erreichen, kann Informationen über auszuführende Transformationen beim Zeichnen verwalten. Die vordefinierte `Matrix`-Instanz definiert natürlich keine Transformationen. Sie können aber über verschiedene Methoden Transformationen der Zeichenkoordinaten in X/Y-Richtung, in einem Winkel oder auf eine andere Skalierung erreichen. Das `Graphics`-Objekt bietet dazu direkt die Methoden `TranslateTransform` und `RotateTransform` an. `TranslateTransform` bewirkt eine Transformation der Zeichenmatrix in X/Y-Richtung, `RotateTransform` eine Drehung um einen bestimmten Winkel mit der oberen linken Ecke als Mittelpunkt. Die Zeichenmatrix wird dabei allerdings nicht wirklich verschoben und verdreht: Die Transformationen werden für *jede* nachfolgende Zeichenoperation separat angewendet, indem die Zeichenkoordinaten entsprechend transformiert werden.

Das zweite, leider ebenfalls sehr spärlich dokumentierte Argument der Transformations-Methoden bestimmt, wann welche Transformation angewendet wird. Mit dem Wert `MatrixOrder.Prepend` wird die Transformation vor der vorherigen angewendet, mit dem Wert `MatrixOrder.Append` nach der vorherigen. Dieses Argument ist zur korrekten Anwendung der Transformation sehr wichtig, da es schon ein Unterschied ist, ob die Zeichenmatrix erst verschoben und danach gedreht oder erst gedreht und dann verschoben wird. Sie werden jetzt

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

vielleicht denken: Was soll's – wenn ich erst drehen und dann verschieben will, rufe ich `RotateTransform` einfach vor `TranslateTransform` auf und gebe am zweiten Argument `MatrixOrder.Append` an (das habe ich auf jeden Fall zuerst gedacht). Wenn Sie aber z. B. *mehrfach* mit *derselben* Verschiebung verdrehen wollen, müssen Sie `TranslateTransform` vor `RotateTransform` aufrufen und bei `RotateTransform` am zweiten Argument `MatrixOrder.Prepend` angeben. Dabei sollten Sie bedenken, dass weitere Aufrufe der Transformationsmethoden die Matrix immer weiter verschieben bzw. verdrehen.

Ein wesentlicher Nachteil der `RotateTransform`-Methode für unsere Zwecke ist, dass diese um die obere linke Ecke dreht. Die zur korrekten Ausrichtung des gedrehten Bildes notwendigen Berechnungen waren mir zu komplex. Deshalb bin ich einen anderen Weg gegangen. Sie können das `Matrix`-Objekt nämlich über dessen eigene Methoden wesentlich flexibler direkt einstellen. Die Methode `Translate` bewirkt eine Verschiebung, der Methode `RotateAt` können Sie zum Drehen neben dem Drehwinkel auch die Koordinaten des Drehpunkts übergeben. `RotateImage` initialisiert nun ein neues `Matrix`-Objekt so, dass die Koordinaten beim Zeichnen nach rechts und unten verschoben und um den Mittelpunkt verdreht werden. Das gezeichnete Bild erscheint somit verdreht in der Mitte der Zeichenfläche.

Dabei ist es leider, aus mir unklaren Gründen, nicht möglich, das `Matrix`-Objekt des `Graphics`-Objekts über dessen `Transform`-Eigenschaft direkt anzusprechen (ein Versuch führte zu keinerlei Transformation, aber auch zu keinem Fehler). Sie müssen ein neues `Matrix`-Objekt erzeugen und dieses der `Transform`-Eigenschaft zuweisen, wie es `RotateImage` zeigt.

Vor dem Zeichnen des Quellbilds wird die Ausgabequalität so definiert, dass diese möglichst gut ist. Vergleichen Sie dazu das Rezept 221. Ich habe in `RotateImage` darauf verzichtet, die hierbei verwendeten Werte als Argumente zu übergeben, aber das können Sie ja bei Bedarf nachholen.

Schließlich zeichnet `RotateImage` das Quellbild auf das Ziel-Bitmap, gibt die Ressourcen des `Graphics`-Objekts frei und gibt eine Referenz auf das neue `Bitmap`-Objekt zurück.

Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Drawing2D` importieren.

```
public static Bitmap RotateImage(Image image, float angle, Color fillColor,
    bool resizeBitmap)
{
    // Neue Breite und Höhe berechnen
    int newHeight, newWidth;
    if (resizeBitmap)
    {
        // Berechnung des Umfassungsrechtecks
        int x = image.Width / 2;
        int y = image.Height / 2;
        double cosTheta = Math.Cos(2 * Math.PI * angle / 360);
        double sinTheta = Math.Sin(2 * Math.PI * angle / 360);
        double a = Math.Max(Math.Abs(x * cosTheta + y * sinTheta),
```

Listing 443: Methode zum Drehen eines Bildes (allerdings mit leichten Qualitätsverlusten)

```

        Math.Abs(x * cosTheta - y * sinTheta));
    double b = Math.Max(Math.Abs(x * sinTheta - y * cosTheta),
        Math.Abs(x * sinTheta + y * cosTheta));
    newWidth = (int)Math.Round(2 * a);
    newHeight = (int)Math.Round(2 * b);
}
else
{
    newHeight = image.Height;
    newWidth = image.Width;
}

// Neues Bitmap-Objekt mit den vergrößerten Ausmaßen des alten erzeugen
Bitmap bitmap = new Bitmap(newWidth, newHeight);

// Graphics-Objekt für das Bitmap erzeugen und mit der übergebenen
// Farbe füllen
Graphics g = Graphics.FromImage(bitmap);
g.Clear(fillColor);

// Neue Transformationsmatrix erzeugen
Matrix matrix = new Matrix();

// Die Transformation so einstellen, dass die Ausgabe des Bildes
// in der Mitte der Zeichenfläche erscheint und dass diese um den
// Mittelpunkt um den angegebenen Winkel gedreht wird
int xOffset = (int)((newWidth - image.Width) / 2);
int yOffset = (int)((newHeight - image.Height) / 2);
matrix.Translate(xOffset, yOffset);
Point rotatePoint = new Point(image.Width / 2, image.Height / 2);
matrix.RotateAt(angle, rotatePoint, MatrixOrder.Prepend);
g.Transform = matrix;

// Die Zeichenqualität einstellen
g.InterpolationMode = InterpolationMode.HighQualityBicubic;
g.PixelOffsetMode = PixelOffsetMode.HighQuality;
g.SmoothingMode = SmoothingMode.HighQuality;

// Das Bild entsprechend der Transformation verschoben und verdreht
// ausgeben
g.DrawImage(image, 0, 0, image.Width, image.Height );

// Das Bild zurückgeben
return bitmap;
}

```

Listing 443: Methode zum Drehen eines Bildes (allerdings mit leichten Qualitätsverlusten) (Forts.)

Abbildung 154 zeigt eine Anwendung, die ein Bild über `RotateImage` um 33 Grad gedreht hat. Um die Drehung und die neue Größe des Bitmap zu verdeutlichen, wurde als Füllfarbe `Color.Silver` übergeben.

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

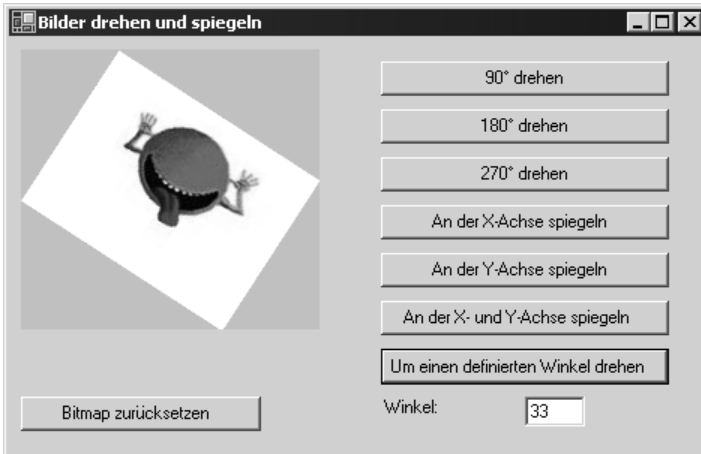


Abbildung 154: Beispielanwendung zum Drehen und Spiegeln eines Bildes mit einem um 33 Grad gedrehten Bild

226 Bildausschnitte auslesen

Wenn Sie beispielsweise ein Verschiebepuzzle programmieren wollen, ist es notwendig, ein Bild in mehrere Teile zu zerschneiden. Um dies zu realisieren, können Sie den Weg gehen, der in der Methode `CropImage` in Listing 444 implementiert ist.

`CropImage` erwartet ein `Image`-Objekt und die Position und Größe des Ausschnitts. Als Erstes erzeugt die Methode ein neues `Bitmap`-Objekt mit den gegebenen Ausmaßen und ein `Graphics`-Objekt für das Zeichnen auf dem Bild. Vor dem Schreiben des Ausschnitts wird die Ausgabequalität so definiert, dass diese möglichst ideal ist. Vergleichen Sie dazu das Rezept 221. Über die `DrawImage`-Methode des `Graphics`-Objekts schreibt `CropImage` schließlich den Ausschnitt aus dem Quellbild in das Ziel-`Bitmap`.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing`, `System.Drawing.Imaging` und `System.Drawing.Drawing2D` importieren.

```
public static Bitmap CropImage(Image image, int x, int y, int width,
    int height)
{
    // Neues Bitmap-Objekt mit den gegebenen Ausmaßen und dafür ein
    // Graphics-Objekt erzeugen
    Bitmap croppedBitmap = new Bitmap(width, height);
    Graphics g = Graphics.FromImage(croppedBitmap);

    // Die Zeichenqualität einstellen
    g.InterpolationMode = InterpolationMode.HighQualityBicubic;
    g.PixelOffsetMode = PixelOffsetMode.HighQuality;
    g.SmoothingMode = SmoothingMode.HighQuality;
```

Listing 444: Methode zum Zuschneiden eines Bildes

```

// Den Ausschnitt des Quellbilds auf das Ziel-Bild kopieren
Rectangle destRect = new Rectangle(0, 0, width, height);
Rectangle sourceRect = new Rectangle(x, y, width, height);
g.DrawImage(image, destRect, sourceRect, GraphicsUnit.Pixel);
g.Dispose();

// Ergebnis zurückgeben
return croppedBitmap;
}

```

Listing 444: Methode zum Zuschneiden eines Bildes (Forts.)

Abbildung 155 zeigt eine Anwendung, die den linken oberen Teil eines Bildes über `CropImage` in einzelne Teile zerschnitten und diese in einzelne `PictureBox`-Steuerelemente geschrieben hat.

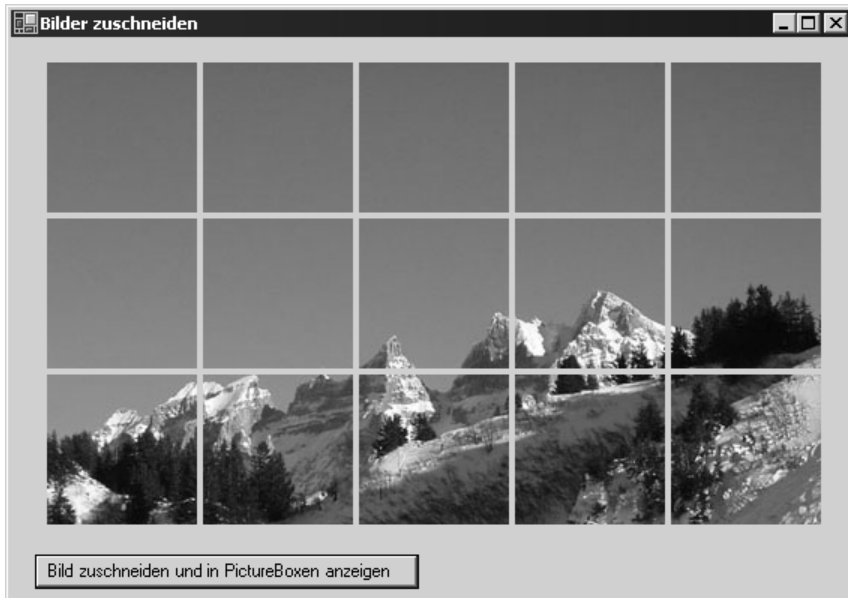


Abbildung 155: Ein Bild wurde in mehrere Teile zerschnitten

227 Farbinformationen von Bildern gezielt verändern

Das .NET Framework bietet mit den Klassen `ColorMap` und `ColorMatrix` hervorragende Möglichkeiten, die Farbinformationen eines Bilds gezielt zu verändern. Über ein `ColorMap`-Objekt können Sie einzelne Farben in andere umwandeln, eine `ColorMatrix`-Instanz ermöglicht die Transformation der ARGB-Werte der einzelnen Pixel.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Beim Nachvollziehen dieses Rezepts werden Sie wahrscheinlich feststellen, dass die .NET-Dokumentation der GDI+-Klassen teilweise sehr mager ausgefallen ist. Wenn Sie sich einmal die Dokumentation der Klasse `ColorMatrix` anschauen, werden Sie erkennen, was ich meine: Die Dokumentation sagt zwar etwas über die Bedeutung aus, aber verschweigt die Arbeitsweise der Farbmatrix. Die eigentliche und etwas aussagekräftigere Dokumentation finden Sie im GDI+-Bereich der Plattform-SDK-Dokumentation. Unter `GRAPHICS AND MULTIMEDIA SERVICES / GDI+ / USING GDI+` finden Sie viele interessante Artikel, u.a. auch einem zur Verwendung einer Farbmatrix (der aber leider die eigentliche Transformation nicht näher erläutert).

ColorMap

Zur Umwandlung einzelner Farbwerte eines Bildes in andere können Sie ein `ColorMap`-Array verwenden. Die darin verwalteten `ColorMap`-Objekte speichern in ihrer Eigenschaft `OldColor` die alte und in `NewColor` die neue Farbe. Da es sich um ein Array handelt, können Sie beliebig viele Farbumwandlungen definieren.

Dieses `ColorMap`-Array übergeben Sie dann über die `SetRemapTable`-Methode an ein neues `ImageAttributes`-Objekt. Ein solches Objekt verwaltet Informationen über die Veränderung der Bildfarben während der Ausgabe. Neben speziellen Einstellungen wie dem Gammawert können Sie dem `ImageAttributes`-Objekt eben auch ein `ColorMap`-Objekt für Farbumwandlungen übergeben.

Dieses Objekt können Sie dann der `DrawImage`-Methode eines `Graphics`-Objekts am Argument `imageAttrs` übergeben oder beim Erzeugen eines `TextureBrush`-Objekts (für das Zeichnen mit einem Textur-Pinsel) dem Konstruktor übergeben.

Listing 445 zeigt die Vorgehensweise. Das Beispiel liest das in der `PictureBox` `pictureBox1` gespeicherte Bild aus, zeichnet dieses auf ein neues `Bitmap`-Objekt und schreibt das Bild in die `PictureBox` `pictureBox2`. Für die Farbumwandlung wird die Farbe des ersten Pixels ausgelesen und als alte Farbe eines `ColorMap`-Objekts gesetzt. Als neue Farbe wird `Color.Silver` eingesetzt.

Zum Kompilieren dieses Quellcodes benötigen Sie ein Formular mit zwei `PictureBox`-Steuerelementen `pictureBox1` und `pictureBox2`. Außerdem müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
// Bitmap aus der ersten PictureBox auslesen
Bitmap sourceBitmap = (Bitmap)this.pictureBox1.Image;

// Neues Bitmap mit den Ausmaßen des Originals erzeugen
Bitmap destBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

// ColorMap-Array für die Transformation der Farben erzeugen
ColorMap[] map = new ColorMap[1];
```

Listing 445: Ändern einer Hintergrundfarbe eines Bildes

```

map[0] = new ColorMap();
map[0].OldColor = sourceBitmap.GetPixel(0,0);
map[0].NewColor = Color.Silver;

// Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
// ImageAttributes-Objekt mit dem ColorMap-Objekt übergeben
ImageAttributes imageAttributes = new ImageAttributes();
imageAttributes.SetRemapTable(map);
Graphics g = Graphics.FromImage(destBitmap);
g.DrawImage(sourceBitmap, new Rectangle(0, 0, sourceBitmap.Width,
    sourceBitmap.Height), 0, 0, sourceBitmap.Width, sourceBitmap.Height,
    GraphicsUnit.Pixel, imageAttributes);
g.Dispose();

// Neues Bitmap-Objekt in der zweiten PictureBox ablegen
pictureBox2.Image = destBitmap;

```

Listing 445: Ändern einer Hintergrundfarbe eines Bildes (Forts.)

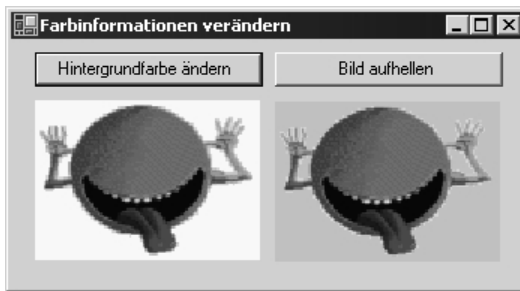


Abbildung 156: Ändern von Farbwerten eines Bilds

Ein kleines Problem dieser Technik ist, dass immer nur die Farben ersetzt werden, die genau den Farben im `ColorMap`-Array entsprechen. Ein Ersetzen von annähernd gleichen Farben (z.B. allen Rottönen) ist damit nicht bzw. nur mit einer Vielzahl an Definitionen im `ColorMap`-Array möglich.

ColorMatrix

Um die Farben der einzelnen Pixel eines Bilds gezielt zu transformieren, verwenden Sie ein `ColorMatrix`-Objekt. Ein solches verwaltet eine 5*5-Matrix in Form eines verzweigten `float`-Array (`float[][][]`). Die Elemente 0 bis 2 der beiden Dimensionen speichern Transformationswerte für den Rot-, Grün- und Blau-Wert, das vierte Element einen Transformationswert für den Alphawert (der die Transparenz bestimmt). Über die fünfte Zeile können Sie zu den RGB-Farbwerten und zum Alpha-Wert Werte hinzuaddieren. Das Element in der Zelle 4,4 muss laut der Dokumentation immer 1 sein. Die fünfte Spalte wird nicht verwendet und muss laut der Dokumentation (bis auf den Wert in der Zelle 4,4) den Wert 0 speichern.

Eine Farbmatrix, die keinerlei Farbinformationen verändert, sieht z.B. so aus wie in Abbildung 157.

	R	G	B	A
R	1	0	0	0
G	0	1	0	0
B	0	0	1	0
A	0	0	0	1

Abbildung 157: Farb-Transformationsmatrix, die keine Farbänderungen bewirkt

Bei einer Farbtransformation werden der Rot-, Grün-, Blau- und der Alpha-Wert jedes Pixels entsprechend der Farbmatrix umgewandelt. Die Transformation ist einfach, aber etwas schwer zu erläutern (weswegen die .NET- und die GDI+-Dokumentation wahrscheinlich auch darauf verzichten). Ich musste recht viel im Internet recherchieren und ein Testprogramm schreiben, um die Transformation zu verstehen. Das Testprogramm ist übrigens ein Bestandteil des Beispiels zu diesem Rezept.

Die Spalten stehen für die Ergebnis-Farbwerte. Die erste Spalte ergibt nach der Transformation den Rotwert, die zweite den Grünwert, die dritte den Blauwert und die vierte den Alphawert. Die Zeilen stehen für die Farbanteile des zu transformierenden Pixels. Der jeweilige Rot-, Grün, Blau- und Alphawert wird mit dem in der zugehörigen Zeile angegebenen Faktor multipliziert. Die Ergebnisse der einzelnen Multiplikationen werden addiert, wobei die mit 255 multiplizierten Werte der letzten Zeile noch auf das Ergebnis aufaddiert werden (weswegen auch nicht-lineare Transformationen möglich sind). Die aus den einzelnen Spalten resultierenden RGBA-Werte werden dann wieder zu einer Farbe zusammengesetzt und der Pixel damit ausgegeben.

Wahrscheinlich drücken Formeln das Ganze etwas klarer aus. In den Formeln steht Z_0 für die erste Zeile, Z_1 für die zweite etc., S_0 für die erste Spalte, S_1 für die zweite etc.

$$\text{Rot} = \text{Rot} * (S_0, Z_0) + \text{Grün} * (S_0, Z_1) + \text{Blau} * (S_0, Z_2) + \text{Alpha} * (S_0, Z_3) + (S_0, Z_4) * 255$$

$$\text{Blau} = \text{Rot} * (S_1, Z_0) + \text{Grün} * (S_1, Z_1) + \text{Blau} * (S_1, Z_2) + \text{Alpha} * (S_1, Z_3) + (S_1, Z_4) * 255$$

$$\text{Grün} = \text{Rot} * (S_2, Z_0) + \text{Grün} * (S_2, Z_1) + \text{Blau} * (S_2, Z_2) + \text{Alpha} * (S_2, Z_3) + (S_2, Z_4) * 255$$

$$\text{Alpha} = \text{Rot} * (S_3, Z_0) + \text{Grün} * (S_3, Z_1) + \text{Blau} * (S_3, Z_2) + \text{Alpha} * (S_3, Z_3) + (S_3, Z_4) * 255$$

Das Ergebnis wird bei der Berechnung auf den Byte-Bereich begrenzt, es kommen also nie Werte kleiner 0 oder größer 255 heraus.

Über diese Matrix können Sie nun alle denkbaren Farbtransformationen ausführen. So können Sie den Grünanteil eines Bildes beispielsweise auf 50 Prozent verringern und 80 Prozent des Grünanteils auf den Blauanteil aufaddieren (Abbildung 158).

Abbildung 159 zeigt eine Matrix, die ein Bild um 20 Prozent aufhellt.

Ich verzichte aber an dieser Stelle auf eine weitere Erläuterung der Möglichkeiten und überlasse dies Ihrer Phantasie. Einige Transformationen wie z. B. die Umwandlung eines Bildes in ein Graustufen-Bild setze ich aber in den folgenden Rezepten ein. Hier geht es nun darum, wie Sie die Transformation anwenden. Und das ist recht einfach.

	R	G	B	A	
R	1	0	0	0	0
G	0	0,5	0,8	0	0
B	0	0	1	0	0
A	0	0	0	1	0
	0	0	0	0	1

Abbildung 158: Farbmatrix zum Verringern des Grünanteils auf 50 Prozent und zum Aufaddieren von 80 Prozent des Grünanteils auf den Blauanteil

	R	G	B	A	
R	1	0	0	0	0
G	0	1	0	0	0
B	0	0	1	0	0
A	0	0	0	1	0
	0,2	0,2	0,2	0	1

Abbildung 159: Farbmatrix zum Aufhellen eines Bildes um 20 Prozent

Farbtransformationen können Sie beim Zeichnen über die `DrawImage`-Methode eines `Graphics`-Objekts anwenden, indem Sie wie schon beim `ColorMap`-Array beschrieben am Argument `imageAttr` ein `ImageAttributes`-Objekt übergeben. Eine andere Möglichkeit ist das normale Zeichnen mit einem `TextureBrush`-Pinsel, den Sie bei der Erzeugung mit einem `ImageAttributes`-Objekt initialisieren können (so können Sie z.B. transparente Texturen erzeugen).

Diesem Objekt weisen Sie zuvor über die `SetColorMatrix`-Methode eine `ColorMatrix`-Instanz zu. Und diese müssen Sie lediglich mit einer passenden Matrix initialisieren. Listing 446 zeigt dies am Beispiel eines Bildes, das in einer `PictureBox` gespeichert ist. Das Bild der linken `PictureBox` wird in diesem Beispiel über eine `ColorMatrix`-Instanz aufgehellt und in die rechte `PictureBox` geschrieben.

Das Beispiel benötigt ein Formular mit zwei `PictureBox`-Steuerelementen `pictureBox1` und `pictureBox2`. Daneben müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
// Bitmap aus der ersten PictureBox auslesen
Bitmap sourceBitmap = (Bitmap)this.pictureBox1.Image;

// Neues Bitmap mit den Ausmaßen des Originals erzeugen
Bitmap destBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

// ColorMatrix für die Transformation der Farben erzeugen
ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
    new float[] { 1, 0, 0, 0, 0 },
```

Listing 446: Aufhellen eines Bildes über ein `ColorMatrix`-Objekt

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benut

Multi-
mediaAutom-
tionSons-
tiges

```

    new float[] {0, 1, 0, 0, 0},
    new float[] {0, 0, 1, 0, 0},
    new float[] {0, 0, 0, 1, 0},
    new float[] {0.3F, 0.3F, 0.3F, 0, 1}
});

// Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
// ImageAttributes-Objekt mit der ColorMatrix übergeben
ImageAttributes imageAttributes = new ImageAttributes();
imageAttributes.SetColorMatrix(colorMatrix);
Graphics g = Graphics.FromImage(destBitmap);
g.DrawImage(sourceBitmap, new Rectangle(0, 0, sourceBitmap.Width,
    sourceBitmap.Height), 0, 0, sourceBitmap.Width, sourceBitmap.Height,
    GraphicsUnit.Pixel, imageAttributes);
g.Dispose();

// Neues Bitmap-Objekt in der zweiten PictureBox ablegen
pictureBox2.Image = destBitmap;

```

Listing 446: Aufhellen eines Bildes über ein ColorMatrix-Objekt (Forts.)

Das ColorMatrix-Objekt muss übrigens nicht unbedingt so initialisiert werden wie in Listing 446. Sie können das Objekt auch über den Default-Konstruktor erzeugen und die Matrix über Eigenschaften definieren, deren Name mit *Matrix* beginnt und mit den Indizes der anzusprechenden Zelle endet. Die Eigenschaft `Matrix01` spricht z.B. die Zelle in der ersten Zeile der zweiten Spalte an. Da der Defaultkonstruktor die Matrix so initialisiert, dass keine Farbänderungen vorgenommen werden, müssen Sie nur die Zellen ändern, die mit einem anderen als dem Defaultwert definiert werden sollen. Die Matrix zum Aufhellen eines Bildes kann z. B. auch so definiert werden:

```

ColorMatrix colorMatrix = new ColorMatrix();
colorMatrix.Matrix40 = 0.3F;
colorMatrix.Matrix41 = 0.3F;
colorMatrix.Matrix42 = 0.3F;

```

Abbildung 160 zeigt eine Anwendung, die auf eine ähnliche Weise wie in Listing 446 das Bild in der linken PictureBox um 30 Prozent aufgehellt in die rechte PictureBox geschrieben hat.

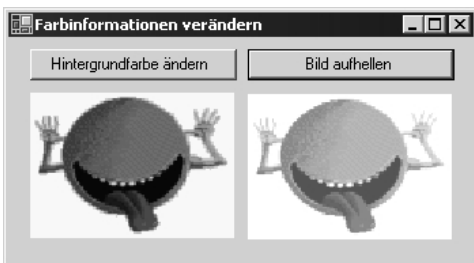


Abbildung 160: Aufhellen eines Bildes

228 Ein Negativ eines Bildes erzeugen

Zur Erzeugung eines Bildnegativs können Sie ein `ColorMatrix`-Objekt (aus dem Namensraum `System.Drawing.Imaging`) verwenden. Die Verwendung dieser Klasse wird im Rezept 227 beschrieben. Zur Erzeugung eines Negativs verwenden Sie eine Farbmatrix, bei der die einzelnen Farben über eine Multiplikation mit `-1` umgekehrt werden.

Die Methode `CreateNegative1` in Listing 447 geht auf diese Weise vor. Sie erzeugt ein neues `Bitmap`-Objekt und zeichnet das Bild des übergebenen `Image`-Objekts darauf. Dabei wird eine `ColorMatrix`-Instanz verwendet, die die Farben des Bildes umkehrt.

Zum Kompilieren dieser Methode müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
public static Bitmap CreateNegative1(Image image)
{
    // Neues Bitmap mit den Ausmaßen des originalen erzeugen
    Bitmap bitmap = new Bitmap(image.Width, image.Height);

    // ColorMatrix für die Transformation der Farben erzeugen
    ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
        new float[] { -1, 0, 0, 0, 0 },
        new float[] { 0, -1, 0, 0, 0 },
        new float[] { 0, 0, -1, 0, 0 },
        new float[] { 0, 0, 0, 1, 0 },
        new float[] { 0, 0, 0, 0, 1 }
    });

    // Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
    // ImageAttributes-Objekt mit der ColorMatrix übergeben
    ImageAttributes imageAttributes = new ImageAttributes();
    imageAttributes.SetColorMatrix(colorMatrix);
    Graphics g = Graphics.FromImage(bitmap);
    g.DrawImage(image, new Rectangle(0, 0, image.Width, image.Height),
        0, 0, image.Width, image.Height, GraphicsUnit.Pixel,
        imageAttributes);
    g.Dispose();

    return bitmap;
}
```

Listing 447: Methode zur Erzeugung eines Bildnegativs

Die in diesem Rezept verwendete Technik ist allerdings nicht ganz perfekt: Rein schwarze Pixel (mit dem Farbwert 0) werden über die Multiplikation mit `-1` nicht umgesetzt. Eigentlich müssten dabei rein weiße Pixel herauskommen, aber für die Berechnung wäre eine Fallunterscheidung notwendig. Um diese umsetzen zu können, können Sie die einzelnen Pixel eines `Bitmap`-Objekts lesen, umrechnen und wieder zurückschreiben, wie es das Rezept 229 zeigt.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

229 Die einzelnen Pixel eines Bildes bearbeiten

Über die `GetPixel`-Methode eines `Bitmap`-Objekts können Sie die einzelnen Pixel des Bildes auslesen und (nach einer eventuellen Umrechnung) über `SetPixel` wieder setzen. Die Methode `CreateNegative2` in Listing 448 zeigt, wie Sie damit umgehen. Diese Methode, die die Referenzierung der Assembly `System.Drawing.dll` und den Import der Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` erfordert, erzeugt ein echtes Negativ eines Bildes.

Der `unchecked`-Block ist in dieser Methode übrigens deswegen enthalten, weil die Multiplikation eines Bytewerts mit `-1` in einen Bytewert konvertiert einen Überlauf ergibt. Ist in Ihrem Programm die Überlaufprüfung eingeschaltet, würde die Methode ohne `unchecked`-Block mit einer Ausnahme fehlschlagen.

```
public static Bitmap CreateNegative2(Bitmap bitmap)
{
    // Neues Bitmap aus dem übergebenen erzeugen
    Bitmap newBitmap = (Bitmap)bitmap.Clone();

    // Die einzelnen Pixel durchgehen und neu berechnen
    for (int x = 0; x < newBitmap.Width; x++)
    {
        for (int y = 0; y < newBitmap.Height; y++)
        {
            Color color = newBitmap.GetPixel(x, y);
            if (color == Color.Black)
                newBitmap.SetPixel(x, y, Color.White);
            else
            {
                unchecked
                {
                    newBitmap.SetPixel(x, y, Color.FromArgb((byte)(color.R * -1),
                                                                (byte)(color.G * -1), (byte)(color.B * -1)));
                }
            }
        }
    }

    return newBitmap;
}
```

Listing 448: Methode zum Erzeugen eines Negativs

Die Methode `CreateNegative2` hat allerdings auch einen Nachteil: Sie ist sehr langsam, weil zum Lesen und Schreiben der einzelnen Pixel Methoden aufgerufen werden. Eine Lösung dieses Problems wäre die Verwendung eines Zeigers zum Durchgehen der Pixel, wie es Eric Gunnerson im Artikel »Unsafe Image Processing« beschreibt. Diesen Artikel finden Sie an der Adresse msdn.microsoft.com/library/en-us/dnscsol/html/csharp11152001.asp.

230 Farb-Bilder in Graustufen-Bilder umwandeln

Für die Umwandlung von Bildern in Graustufen-Bitmaps bietet das .NET Framework keinen direkten Support. Sie können die Umwandlung aber relativ einfach selbst programmieren.

Die Grundidee einer einfachen Umwandlung in ein Graustufen-Bild ist, einfach den Mittelwert der RGB-Werte jedes Pixels des Bilds zu berechnen und als Rot-, Grün- und Blauanteil zur Ermittlung eines neuen RGB-Werts zu verwenden. Eine bessere Methode, die unter dem Begriff *Luminanz* bekannt ist, nutzt die Tatsache, dass das menschliche Auge für die einzelnen Grundfarben unterschiedlich empfindlich ist. Diese Methode ermittelt den RGB-Grauwert über die folgende Formel:

$$Y = 0,3 * Rot + 0,59 * Grün + 0,11 * Blau;$$

Das entstehende Graustufenbild entspricht im Kontrast wesentlich mehr dem Originalbild als bei der einfachen Mittelwert-Berechnung.

Um die Farben der einzelnen Pixel neu zu berechnen, können Sie jedes einzelne Pixel auslesen, neu berechnen und wieder zurückschreiben, wie ich es im Rezept 229 beschreibe. Diese Methode, die auch als Lösung im Internet gehandelt wird, ist jedoch recht langsam (auf jeden Fall wenn Sie dazu reine .NET-Features verwenden). Vergleichen Sie dazu den Artikel »Unsafe image processing« von Eric Gunnerson an der Adresse msdn.microsoft.com/library/en-us/dnscsol/html/csharp11152001.asp. Eric Gunnerson löst das Geschwindigkeitsproblem hier übrigens über einen guten alten Zeiger.

Sie können das Problem aber auch mit .NET-Mitteln lösen ohne die einzelnen Pixel selbst zu transformieren. Die dazu verwendete Technik beschreibt das Rezept 227.

Die Methode `CreateGrayscaledBitmap` in Listing 444 nutzt diese Technik. Sie erzeugt zunächst ein neues `Bitmap`-Objekt mit den Ausmaßen des übergebenen Bilds und initialisiert ein neues `ColorMatrix`-Objekt mit der Matrix für eine Graustufenumwandlung nach der Luminanz-Methode. Dieses Objekt übergibt `CreateGrayscaledBitmap` dann an ein neues `ImageAttributes`-Objekt. Das übergebene Bild wird schließlich über ein `Graphics`-Objekt in das erzeugte `Bitmap` geschrieben, wobei am letzten Argument das `ImageAttributes`-Objekt übergeben wird. Beim Übertragen der Pixel wird damit automatisch die im `ColorMatrix`-Objekt definierte Transformation angewendet.

Zum Kompilieren der `CreateGrayscaledBitmap`-Methode müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

Basics

Datum
ZeitAnwer-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

public static Bitmap CreateGrayscaledBitmap(Image image)
{
    // Neues Bitmap-Objekt mit den Ausmaßen der Quelle erzeugen
    Bitmap bitmap = new Bitmap(image.Width, image.Height);

    // ColorMatrix für die Transformation der Farben erzeugen
    ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
        new float[] {0.3F, 0.3F, 0.3F, 0, 0},
        new float[] {0.59F, 0.59F, 0.59F, 0, 0},
        new float[] {0.11F, 0.11F, 0.11F, 0, 0},
        new float[] {0, 0, 0, 1, 0},
        new float[] {0, 0, 0, 0, 1}
    });

    // Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
    // ImageAttributes-Objekt mit der ColorMatrix übergeben
    ImageAttributes imageAttributes = new ImageAttributes();
    imageAttributes.SetColorMatrix(colorMatrix);
    Graphics g = Graphics.FromImage(bitmap);
    g.DrawImage(image, new Rectangle(0, 0, image.Width, image.Height),
        0, 0, image.Width, image.Height, GraphicsUnit.Pixel,
        imageAttributes);
    g.Dispose();

    return bitmap;
}

```

Listing 449: Einfache Erstellung eines Graustufen-Bitmap

Auf die Abbildung eines Programms, das ein Farb-Bild über `CreateGrayscaledBitmap` in ein Graustufen-Bild umwandelt, verzichte ich an dieser Stelle. Sie würden im Buch wohl keinen Unterschied erkennen 😊.

231 Transparente Bilder und Grafiken erzeugen

Die Transparenz von Grafiken, die Sie ausgeben, können Sie bestimmen, indem Sie den Alphawert der Zeichenfarbe angeben. Mit einem Alphawert von 255 werden Farben undurchsichtig ausgegeben, ein Alphawert von 0 steht für eine vollständige Transparenz. Eine Farbe mit einem definierten Alphawert können Sie über die `FromArgb`-Methode der `Color`-Struktur erzeugen, indem Sie in der Variante mit vier Argumenten am ersten Argument den Alphawert übergeben. Listing 450 zeigt dies am Beispiel eines Formulars, in dessen `Paint`-Ereignis zunächst ein halb-transparentes Rechteck und ein halb-transparenter Text gezeichnet werden.

Zum transparenten Zeichnen von Bildern können Sie die `MakeTransparent`-Methode eines `Bitmap`-Objekts, ein `ColorMap`-Array und/oder ein `ColorMatrix`-Objekt (aus dem Namensraum `System.Drawing.Imaging`) verwenden (siehe Rezept 227). Die `MakeTransparent`-Methode erläutere ich hier nicht weiter. Dieser Methode übergeben Sie einfach die Farbe, die in Transparenz umgewandelt werden soll.

Über ein `ColorMap`-Array können Sie einzelne Farben in Farben mit einem niedrigen Alphawert umdefinieren. Damit besitzen Sie mehr Flexibilität, weil Sie über den Alphawert der neuen Farbe den Grad der Transparenz bestimmen können. Ein `ColorMatrix`-Objekt ermöglicht die Verminderung des Alphawerts für das gesamte Bild.

In Listing 450 wird auf diese Weise ein Bitmap, das in der Ressource der Anwendungs-Assembly gespeichert ist, ausgegeben. Der schwarze Hintergrund des Bitmaps wird über ein `ColorMap`-Objekt in einen durchsichtigen Hintergrund umgewandelt. Ein `ColorMatrix`-Objekt übernimmt die Transformation des Alphawerts der einzelnen Pixel, so dass dieser nur noch 60 Prozent des originalen Werts beträgt.

Neben den für `Windows.Forms`-Anwendungen üblichen Referenzen und Importen müssen Sie zum Kompilieren dieses Programms die `Assembly System.Drawing.dll` referenzieren und die Namensräume `System`, `System.Drawing`, `System.Drawing.Imaging` und `System.Reflection` importieren. Außerdem muss der Anwendungs-Ressource ein Bitmap zugeordnet sein, das den Namen `Transparente_Grafiken.Hitchhiker.bmp` trägt.

```
private void StartForm_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // Halb-Transparentes Rechteck und Text zeichnen
    Color color = Color.FromArgb(80, 0, 0, 30);
    g.FillRectangle(new SolidBrush(color), 20, 60,
        this.ClientRectangle.Width - 40, 60);
    Font font = new Font("Tahoma", 28, FontStyle.Bold);
    SizeF textSize = g.MeasureString("Hitchhiker", font);
    int x = (int)((this.ClientRectangle.Width - textSize.Width) / 2);
    g.DrawString("Hitchhiker", font, new SolidBrush(color), x, 65);

    // ColorMap für die Umwandlung aller schwarzen Pixel in
    // durchsichtige Pixel erzeugen
    ColorMap[] colorMap = new ColorMap[1];
    colorMap[0] = new ColorMap();
    colorMap[0].OldColor = Color.Black;
    colorMap[0].NewColor = Color.FromArgb(0, 0, 0, 0);

    // ColorMatrix für die Transformation des Alphawerts erzeugen
    ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
        new float[] {1, 0, 0, 0, 0},
        new float[] {0, 1, 0, 0, 0},
        new float[] {0, 0, 1, 0, 0},
        new float[] {0, 0, 0, 0.6F, 0},
        new float[] {0, 0, 0, 0, 1}
    });

    // Neues ImageAttributes-Objekt erzeugen und das ColorMap- und
    // ColorMatrix-Objekt übergeben
```

Listing 450: Transparentes Zeichnen von Grafiken und Bildern

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

ImageAttributes imageAttributes = new ImageAttributes();
imageAttributes.SetRemapTable(colorMap);
imageAttributes.SetColorMatrix(colorMatrix);

// Bitmap aus der Ressource auslesen und zeichnen
Assembly assembly = Assembly.GetExecutingAssembly();
Bitmap hitchhikerBitmap = new Bitmap(assembly.GetManifestResourceStream(
    "Transparente_Grafiken.Hitchhiker.bmp"));
Rectangle destRect = new Rectangle((int)((this.ClientRectangle.Width -
    hitchhikerBitmap.Width) / 2), 130, hitchhikerBitmap.Width,
    hitchhikerBitmap.Height);
g.DrawImage(hitchhikerBitmap, destRect, 0, 0, hitchhikerBitmap.Width,
    hitchhikerBitmap.Height, GraphicsUnit.Pixel, imageAttributes);

g.Dispose();
hitchhikerBitmap.Dispose();
}

```

Listing 450: Transparentes Zeichnen von Grafiken und Bildern (Forts.)

Abbildung 161 zeigt das Ergebnis dieses Programms. Dem Formular wurde zuvor ein Hintergrundbild (der Mond) zugewiesen.



Abbildung 161: Transparentes Zeichnen eines Rechtecks, eines Textes und eines Bitmaps auf einem Formular mit Hintergrundbild

232 Bilder mit Schatten zeichnen

Das übliche Verfahren zum Zeichnen eines Bildes mit einem Schatten ist, dass unter dem Bild ein gefülltes Rechteck gezeichnet wird, das um einige Pixel nach rechts und unten verschoben ist. Wenn Sie für die Farbe des Rechtecks Schwarz mit einem Alphawert kleiner 255 verwenden, wird der Schatten leicht durchsichtig und damit sehr realistisch.

Die Methode `DrawBitmapWithShadow` in Listing 451 macht genau das. Das Rechteck unter dem Bild wird mit einem Alphawert von 160 gezeichnet, was eine leichte Durchsichtigkeit bewirkt. Damit die Schattengröße einstellbar ist, erwartet `DrawBitmapWithShadow` neben dem `Graphics`-Objekt, auf dem gezeichnet werden soll, dem Bild und der Ausgabeposition auch einen Wert für die Schattengröße.

Diese Methode benötigt eine Referenz auf die Assembly `System.Drawing.dll` und den Import der Namensräume `System` und `System.Drawing`.

```
public static void DrawBitmapWithShadow(Graphics g, Image image, int left,
    int top, int shadowSize)
{
    // Schatten zeichnen
    Color shadowColor = Color.FromArgb(160, 0, 0, 0);
    g.FillRectangle(new SolidBrush(shadowColor), left + shadowSize,
        top + shadowSize, image.Width, image.Height);

    // Bild zeichnen
    g.DrawImage(image, new Rectangle(left, top, image.Width, image.Height),
        0, 0, image.Width, image.Height, GraphicsUnit.Pixel);
}
```

Listing 451: Methode zum Zeichnen eines Bildes mit einem Schatten

Abbildung 162 zeigt eine Anwendung, die über diese Methode ein Bild auf einem Formular ausgegeben hat.



Abbildung 162: Zeichnen eines Bildes mit einem Schatten auf einem Formular

233 Schräg zeichnen

Wenn Sie auf einer Grafikoberfläche in einem bestimmten Winkel schräg zeichnen wollen, müssen Sie dazu eine Transformation der Zeichenmatrix des verwendeten `Graphic`-Objekts einstellen.

Eine Transformation bewirkt, dass die beim Zeichnen übergebenen Koordinaten vor den Ausgaben transformiert werden. Sie können eine Transformation in X/Y-Richtung, eine Transformation in einem Drehwinkel und eine Transformation in eine andere Skalierung erreichen. Wichtig dabei ist, dass die definierten Transformationen auf jede Zeichenoperation separat angewendet werden.

Das Graphics-Objekt verwaltet dazu eine Instanz der Klasse `Matrix`, die Sie über die Eigenschaft `Transform` erreichen. In diese Eigenschaft können Sie ein eigenes erzeugtes und initialisiertes `Matrix`-Objekt schreiben um eine spezielle Transformation zu erreichen (der direkte Zugriff auf das `Matrix`-Objekt der `Graphics`-Instanz führt leider zu keinem Resultat). Die Methode `Translate` eines `Matrix`-Objekts bewirkt eine Verschiebung in X/Y-Richtung, über die Methoden `Rotate` und `RotateAt` können Sie eine Verdrehung erreichen. `Rotate` dreht dabei um die linke obere Ecke, bei `RotateAt` können Sie den Drehpunkt bestimmen.

An Stelle eines eigenen `Matrix`-Objekts können Sie aber auch Methoden des `Graphics`-Objekts direkt verwenden (allerdings etwas eingeschränkt). Die Methode `TranslateTransform` stellt eine Transformation der Zeichenmatrix in X/Y-Richtung ein, `RotateTransform` initiiert eine Drehung um einen bestimmten Winkel mit der oberen linken Ecke als Mittelpunkt.

Wichtig bei der Transformation ist das zweite Argument (`order`) der verwendeten Methoden. Dieses Argument bestimmt, wann welche Transformation angewendet wird. Mit dem Wert `MatrixOrder.Prepend` wird die Transformation vor der vorherigen angewendet, mit dem Wert `MatrixOrder.Append` nach der vorherigen. Dieses Argument ist zur korrekten Anwendung der Transformation sehr wichtig, wenn Sie mehrfach zeichnen, z.B. ein Rechteck um definierte X/Y-Werte verschoben und mehrfach um jeweils 30 Grad weiterverdreht. Da jeder Aufruf einer Transformationsmethode die Transformation relativ zur aktuellen bewirkt, können Sie `Translate` bzw. `TranslateTransform` hier nur einmal aufrufen. Vor dem Zeichnen der Rechtecke rufen Sie dann `Rotate`, `RotateAt` bzw. `RotateTransform` auf um die Drehung um weitere 30 Grad fortzuführen. Dabei müssen Sie mit `MatrixOrder.Prepend` angeben, dass die vorherige Transformation (die Verschiebung) vor der Verdrehung angewendet wird, wenn der Mittelpunkt der einzelnen Drehungen derselbe sein soll.

Listing 452 zeigt das schräge Zeichnen eines Textes auf einer `PictureBox`. Dabei wird die Transformation über `TranslateTransform` zunächst so eingestellt, dass die Koordinaten um 30 Punkte in X- und Y-Richtung verschoben werden. Über `RotateTransform` wird eine Verschiebung um 30 Grad erreicht. Bei der nachfolgenden Ausgabe eines Textes wird diese Transformation angewendet. Das Beispiel setzt die Transformation über `ResetTransform` dann wieder zurück und zeichnet einen geraden Text.

Zum Kompilieren benötigen Sie ein Formular mit einer `PictureBox`, die üblichen Referenzen und Importe und den Import der Namensräume `System.Drawing` und `System.Drawing.Drawing2D`.

```
Graphics g = Graphics.FromHwnd(this.pictureBox1.Handle);

// Transformation einstellen
g.TranslateTransform(30, 30, MatrixOrder.Prepend);
g.RotateTransform(30, MatrixOrder.Prepend);

// SmoothingMode auf HighQuality einstellen, damit die Linien sauber
// gezeichnet werden
g.SmoothingMode = SmoothingMode.HighQuality;
```

Listing 452: Schräges Zeichnen eines Textes

```
// Text ausgeben
g.DrawString("Das ist ein gedrehter Text", new Font("Arial", 20),
    new SolidBrush(Color.Black), 0, 0);

// Transformation wieder zurücksetzen
g.ResetTransform();

// Text ausgeben
g.DrawString("Das ist ein gerader Text", new Font("Arial", 20),
    new SolidBrush(Color.Black), 0, 0);

g.Dispose();
```

Listing 452: Schräges Zeichnen eines Textes (Forts.)



Abbildung 163: Schräges Zeichnen

Die Drehung über `RotateTransform` verwendet immer den linken oberen Eckpunkt der aktuellen Zeichenmatrix (die eventuell über vorherige Transformationen bereits transformiert wurde) als Drehpunkt. Über eine gezielte Verschiebung mit nachfolgender Drehung können Sie Ausgaben recht flexibel verdrehen. Für einige Aufgabenstellungen müssen Sie aber um einen anderen Mittelpunkt drehen. Dazu verwenden Sie dann ein `Matrix`-Objekt und dessen `RotateAt`-Methode, der Sie neben dem Winkel auch den Mittelpunkt der Drehung übergeben können.

Listing 453 zeigt, wie Sie dies ausführen. Das Programm zeichnet auf einer `PictureBox` ein Rechteck, das um 20 Grad verdreht und in der Mitte ausgegeben wird. Dazu wird zuerst ein `Matrix`-Objekt erzeugt und dessen `Translate`-Methode so aufgerufen, dass das Rechteck in der Mitte gezeichnet wird. Über `RotateAt` stellt das Beispiel eine Verdrehung der Zeichenmatrix um 20 Grad ein, wobei der Mittelpunkt des Rechtecks als Drehpunkt angegeben wird. Die `Matrix` wird dann der `Transform`-Eigenschaft zugewiesen. Wie bereits gesagt

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

sind die Erzeugung eines neuen `Matrix`-Objekts und die Zuweisung an `Transform` notwendig, weil die direkte Verwendung der `Transform`-Eigenschaft zu keiner Transformation führt.

Schließlich wird dann nur noch ein Rechteck ausgegeben und das `Graphics`-Objekt freigegeben.

```
Graphics g = Graphics.FromHwnd(this.pictureBox1.Handle);

// SmoothingMode auf HighQuality einstellen, damit die Linien sauber
// gezeichnet werden
g.SmoothingMode = SmoothingMode.HighQuality;

// Matrix-Objekt erzeugen und Transformation einstellen
Matrix matrix = new Matrix();
const int rectangleSize = 150;
matrix.Translate ((int)((pictureBox1.Width / 2) - (rectangleSize / 2)) ,
    (int)((pictureBox1.Height / 2) - (rectangleSize / 2)),
    MatrixOrder.Prepend);
matrix.RotateAt(20, new Point(rectangleSize / 2, rectangleSize / 2),
    MatrixOrder.Prepend );

// Die neue Transformationsmatrix zuweisen
g.Transform = matrix;

// Zeichnen
g.FillRectangle(new SolidBrush(Color.Blue), 0, 0, rectangleSize,
    rectangleSize);
g.DrawRectangle(new Pen(new SolidBrush(Color.Black), 1), 0, 0,
    rectangleSize, rectangleSize);

g.Dispose();
```

Listing 453: Drehen um einen definierten Mittelpunkt



Abbildung 164: Verdrehtes Zeichnen um einen definierten Mittelpunkt

234 Die Breite und Höhe eines auszugebenden Textes bestimmen

Bei der Ausgabe von Texten ist es häufig wichtig zu wissen, wie breit und hoch ein Text bei der Ausgabe in einer definierten Schriftart erscheint. So können Sie einen Text z.B. in der Mitte eines Formulars ausgeben.

Zur Ermittlung der Breite und der Höhe eines Textes können Sie die Methode `MeasureString` eines `Graphics`-Objekts verwenden. Am ersten Argument übergeben Sie den String, am zweiten ein `Font`-Objekt, das die Schriftart bestimmt. `MeasureString` liefert ein `SizeF`-Objekt zurück, aus dessen Eigenschaften `Width` und `Height` Sie die Breite und Höhe des Textes auslesen können.

`MeasureString` führt aber manchmal zu Problemen. Einmal werden per Voreinstellung rechte Leerzeichen nicht mit eingerechnet (linke schon). Hinzu kommt, dass `MeasureString` einen kleinen Rand zum ermittelten Wert addiert, der bewirken soll, dass Ausgaben auf gerasterten Flächen korrekt erfolgen (was auch prinzipiell funktioniert). Das letzte Problem ist, dass `MeasureString` in der Voreinstellung die Breite von Texten nicht korrekt berechnet, wenn diese verschiedene Satzzeichen wie z.B. Doppelpunkte beinhalten.

Sie können die Einstellungen, die für die Berechnung verwendet werden, aber auch anpassen. Dazu setzen Sie ein `StringFormat`-Objekt ein (aus dem Namensraum `System.Drawing`). Ein solches Objekt verwaltet Informationen zum Textlayout. Die `StringFormat`-Klasse bietet dazu zwei statische Eigenschaften `GenericTypographic` und `GenericDefault`, über die Sie ein solches Objekt erzeugen können. Ein `Default-StringFormat`-Objekt definiert zunächst das `Default-Verhalten` (mit Randzugabe), ein `typographisches` liefert echte Werte ohne Zugaben. Über verschiedene Eigenschaften können Sie die Berechnung der Werte beeinflussen. Die Eigenschaft `FormatFlags` verwaltet z.B. Flags für die Formatierung. Das Flag `StringFormatFlags.MeasureTrailingSpaces` bewirkt, dass anhängende Leerzeichen mit eingerechnet werden.

Das Problem der inkorrekten Berechnung der Breite von Strings mit bestimmten Satzzeichen können Sie lösen, indem Sie ein `typografisches StringFormat`-Objekt bei der Berechnung übergeben. Dabei ist es sehr wichtig, dass Sie dieses Objekt ebenfalls der `DrawString`-Methode am Argument `format` übergeben, damit diese dieselben Layoutinformationen verwendet, die bei der Berechnung eingesetzt wurden.

Die Methode `StartForm_Paint` in Listing 454 zeigt, wie Sie mit `MeasureString` umgehen. `StartForm_Paint` (die dem `Paint`-Ereignis eines Formulars zugewiesen ist) zeichnet zuerst einen Text, der vorne und hinten Leerzeichen enthält, mittig auf dem Formular. Um die rechten Leerzeichen mit in der Berechnung zu berücksichtigen, erzeugt und initialisiert die Methode ein neues `Default-StringFormat`-Objekt mit dem Flag `StringFormatFlags.MeasureTrailingSpaces`. Beim Aufruf der `MeasureString`-Methode wird am dritten Argument `-1` übergeben. An diesem Argument können Sie eine Maximalbreite übergeben, `-1` steht dafür, dass keine Maximalbreite definiert ist. Am letzten Argument übergibt `StartForm_Paint` das `StringFormat`-Objekt.

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Danach zeichnet das Beispiel denselben Text um eine Texthöhe nach unten versetzt, verwendet dieses Mal aber ein typografisches `StringFormat`-Objekt. Den Unterschied der beiden Varianten sehen Sie in Abbildung 165.

Dieses Beispiel benötigt eine Referenz auf die Assembly *System.Drawing.dll*, den Import des Namensraums `System.Drawing` und natürlich ein Formular, dessen `Paint`-Ereignis mit dieser Methode initialisiert wurde.

```
private void StartForm_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // Ermitteln der Breite und der Höhe des auszugebenden Textes
    string text = "    Test1:::Test2.Test3    ";
    Font font = new Font("Tahoma", 20, FontStyle.Bold);
    StringFormat sf = StringFormat.GenericDefault;
    sf.FormatFlags = StringFormatFlags.MeasureTrailingSpaces;
    SizeF textSize = g.MeasureString(text, font, -1, sf);

    // Text mittig und etwas nach oben versetzt ausgeben
    float x = (this.ClientRectangle.Width - textSize.Width) / 2;
    float y = (this.ClientRectangle.Height - textSize.Height) / 2;
    y -= textSize.Height / 2;
    g.DrawString(text, font, new SolidBrush(Color.Black), x, y);

    // Noch einmal mit einem typografischen StringFormat-Objekt
    sf = StringFormat.GenericTypographic;
    sf.FormatFlags = StringFormatFlags.MeasureTrailingSpaces;
    textSize = g.MeasureString(text, font, -1, sf);
    x = (this.ClientRectangle.Width - textSize.Width) / 2;
    y = ((this.ClientRectangle.Height - textSize.Height) / 2);
    y += textSize.Height / 2;
    g.DrawString(text, font, new SolidBrush(Color.Black), x, y, sf);
}
```

Listing 454: Berechnen der Breite und Höhe eines auszugebenden Textes zur mittigen Ausgabe

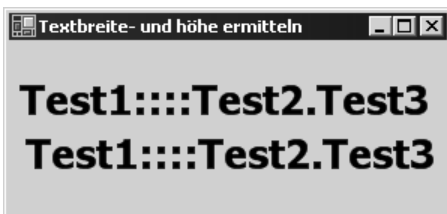


Abbildung 165: Mittige Ausgabe zweier Texte. Die Position des oberen wurde über ein Default-String-Format-Objekt, die des unteren über ein typografisches StringFormat-Objekt berechnet.

COM-Interop mit Office (Automation)

235 Laufende COM-Komponenten-Instanzen referenzieren

Bei der Verwendung von COM-Komponenten, die gleichzeitig interaktiv verwendbare Anwendungen sind (wie es z. B. bei den Microsoft-Office-Anwendungen der Fall ist), ist es häufig sinnvoll, eventuell bereits ausgeführte Instanzen der Anwendung für Fernsteuerungszwecke zu referenzieren. Die Verwendung einer bereits laufenden Instanz führt dazu, dass das Programm schneller ausgeführt wird, weniger Ressourcen verbraucht und die erzeugte Instanz nicht wieder geschlossen werden muss. Das ist z. B. dann der Fall, wenn Sie über Outlook eine E-Mail versenden wollen und davon ausgehen, dass die meisten Anwender Outlook permanent geöffnet haben.

Um eine laufende Instanz einer COM-Komponente zu referenzieren, verwenden Sie die Methode `GetActiveObject` der Klasse `Marshal` aus dem Namensraum `System.Runtime.InteropServices`. Dieser Methode übergeben Sie die Prog-Id der COM-Klasse (siehe »COM-Komponenten mit später Bindung verwenden« in der Einführung). `GetActiveObject` gibt eine `object`-Referenz auf die Instanz zurück, falls gerade eine ausgeführt wird. Wird keine Instanz der COM-Klasse ausgeführt, erzeugt diese Methode eine Ausnahme vom Typ `COMException` mit der Meldung »Vorgang nicht verfügbar«.

Listing 455 setzt dieses Wissen um. Das Programm versucht zunächst, eine laufende Outlook-Instanz zu referenzieren. Der Aufruf der `GetActiveObject`-Methode erfolgt in einer Ausnahmebehandlung (die die Ausnahme allerdings verwirft) um darauf zu reagieren, dass Outlook gerade nicht ausgeführt wird. Da Outlook auf jeden Fall verwendet werden soll, fragt das Programm danach über einen Vergleich der erhaltenen `object`-Referenz auf `null` ab, ob eine Instanz referenziert werden konnte. Ist dies der Fall, wird die `object`-Referenz in den Datentyp `Outlook.Application` konvertiert. Das Programm arbeitet mit früher Bindung, benötigt also Wrapper-Assemblies für die Outlook-COM-Komponenten (siehe »COM-Komponenten mit früher Bindung verwenden« in der Einführung). Sie können aber natürlich auch mit später Bindung arbeiten (was ich aber nicht im Listing darstellen wollte).

Kann keine Outlook-Instanz referenziert werden, erzeugt das Programm einfach eine neue, wobei die Variable `outlookInstanceCreated` für das spätere Beenden von Outlook auf `true` gesetzt wird. Danach erzeugt Listing 455 eine neue E-Mail und versendet diese. Wurde Outlook neu gestartet, beendet das Programm diese Instanz schließlich.

Zum Kompilieren dieses Programms müssen Sie die Outlook-Objektbibliothek *Microsoft Outlook x.x Object Library* (bei einer Standard-Office-Installation ist das die Datei `C:\Programme\Microsoft Office\Office10\msoutl.olb`) referenzieren (wobei *x.x* für die Outlook-Version steht) und die Namensräume `System` und `System.Runtime.InteropServices` importieren.

Basics

Datum
Zeit

Anwen-
dungen

Datei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
media

Autom-
tion

Sons-
tiges

```

// Versuch, eine eventuell bereits ausgeführte Outlook-Instanz zu
// referenzieren
object outlookObject = null;
try
{
    outlookObject = Marshal.GetActiveObject("Outlook.Application");
}
catch {}

Outlook.Application outlook = null;
bool outlookInstanceCreated = false;
if (outlookObject != null)
{
    // Wenn eine Instanz referenziert werden konnte, wird diese in den
    // passenden .NET-Typ konvertiert (frühe Bindung)
    outlook = (Outlook.Application)outlookObject;
}
else
{
    // Wenn keine Instanz referenziert werden konnte, wird einfach eine neue
    // erzeugt
    outlookInstanceCreated = true; // Merker für das Schließen unten
    outlook = new Outlook.ApplicationClass();
}

// Neue E-Mail erzeugen und versenden
Outlook.MailItem mailItem = (Outlook.MailItem)outlook.CreateItem(
    Outlook.OlItemType.olMailItem);
mailItem.To = "donald.duck@entenhausen.de";
mailItem.Subject = "Party";
mailItem.Body = "Hallo Donald, wir machen am Samstag eine Party." +
    "Du bist eingeladen";
mailItem.Send();

// Outlook beenden, wenn eine neue Instanz erzeugt wurde
if (outlookInstanceCreated)
    outlook.Quit();

```

Listing 455: Versenden einer E-Mail über eine ausgeführte bzw. eine neue Outlook-Instanz

Hinweis

Das Versenden von E-Mails über Outlook wird in Rezept 170 grundlegend behandelt. Dort finden Sie den folgenden Hinweis noch einmal, den ich hier nur für den Fall anbringe, dass Sie sich über Outlook wundern ...

Outlook 2000 ab Service Pack 2 und Outlook XP zeigen beim Versuch, über externe Programme E-Mails zu versenden, aus Sicherheitsgründen eine Warnmeldung an (Abbildung 166). Der Ja-Schalter dieses Dialogs kann für fünf Sekunden nicht betätigt werden.

Damit verhindert Microsoft, dass Viren-Programme unkontrolliert E-Mails versenden. Dieses in der Regel eher lästige Feature kann nicht abgeschaltet werden. Sie können jedoch an der Adresse www.express-soft.com/mailmate/clickyes.html das Tool *Express KlickYes* downloaden, das die Betätigung des Ja-Schalters automatisch vornimmt. Nach der Installation finden Sie dieses Tool im Infobereich der Taskleiste. Wählen Sie im Kontextmenü dieses Eintrags die Option *START SUSPENDED* ab und schalten Sie die Option *START ON LOGON* ein, damit *Express KlickYes* beim nächsten Windows-Start automatisch ausgeführt wird. Nach der Installation müssen Sie einmal den Eintrag *RESUME* im Kontextmenü anklicken, damit *Express KlickYes* ohne einen Neustart arbeitet.

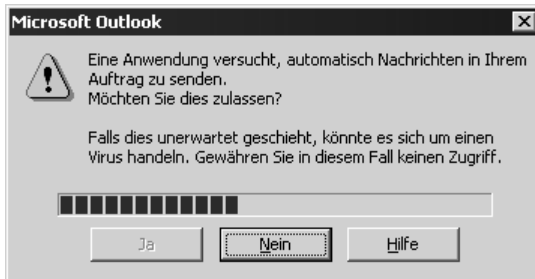


Abbildung 166: Warnmeldung von Outlook 2000 SP 2 und Outlook XP beim Versenden einer E-Mail über ein externes Programm

236 Word-Dokumente öffnen

Word-Dokumente können Sie in .NET nicht direkt öffnen und anzeigen. Dazu müssen Sie Word fernsteuern, also eine neue Word-Instanz erzeugen (oder eine vorhandene referenzieren, siehe Rezept 235) und deren Eigenschaften und Methoden einsetzen. Idealerweise setzen Sie dazu die frühe Bindung ein (siehe »COM-Komponenten mit früher Bindung verwenden« in der Einführung). Dazu referenzieren Sie in Visual Studio .NET die COM-Klassenbibliothek *Microsoft Word x.x Object Library*, wobei *x.x* für die Version von Word steht. Wenn Sie Visual Studio .NET nicht verwenden, erzeugen Sie über *tlbimp.exe* Wrapper-Assemblies für die Datei *msword.olb* (im Office-Ordner):

```
tlbimp.exe "C:\Programme\Microsoft Office\Office10\msword.olb" /out:word.dll
/namespace:Word
```

Die erzeugten Wrapper-Assemblies referenzieren Sie dann beim Kompilieren der Anwendung:

```
csc Demo.cs /r:word.dll,office.dll,vbide.dll
```

Die Fernsteuerung von Word ist relativ einfach. Word stellt, wie andere Office-Anwendungen auch, in seiner Klassenbibliothek ein Objektmodell zur Verfügung, das eine Word-Instanz und deren Dokumente repräsentiert. Das einzig Problematische ist vielleicht, dass Sie dieses Objektmodell nicht kennen. Die Dokumentation finden Sie an der Adresse *msdn.microsoft.com/library/en-us/modcore/html/deovrmicrosoftword2000.asp* oder indem Sie bei *msdn.microsoft.com* nach »Word object model« suchen.

Zum Kompilieren des folgenden Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume *System*, *System.IO*, *System.Reflection* und *System.Windows.Forms* importieren.

Zur Fernsteuerung benötigen Sie zunächst eine Instanz der Klasse *Word.ApplicationClass* (die die Schnittstelle *Word.Application* implementiert). Wenn Sie diese nicht aus einer laufenden Instanz referenzieren (siehe Rezept 235), sollten Sie darauf achten, dass eine neue Word-Instanz immer zunächst unsichtbar ausgeführt wird. Wenn Sie diese sichtbar schalten wollen, müssen Sie deren *Visible*-Eigenschaft auf *true* setzen:

```
// Word-Instanz erzeugen und sichtbar schalten
Word.Application word = new Word.ApplicationClass();
word.Visible = true;
```

Listing 456: Erzeugen einer Word-Instanz und Sichtbar-Schalten derselben

Sie können die Instanz aber auch unsichtbar verwenden, wenn der Anwender nicht sehen soll, was im Hintergrund passiert. Da der Bildschirm nicht aktualisiert werden muss, ist die Verwendung einer unsichtbaren Instanz auch etwas performanter. Sie müssen aber gerade bei unsichtbaren Instanzen darauf achten, diese (über die *Quit*-Methode) schließlich wieder zu beenden, da der Prozess ansonsten weiter ausgeführt wird und Ressourcen verbraucht.

Um ein Dokument zu öffnen, rufen Sie nun die *Open*-Methode der *Documents*-Auflistung auf. Die *Documents*-Auflistung verwaltet alle Dokumente, die in der Word-Instanz geöffnet sind. Im Prinzip handelt es sich dabei um eine ganz normale Auflistung. Der wesentliche Unterschied zu *.NET*-Auflistungen ist, dass *COM*-Auflistungen (und speziell *Documents*) in der Regel 1-basiert sind (also nicht bei 0 beginnen).

Open ist eine typische (Office-)COM-Methode mit einer Vielzahl an optionalen Argumenten. Die meisten dieser Argumente sind für die Praxis eher unwichtig, hier können Sie also einfach *System.Reflection.Missing.Value* übergeben. Da die Argumente der *Open*-Methode als *object* und *By-Reference* deklariert sind, können Sie diesen Wert jedoch nicht direkt einsetzen, sondern müssen dazu eine *object*-Variable verwenden, wie es Listing 457 zeigt. In diesem Beispiel wird das Dokument *hitchhiker.doc* im Ordner der Anwendung geöffnet.

```
// Variable für den Wert Missing.Value für nicht belegte optionale Argumente
object missing = Missing.Value;
// Dokument öffnen
object fileName = Path.Combine(Application.StartupPath, "Hitchhiker.doc");
word.Documents.Open(ref fileName, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing);
```

Listing 457: Einfaches Öffnen eines Word-Dokuments

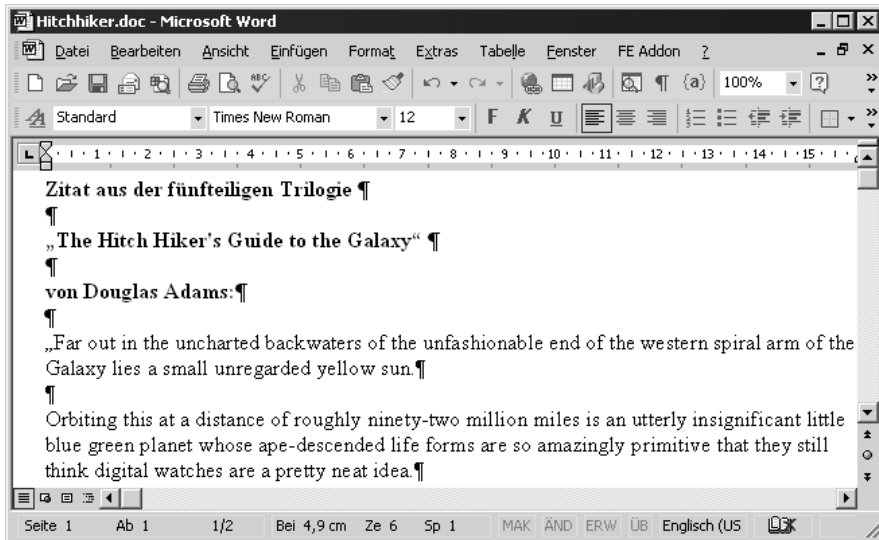


Abbildung 167: Das Beispielprogramm hat Word gestartet und das Dokument geöffnet

Hinweis

Ein guter »Trick« zur Fernsteuerung von Office-Anwendung ist, dass Sie in der Office-Anwendung (sofern möglich) einfach ein Makro aufzeichnen, das die gewünschten Aktionen ausführt. Dann wissen Sie schon einmal, welche Eigenschaften und Methoden Sie einsetzen müssen. Das einzige Problem dabei ist, dass der Office-Makrorekorder (leider) eine spezielle Visual-Basic-Syntax einsetzt, bei der die Argumente einer Methode unter der Angabe des Argumentsnamens verwendet werden. Ein Makro, das das aktuelle Dokument ausdruckt, sieht z.B. so aus:

```
Application.PrintOut FileName="", Range:=wdPrintAllDocument, Item:= _
    wdPrintDocumentContent, Copies:=1, Pages:="", PageType:=wdPrintAllPages, _
    ManualDuplexPrint:=False, Collate:=True, Background:=True, PrintToFile:= _
    False, PrintZoomColumn:=0, PrintZoomRow:=0, PrintZoomPaperWidth:=0, _
    PrintZoomPaperHeight:=0
```

Listing 458: Ein Word-Makro, das das aktuelle Dokument ausdruckt

Wenn Sie diese Syntax nach C# umsetzen, müssen Sie die Argumente natürlich in Klammern setzen und die zu übergebenden Argumente an dem angegebenen Namen (vor dem :=) identifizieren und korrekt platzieren. Optionale und für die auszuführende Aktion unwichtige Argumente können Sie einfach durch `Missing.Value` ersetzen. Die Aufzählungen der verwendeten Konstanten (z.B. `wdPrintAllDocument`) können Sie in Visual Studio .NET über den Objektbrowser ermitteln, indem Sie nach dem Konstantennamen suchen.

In C# würde dieser Methodenaufruf dann so aussehen:

```
object missing = Missing.Value;
object background = true;
object range = Word.WdPrintOutRange.wdPrintAllDocument;
object item = Word.WdPrintOutItem.wdPrintDocumentContent;
object copies = 1;
object pageType = Word.WdPrintOutPages.wdPrintAllPages;
object printToFile = false;
object collate = true;
object manualDuplexPrint = false;
object printZoomColumn = 0;
object printZoomRow = 0;
object printZoomPaperWidth = 0;
object printZoomPaperHeight = 0;

word.PrintOut(ref background, ref missing, ref range, ref missing, ref missing,
    ref missing, ref item, ref copies, ref missing, ref pageType, ref printToFile,
    ref collate, ref missing, ref missing, ref manualDuplexPrint,
    ref printZoomColumn, ref printZoomRow, ref printZoomPaperWidth,
    ref printZoomPaperHeight);
```

Listing 459: Ein nach C# umgesetzter Aufruf einer über den Office Makrorekorder erzeugten Visual Basic COM-Methode

237 Word-Dokumente basierend auf einer Dokumentenvorlage erzeugen, füllen, ausdrucken und speichern

In der Praxis kommt es häufig vor, dass eine Anwendung ein Word-Dokument erzeugen und mit Daten füllen muss, das auf einer Dokumentenvorlage basiert. Damit erleichtern Sie dem Anwender – der in der Regel mit Word sehr gut umgehen kann – die Anpassung auszuführender Dokumente. Ich drucke z.B. meine Rechnungen in einem selbst geschriebenen Buchführungs-Programm über eine spezielle Word-Dokumentenvorlage aus.

Als Beispiel verwende ich eine einfache Dokumentenvorlage, die einen Brief darstellen soll. Diese Vorlage ist mit einigen Textmarken versehen (was schon der ganze Trick ist). In Abbildung 168 sind die Textmarken über spitze Klammern gekennzeichnet.

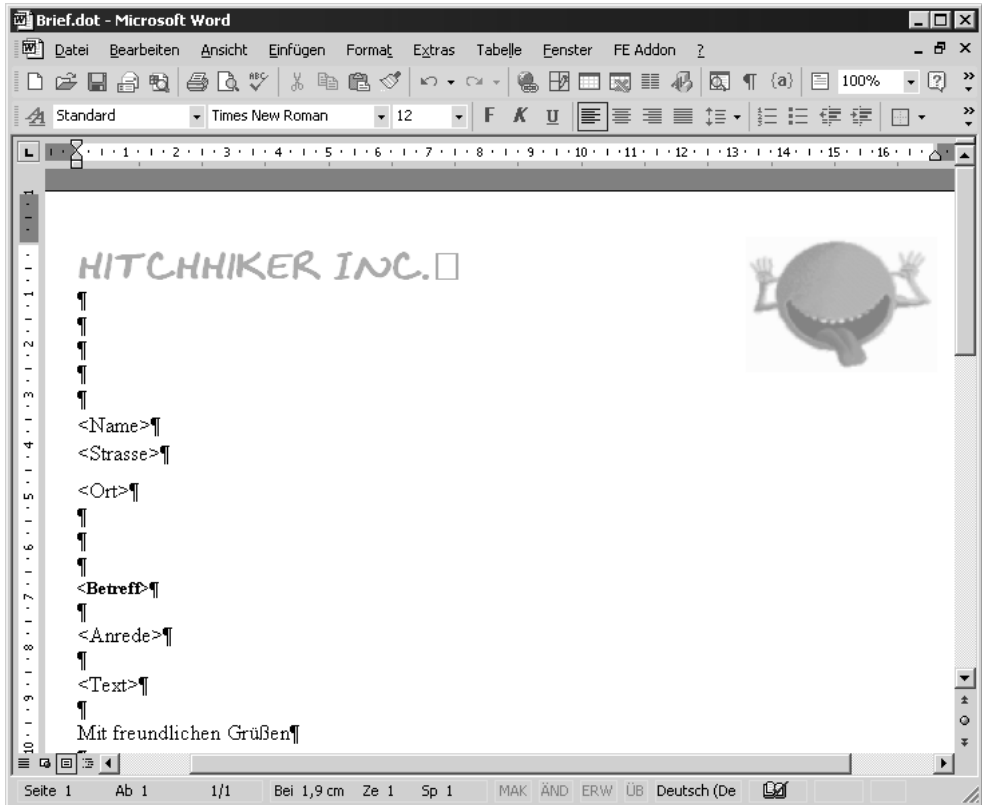


Abbildung 168: Eine einfache Dokumentenvorlage für einen Brief

Hinweis

Falls Sie nicht wissen, wie Sie in Word eine solche Dokumentenvorlage erzeugen: Erstellen Sie ein neues Dokument und fügen Sie den gewünschten Text und die gewünschten Grafiken ein. Erstellen Sie an den Stellen, an denen das Programm Texte einfügen soll, Textmarken, indem Sie die Stelle markieren und den Befehl EINFÜGEN / TEXTMARKE wählen. Speichern Sie das Dokument dann als Dokumentenvorlage (*.dot) ab. Wenn Sie die Dokumentenvorlage nur für eine spezielle Anwendung benötigen, bietet es sich an, diese im Anwendungsordner zu speichern. Ansonsten können Sie auch den Ordner verwenden, der für Ihre Dokumentenvorlagen vorgesehen ist (per Voreinstellung ist das unter Windows XP der Ordner C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Vorlagen).

Um nun basierend auf einer Dokumentenvorlage ein neues Dokument zu erzeugen, erstellen Sie zunächst eine neue Word-Instanz (bzw. referenzieren eine vorhandene, wie ich es prinzipiell im Rezept 235 beschreibe) und rufen die Add-Methode der Documents-Auflistung auf. Grundlagen dazu finden Sie im Rezept 236.

Der Add-Methode können Sie am ersten Argument den Dateinamen einer Dokumentenvorlage übergeben. Listing 460 zeigt dies am Beispiel der Dokumentenvorlage *Brief.dot*, die im Ordner der Anwendung erwartet wird.

Zum Kompilieren des Programms müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume *System*, *System.IO*, *System.Reflection* und *System.Windows.Forms* importieren.

```
// Word-Instanz erzeugen und sichtbar schalten
Word.Application word = new Word.ApplicationClass();
word.Visible = true;

// Neues Dokument basierend auf der Dokumentenvorlage Brief.dot erzeugen
object missing = Missing.Value;
object template = Path.Combine(Application.StartupPath, "Brief.dot");
word.Documents.Add(ref template, ref missing, ref missing, ref missing);
```

Listing 460: Erzeugen eines neuen Word-Dokuments auf der Basis einer Dokumentenvorlage

Nun können Sie die einzelnen Textmarken anspringen und den gewünschten Text einfügen. Dazu verwenden Sie die *GoTo*-Methode der *Selection*-Eigenschaft, der Sie am ersten Argument die *Word*-Konstante *WdGoToItem.wdGoToBookmark* übergeben (weil Sie eine Textmarke anspringen wollen). Da dieses Argument (wie auch die anderen) ein COM-Variant-Typ ist, der By-Reference übergeben wird, müssen Sie zur Übergabe eine *object*-Variable verwenden. Das zweite und dritte Argument sind für unseren Zweck unwichtig. Hier übergeben Sie den Wert *Missing.Value*, wieder als *object*-Variable. Am letzten Argument übergeben Sie den Namen der Textmarke.

```
object what = Word.WdGoToItem.wdGoToBookmark;
object name = "Name";
word.Selection.Goto(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Donald Duck");

name = "Strasse";
word.Selection.Goto(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Entenweg 1");

name = "Ort";
word.Selection.Goto(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Entenhausen");

name = "Betreff";
word.Selection.Goto(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Party");

name = "Anrede";
word.Selection.Goto(ref what, ref missing, ref missing, ref name);
```

Listing 461: Anspringen von Textmarken und Einfügen von Text

```
word.Selection.TypeText("Hallo Donald,");

name = "Text";
word.Selection.Goto(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("wir machen Samstag eine Party. Du bist eingeladen.");
```

Listing 461: Anspringen von Textmarken und Einfügen von Text (Forts.)

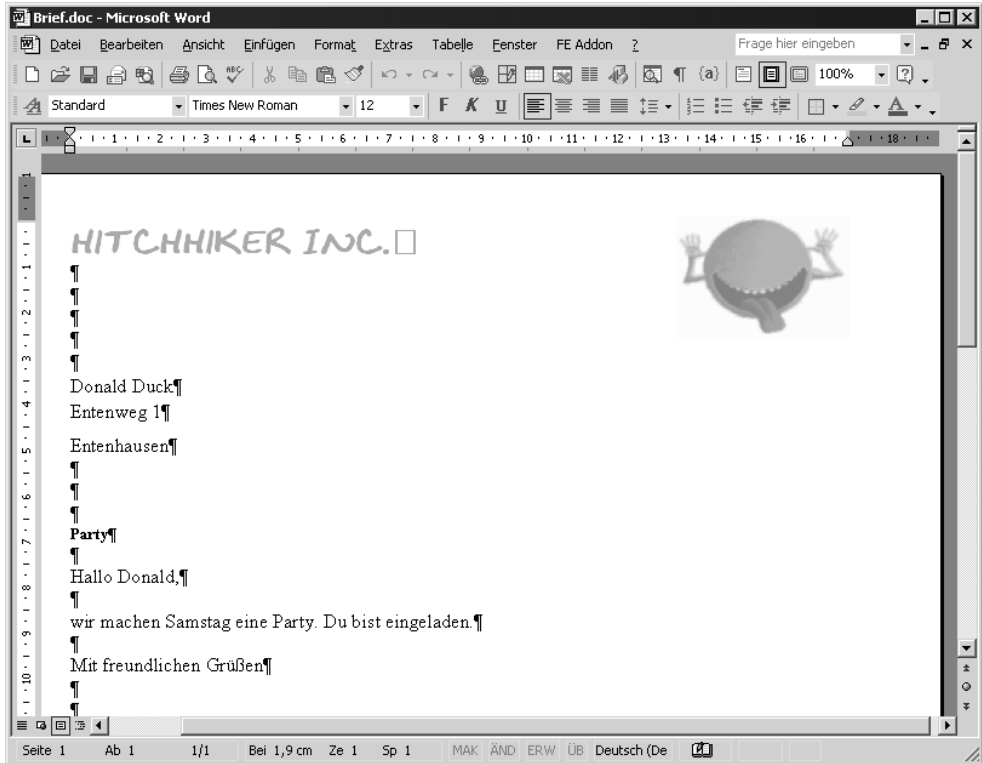


Abbildung 169: Das erzeugte Word-Dokument

Das Dokument können Sie über die `PrintOut`-Methode des aktiven Dokuments ausdrucken (wobei viele für die Praxis eher unwichtige optionale Argumente belegt werden müssen), über die `SaveAs`-Methode speichern (wieder mit vielen optionalen Argumenten) und Word schließlich über die `Quit`-Methode der Word-Instanz beenden:

```
// Das Dokument ausdrucken ...
word.ActiveDocument.PrintOut(ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing);
```

Listing 462: Ausdrucken und Speichern eines Dokuments und Beenden von Word

```
// ... und speichern
object fileName = Path.Combine(Application.StartupPath, "Brief.doc");
word.ActiveDocument.SaveAs(ref fileName, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing);

// Word beenden
object saveChanges = false;
word.Quit(ref saveChanges, ref missing, ref missing);
```

Listing 462: Ausdrucken und Speichern eines Dokuments und Beenden von Word (Forts.)

238 Excel-Arbeitsmappen erzeugen

Excel kann ebenso einfach ferngesteuert werden wie Word (siehe Rezept 236). Zur Erzeugung von Excel-Arbeitsmappen benötigen Sie zunächst eine Instanz der Klasse `ApplicationClass` (die die Schnittstelle `Application` implementiert), die in der COM-Komponente *Excel.exe* (die gleichzeitig auch das interaktiv startbare Programm darstellt) verwaltet wird. Wenn Sie die frühe Bindung verwenden (siehe COM-Komponenten mit früher Bindung verwenden in der Einführung), legen Sie in Visual Studio .NET dazu einen Verweis auf die Komponente *Microsoft Excel x.x Object Library* an, wobei *x.x* für die Excel-Version steht. Wenn Sie Visual Studio .NET nicht besitzen, erzeugen Sie die notwendigen Wrapper-Assemblies über *tlbimp.exe*, indem Sie als Klassenbibliothek die Datei *excel.exe* im Office-Ordner angeben:

```
tlbimp.exe "C:\Programme\Microsoft Office\Office10\Excel.exe" /out:excel.dll /
namespace:Excel
```

Die erzeugten Assemblies referenzieren Sie dann beim Kompilieren der Anwendung:

```
csc Demo.cs /r:excel.dll,office.dll,vbide.dll
```

Um nun eine Excel-Arbeitsmappe zu erstellen, erzeugen Sie ein `Application`-Objekt oder referenzieren Sie eine vorhandene Excel-Instanz (siehe Rezept 235). Über die `Add`-Methode der `Workbooks`-Auflistung können Sie dann eine neue Arbeitsmappe erzeugen. Die `Workbooks`-Auflistung verwaltet alle in der Excel-Instanz geöffneten Arbeitsmappen.

Die `Add`-Methode erwartet ein (in COM optionales) `object`-Argument, über das Sie die zu verwendende Vorlage spezifizieren können. Um eine normale Arbeitsmappe zu erzeugen, übergeben Sie an diesem Argument den Wert `System.Reflection.Missing.Value`. Anders als bei Word sind die Argumente der Excel-Methoden in der Regel nicht `By Reference` deklariert, weswegen Sie prinzipiell den nach `object` konvertierten Wert `Missing.Value` direkt übergeben können. Um den Aufruf zu vereinfachen, verwende ich dafür aber eine Variable.

Um das folgende Beispiel kompilieren zu können müssen Sie neben der Excel-Objektbibliothek die `Assembly System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Reflection` und `System.Windows.Forms` importieren.

```
// Excel-Instanz erzeugen und sichtbar schalten
Excel.Application excel = new Excel.ApplicationClass();
excel.Visible = true;

// Arbeitsmappe ohne Vorlage erzeugen
object missing = Missing.Value;
excel.Workbooks.Add(missing);
```

Listing 463: Erzeugen einer Excel-Arbeitsmappe

Nun können Sie die aktive Arbeitsmappe referenzieren und über die `Add`-Methode der `Worksheets`-Auflistung ein Arbeitsblatt erzeugen. Da Excel so eingestellt werden kann, dass beim Start automatisch Arbeitsblätter erzeugt werden (per Voreinstellung sind das drei), können Sie zuvor abfragen, ob bereits Arbeitsblätter bestehen:

```
// Das aktive Workbook referenzieren
Excel.Workbook workbook = excel.ActiveWorkbook;

// Arbeitsblatt erzeugen
if (workbook.Worksheets.Count == 0)
    workbook.Worksheets.Add(missing, missing, missing, missing);
```

Listing 464: Erzeugen eines Arbeitsblattes

Um mit dem ersten Arbeitsblatt zu arbeiten, sollten Sie dieses referenzieren. Dabei müssen Sie beachten, dass die `Worksheets`-Auflistung wie viele anderen COM-Auflistungen auch 1-basiert ist. Über die `Name`-Eigenschaft können Sie das Arbeitsblatt gleich auch umbenennen:

```
// Das erste Arbeitsblatt referenzieren und umbenennen
Excel.Worksheet worksheet = (Excel.Worksheet)workbook.Worksheets[1];
worksheet.Name = "Einkommensberechnung";
```

Listing 465: Referenzieren und Umbenennen des ersten Arbeitsblattes

Nun können Sie einzelne Zellen oder ganze Zellbereiche mit Text oder Formeln beschreiben und beliebig formatieren. Um herauszufinden, wie Sie einzelne Aktionen programmieren, können Sie in Excel einfach ein Makro aufzeichnen und den erzeugten Visual Basic-Programmcode nach C# konvertieren. Der Vorgang ist ähnlich dem, den ich im Rezept 236 bereits für Word beschrieben habe.

In Visual Basic würden Sie die `Range`-Eigenschaft des `Worksheet`-Objekts verwenden um direkt auf eine Zelle oder auf einen Zellbereich zuzugreifen. Diese spezielle Eigenschaft, der Sie als Index einen oder zwei Zellbezüge (in der in Excel üblichen Form `A1` oder `A1:C3`) als String

übergeben können, kann in C# nicht direkt umgesetzt werden. Die Wrapper-Assembly enthält deswegen statt der Eigenschaft `Range` eine Methode `get_Range`, die ähnlich arbeitet. Dieser Methode übergeben Sie normalerweise am ersten Argument einen Zellbezug. Am zweiten Argument, das einen zweiten Zellbezug aufnimmt, können Sie `Missing.Value` übergeben. Sie erhalten dann ein `Range`-Objekt zurück, über das Sie die Zelle bzw. den Bereich mit einem Text beschreiben oder umformatieren können.

Zum Schreiben eines Textes können Sie in Visual Basic die `Value`-Eigenschaft verwenden, die aber (aus mir unbekannten Gründen) in der Wrapper-Assembly nicht direkt vorhanden ist und über die Methode `set_Value` repräsentiert wird. Alternativ können Sie aber auch die Eigenschaft `Value2` verwenden, die auch in der Wrapper-Assembly implementiert ist.

Das folgende Beispiel schreibt zunächst einen Text in die Zelle A1 und formatiert dann deren Schriftart um. Dann werden noch die Zellen A2 bis A5 mit Texten beschrieben:

```
// Die Zelle A1 referenzieren, beschreiben und die Schriftart definieren
Excel.Range range = worksheet.get_Range("A1", missing);
range.set_Value(missing, "Einkommens-Berechnung");
// Alternative range.Value2 = "Einkommens-Berechnung";
range.Font.Size = 16;
range.Font.Bold = true;

// A2 bis A5 beschreiben
worksheet.get_Range("A2", missing).Value2 = "2000";
worksheet.get_Range("A3", missing).Value2 = "2001";
worksheet.get_Range("A4", missing).Value2 = "2002";
worksheet.get_Range("A5", missing).Value2 = "2003";
```

Listing 466: Beschreiben und Formatieren von Zellen

Über die `Cells`-Eigenschaft können Sie über Integer-Indizes auf Zellen zugreifen, z.B. um diese in einer Schleife zu beschreiben oder auszulesen. Am ersten Argument übergeben Sie den (1-basierten) Index der Zeile, am zweiten den Index der Spalte. `Cells` gibt wie `get_Range` ein `Range`-Objekt zurück, über das Sie den Inhalt und die Formatierung der Zelle erreichen. Das folgende Beispiel schreibt den Inhalt eines Arrays in die Zellen B2 bis B5:

```
double[] sales = {120000, 118000, 130000, 150000};
for (int i = 0; i < 4; i++)
{
    int row = i + 2;
    int col = 2;
    worksheet.Cells[row, col] = sales[i];
}
```

Listing 467: Zugriff auf Zellen über Integer-Indizes

Formeln können Sie über die `Formula`- oder `FormulaLocal`-Eigenschaft eines `Range`-Objekts erzeugen. Dazu verwenden Sie die Syntax, die auch in Excel direkt eingesetzt wird.

Hinweis

Laut der Dokumentation verwendet die `Formula`-Eigenschaft die englische Formelsyntax, für die lokale Variante müssten Sie die `FormulaLocal`-Eigenschaft verwenden. Bei der Fernsteuerung von Excel über ein Visual-Basic-6-Programm funktioniert das auch. Eigenartigerweise konnte ich in meinen Versuchen auch mit der `Formula`-Eigenschaft nur die lokale (deutsche) Formelvariante verwenden.

```
// Eine Formel in die Zelle B6 schreiben und die Zelle fett formatieren
worksheet.get_Range("B6", missing).FormulaLocal = "=SUMME(B2:B5)";
worksheet.get_Range("B6", missing).Font.Bold = true;
```

Listing 468: Erzeugen einer Formel

Statt direkte Bezüge für die Formel zu verwenden, könnten Sie auch über die `FormulaR1C1`-Eigenschaft Formeln mit Bezügen im so genannten Z1S1-Bezugssystem (in Englisch: R1C1-System) übergeben. Dabei geben Sie die Zeile und die Spalte jeweils relativ zur aktuellen Zelle über einen Index in runden (bzw. in der englischen Syntax in eckigen) Klammern an. `Z(-4)S:Z(-1)S` bedeutet z.B.: »Von der Zelle in der aktuellen Zeile -4 und der aktuellen Spalte bis zur Zelle in der aktuellen Zeile -1 und der aktuellen Spalte«. Damit könnten Sie Formeln auch für dynamisch erzeugte Zeilen relativ einfach erzeugen:

```
worksheet.get_Range("B6", missing).FormulaR1C1Local = "=SUMME(Z(-4)S:Z(-1)S)";
```

Hinweis

Wie schon bei der `Formula`-Eigenschaft sollte auch `FormulaR1C1` die Formel eigentlich in der englischen Syntax verwalten. Bei meinen Versuchen war dies aber unter Excel XP nicht möglich. Ich musste auch der `FormulaR1C1`-Eigenschaft die deutsche Syntax übergeben.

Hinweis

Wenn die Formel einen Fehler enthält (bzw. in der falschen Sprachversion übergeben wurde), resultiert dies normalerweise in einer wenig aussagekräftigen Ausnahme vom Typ `COMException` mit der Meldung »Ausnahme von HRESULT: 0x800A03EC«.

Nun können Sie das erzeugte Arbeitsblatt noch ausdrucken, wozu Sie die `PrintOut`-Methode des `Worksheet`-Objekts verwenden. Diese Methode arbeitet mit einigen optionalen Argumenten, über die Sie u.a. den Seitenbereich, die Anzahl der Kopien und den Drucker spezifizieren können. Da ich davon ausgehe, dass normalerweise das ganze Dokument auf dem Standarddrucker gedruckt wird, definiere ich nur die Anzahl der Kopien und übergebe an den anderen Argumenten den Wert `Missing.Value`.

```
object copies = 1;
worksheet.PrintOut(missing, missing, copies, missing, missing, missing,
    missing, missing);
```

Listing 469: Ausdrucken eines Arbeitsblatts

Zum Speichern einer Arbeitsmappe können Sie deren `SaveAs`-Methode aufrufen. Diese Methode besitzt eine Vielzahl an optionalen `object`- und ein eigentlich ebenfalls optionales Argument mit dem definierten Typ `Excel.XlSaveAsAccessMode`. Am ersten Argument übergeben Sie den Dateinamen als String (!). Am zweiten Argument können Sie das Dateiformat in der Form eines Wertes der Aufzählung `Excel.XlFileFormat` bestimmen. Dieses Argument ist interessant, wenn Sie die Datei z.B. in einem älteren Excel-Format speichern wollen, damit ältere Excel-Versionen darauf zugreifen können. Das Beispiel speichert die Datei im Excel 95/97-Format. Die weiteren Argumente, über die Sie u.a. ein Passwort vergeben können, sind relativ uninteressant und können mit `Missing.Value` belegt werden. Am siebten Argument erwartet `SaveAs` dummerweise keinen `object`-Typ, sondern einen Wert der Aufzählung `Excel.XlSaveAsAccessMode`. Dieses Argument bestimmt, ob mehrere Benutzer gleichzeitig auf die Excel-Arbeitsmappe zugreifen können oder nicht. Übergeben Sie hier idealerweise `Excel.XlSaveAsAccessMode.xlNoChange` um die in der Arbeitsmappe bereits vorgenommene Einstellung nicht zu ändern.

`SaveAs` besitzt ein kleines Manko: Falls die Datei bereits existiert, haben Sie keine Möglichkeit, festzulegen, dass diese überschrieben wird. Excel fragt dann immer automatisch nach, ob die Datei überschrieben werden soll, und erzeugt eine Ausnahme vom Typ `COMException`, wenn der Anwender nicht den Ja-Schalter betätigt. Um diesen automatischen Dialog zu vermeiden, sollten Sie selbst abfragen, ob die Datei bereits existiert, und diese gegebenenfalls zuvor löschen, wie ich es in Listing 470 implementiert habe.

```
// Dateiname definieren
string fileName = Path.Combine(Application.StartupPath, "Einkommensberechnung.xls");

// Abfragen, ob die Datei bereits existiert
bool writeFile = true;
if (File.Exists(fileName))
{
    if (MessageBox.Show("Die Datei '" + fileName + "' existiert " +
        "bereits.\r\n\r\nWollen Sie diese Datei überschreiben?",
        Application.ProductName, MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes)
    {
        // Datei löschen
        File.Delete(fileName);
        writeFile = true;
    }
}
if (writeFile)
```

Listing 470: Speichern einer Excel-Arbeitsmappe und Beenden von Excel

```

{
    // Die Arbeitsmappe im Excel-95/97-Format speichern
    object fileFormat = Excel.XlFileFormat.xlExcel9795;
    workbook.SaveAs(fileName, fileFormat, missing, missing, missing, missing,
        Excel.XlSaveAsAccessMode.xlNoChange, missing, missing, missing, missing,
        missing);
}

// Excel beenden
excel.Quit();

```

Listing 470: Speichern einer Excel-Arbeitsmappe und Beenden von Excel (Forts.)

	A	B	C	D	E	F
1	Einkommens-Berechnung					
2	2000	120000				
3	2001	118000				
4	2002	130000				
5	2003	150000				
6		518000				
7						
8						
9						
10						

Abbildung 170: Die erzeugte Excel-Arbeitsmappe

239 Daten in Excel-Arbeitsmappen erweitern

Excel-Arbeitsmappen werden häufig verwendet, um Daten tabellarisch zu verwalten und über diese Daten Berechnungen auszuführen. Wenn Sie diese Daten programmgesteuert erweitern müssen, können Sie, nachdem Sie die Arbeitsmappe geöffnet und das Arbeitsblatt referenziert haben, einfach die letzte Zeile suchen, die noch Daten speichert, und hinter diese Zeile neue Zeilen für die neuen Daten einfügen.

Die folgenden Listings zeigen, wie das programmiert wird. Als Basis verwende ich die Excel-Datei, die in Rezept 238 erstellt wurde. In diesem Rezept behandle ich auch die Grundlagen der Fernsteuerung von Excel.

Zunächst müssen Sie die Arbeitsmappe öffnen und das Arbeitsblatt referenzieren. Dazu erzeugen Sie eine Excel-Instanz und rufen die `Open`-Methode der `Workbooks`-Auflistung auf. `Open` erwartet am ersten Argument den Dateinamen (als String). Die weiteren Argumente, die alle optional sind, bestimmen u. a., ob Links in der Datei automatisch aktualisiert werden sollen, ob die Datei schreibgeschützt geöffnet werden soll, und geben ein eventuelles Pass-

wort an. Da diese Argumente in der Praxis meist unerheblich sind, übergebe ich einfach `Missing.Value`. Die `Open`-Methode gibt eine Referenz auf die geöffnete Arbeitsmappe zurück, die Sie idealerweise einer Variablen vom Typ `Workbook` zuweisen.

Dann referenzieren Sie das anzusprechende Arbeitsblatt über die `Worksheets`-Auflistung des `Workbook`-Objekts. Dieser Auflistung können Sie den Index oder den Namen des Arbeitsblattes übergeben. Listing 471 übergibt den Namen.

Zum Kompilieren dieses Programms müssen Sie neben der Excel-Objektbibliothek die `Assembly System.Windows.Forms.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Reflection` und `System.Windows.Forms` einbinden.

```
// Excel-Instanz erzeugen und sichtbar schalten
Excel.Application excel = new Excel.ApplicationClass();
excel.Visible = true;

// Arbeitsmappe öffnen
object missing = Missing.Value;
string fileName = Path.Combine(Application.StartupPath,
    "Einkommensberechnung.xls");
Excel.Workbook workbook = excel.Workbooks.Open(fileName, missing, missing,
    missing, missing, missing, missing, missing, missing, missing,
    missing, missing, missing, missing);

// Arbeitsblatt 'Einkommensberechnung' referenzieren
Excel.Worksheet worksheet =
    (Excel.Worksheet)workbook.Worksheets["Einkommensberechnung"];

```

Listing 471: Öffnen einer Excel-Arbeitsmappe und Referenzieren eines Arbeitsblatts

Zum Suchen der ersten Zeile, die keine Daten mehr enthält, können Sie die Zeilen in einer Schleife durchgehen und den Wert der Zellen in der Zeile über die `Cells`-Eigenschaft abfragen, der Sie am ersten Argument den Zeilen- und am zweiten Argument den Spaltenindex übergeben. Gibt die `Value2`-Eigenschaft einer Zelle `null` zurück, ist diese Zelle leer. Listing 472 geht die Zeilen ab der ersten Zeile durch und fragt den Wert der Zelle in der ersten Spalte ab. Zur Vereinfachung referenziert das Programm die einzelnen Zellen in der Variable `range`. Da `Cells` eine object-Referenz zurückgibt, muss diese in den Datentyp `Excel.Range` konvertiert werden. Enthält eine Zelle den Wert `null`, bricht das Programm die Schleife ab. Da nun die erste Zelle in der Zeile referenziert ist, die der letzten Datenzeile folgt, referenziert Listing 472 dann über die `get_Offset`-Methode die Zelle oberhalb der gefundenen. `get_Offset` erwartet einen Offset-Index für die Zeile und die Spalte und gibt eine Referenz auf das `Range`-Objekt zurück, das diesem Offset entspricht.

```
// Suchen der ersten Zeile, die in der ersten Spalte keine Daten mehr enthält
int row = 0;
Excel.Range range = null;
do
{
    row++;
    range = (Excel.Range)worksheet.Cells[row, 1];
    if (range.Value2 == null)
        break;
} while (true);

// Die erste Zelle der letzten Datenzeile referenzieren
range = (Excel.Range)worksheet.Cells[row - 1, 1];
```

Listing 472: Suchen der ersten Zeile in einem Excel-Arbeitsblatt, die keine Daten mehr enthält

Diese Methode, die letzte Datenzeile zu finden, ist sehr flexibel, da Sie auch mehrere Zellen daraufhin überprüfen können, ob diese noch Daten enthalten. Wenn Sie sich darauf verlassen können, dass eine Spalte immer Daten enthält und ein Leerwert in dieser Spalte das Ende der Daten kennzeichnet, können Sie auch die folgende, etwas effizientere Lösung einsetzen:

```
// Ausgehend von der Zelle A1 die letzte Zelle suchen, die noch Daten enthält
Excel.Range range = worksheet.get_Range("A1", missing).get_End(
    Excel.XlDirection.xlDown);
```

Listing 473: Effizienteres Ermitteln der letzten Datenzeile

Diese Lösung ruft, ausgehend von der Zelle A1, die `get_End`-Methode unter der Übergabe von `Excel.XlDirection.xlDown` auf. `get_End` gibt dann eine Referenz auf die letzte Zelle zurück, die noch Daten verwaltet. Damit erreichen Sie programmgesteuert das, was Sie in Excel auch durch die Betätigung von `[Strg] + [Pfeil u]` erreichen. Den entsprechenden Programmcode habe ich auch einfach über die Aufzeichnung eines Makros in Excel ermittelt.

Nun, da die Zelle in der ersten Spalte der letzten Datenzeile referenziert ist, könnten Sie die Zelle unterhalb der gefundenen referenzieren und dort eine neue Zeile einfügen (Excel fügt neue Zeilen standardmäßig oberhalb ein). Leider führt das aber zu einem Problem: Auf diese Weise werden die Zellbezüge von Formeln, die sich auf den Datenbereich beziehen (im Beispiel ist das die Formel `SUMME(B2:B5)`) nicht automatisch erweitert. Sie müssten also alle Formeln überarbeiten, was natürlich nicht sinnvoll ist.

Die Lösung dieses Problems ist, dass Sie einfach eine Zeile oberhalb der letzten Datenzeile einfügen und die Daten der letzten Zeile in diese neue Zeile umkopieren. In diesem Fall passt Excel alle Zellbezüge automatisch an. Listing 474 setzt dies für zwei neue Zeilen um. Dabei wird zunächst über die `Insert`-Methode des `Range`-Objekts, das die gesamte aktuelle Zeile darstellt, eine neue Zeile eingefügt. Die gesamte Zeile erhalten Sie über die `EntireRow`-Eigenschaft eines `Range`-Objekts. Normalerweise müsste dann immer noch die erste Zelle der aktuellen Zeile referenziert sein. Da dies aber beim zweiten Einfügen einer Zeile nicht mehr der Fall war, referenziert Listing 474 zur Sicherheit die erste Zelle explizit (über die `Cells`-Eigenschaft der gesamten Zeile).

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Dann kopiert das Programm die Daten der zuvor letzten Datenzeile nach oben in die neue Zeile und fügt die neuen Daten in die aktuelle Zeile ein. Das Ganze wird schließlich noch für eine weitere neue Zeile wiederholt.

```
// Eine Zeile einfügen
range.EntireRow.Insert(missing, missing);

// Die erste Zelle in der aktuellen Zeile referenzieren (zur Sicherheit)
range = (Excel.Range)range.EntireRow.Cells[1, 1];

// Die Daten der aktuellen Zeile in die neue oben kopieren
range.get_Offset(-1, 0).Value2 = range.Value2;
range.get_Offset(-1, 1).Value2 = range.get_Offset(0, 1).Value2;

// Daten in die aktuelle Zeile einfügen
range.Value2 = 2004;
range = range.get_Offset(0, 1);
range.Value2 = 185000;

// Eine weitere Zeile einfügen, Daten kopieren und neue Daten einfügen
range.EntireRow.Insert(missing, missing);
range = (Excel.Range)range.EntireRow.Cells[1, 1];
range.get_Offset(-1, 0).Value2 = range.Value2;
range.get_Offset(-1, 1).Value2 = range.get_Offset(0, 1).Value2;
range.Value2 = 2005;
range = range.get_Offset(0, 1);
range.Value2 = 221000;
```

Listing 474: Einfügen von zwei neuen Zeilen in das Arbeitsblatt

Nun können Sie noch ausdrucken und die Arbeitsmappe speichern. Grundlagen zum Drucken finden Sie im Rezept 238. Speichern können Sie einfach über die `Save`-Methode des `Workbook`-Objekts:

```
// Arbeitsmappe speichern
workbook.Save();

// Excel beenden
excel.Quit();
```

Listing 475: Speichern der Arbeitsmappe und Beenden von Excel

Abbildung 171 zeigt die erweiterte Arbeitsmappe in Excel.

	A	B	C	D	E	F
1	Einkommens-Berechnung					
2	2000	120000				
3	2001	118000				
4	2002	130000				
5	2003	150000				
6	2004	185000				
7	2005	221000				
8		924000				
9						
10						
11						
12						
13						

Abbildung 171: Die erweiterte Excel-Arbeitsmappe

240 Kontakte aus Outlook auslesen

Microsoft Outlook wird von vielen Anwendern zur Verwaltung ihrer Kontakte und als E-Mail-Client eingesetzt. Die Kontakte können Sie natürlich über die Fernsteuerung von Outlook auslesen, z.B. um dem Anwender in Ihrem Programm eine Auswahl der Kontakte anzubieten und damit einen Serienbrief zu generieren.

Dazu referenzieren Sie zunächst die COM-Komponente *Microsoft Outlook x.x Object Library* (bei einer Standard-Office-Installation ist das die Datei *C:\Programme\Microsoft Office\Office10\msoutl.olb*). Zum Kompilieren der Quellcodes in diesem Rezept benötigen Sie (aufgrund der Verwendung der *MessageBox*-Klasse) außerdem eine Referenz auf die Assembly *System.Windows.Forms.dll* und müssen Sie die Namensräume *System* und *System.Windows.Forms* einbinden.

Dann erzeugen Sie wie bei allen anderen Office-Anwendungen eine Instanz der Klasse *Outlook.ApplicationClass*. Alternativ können Sie auch eine vorhandene Outlook-Instanz referenzieren, wie ich es prinzipiell im Rezept 235 beschreibe. Gerade bei Outlook ist diese Technik aufgrund des relativ langsamen Starts sehr interessant. Ein Sichtbar-Schalten ist allerdings bei Outlook nicht möglich.

Zum Zugriff auf die Outlook-Ordner benötigen Sie eine Referenz auf die aktuelle MAPI-Sitzung (Outlook verwendet das Messaging API zur Verwaltung der Ordner und für E-Mail-Funktionalitäten), die Sie über die Eigenschaft *Session* erreichen. Diese Eigenschaft besitzt den Typ *Outlook.NameSpace*. Sofern der Zugang zu Outlook geschützt ist, können Sie das Programm über die *Logon*-Methode des *NameSpace*-Objekts authentifizieren:

```

Outlook.Application outlook = new Outlook.Application();
Outlook.Namespace session = outlook.Session;
string profileName = "Outlook";
string password = null;
bool showDialog = false;
bool newSession = false;
session.Logon(profileName, password, showDialog, newSession);

```

Listing 476: Erzeugen einer Outlook-Instanz, Referenzieren der aktuellen Sitzung und Logon in Outlook

Den Default-Kontakte-Ordner können Sie sehr einfach über die `GetDefaultFolder`-Methode des `NameSpace`-Objekts ermitteln, indem Sie als Argument den Wert `Outlook.OlDefaultFolders.OlFolderContacts` übergeben. Sie erhalten eine Referenz auf ein `MAPIFolder`-Objekt zurück. Ein solches Objekt repräsentiert einen beliebigen Ordner (auch z.B. E-Mail- oder Kalender-Ordner). Über die Eigenschaft `Items`, die eine normale COM-Auflistung (mit dem Startindex 1) ist, erreichen Sie die in einem Ordner gespeicherten Elemente. Dabei müssen Sie ein wenig aufpassen, denn ein Ordner kann beliebige Elemente speichern. Sie können nicht davon ausgehen, dass ein Kontakte-Ordner nur Elemente des Typs `Outlook.ContactItem` verwaltet. Deswegen sollten Sie das Lesen der Elemente wie in Listing 477 gezeigt vornehmen:

```

Outlook.MAPIFolder folder =
    session.GetDefaultFolder(Outlook.OlDefaultFolders.OlFolderContacts);
Console.WriteLine("{0} Kontakte im Ordner {1}", folder.Items.Count,
    folder.Name);

// Durchgehen aller Kontakte im Ordner
for (int i = 1; i <= folder.Items.Count; i++)
{
    Outlook.ContactItem contact = folder.Items.Item(i) as Outlook.ContactItem;
    if (contact != null)
    {
        Console.WriteLine("{0} {1} {2} {3}, {4} {5}", contact.CompanyName,
            contact.FirstName, contact.MiddleName, contact.LastName,
            contact.HomeAddressPostalCode, contact.HomeAddressCity);
    }
}

```

Listing 477: Auslesen der Kontakte im Outlook-Default-Kontakteordner

Ein Outlook-Kontakt-Element besitzt eine Vielzahl an Eigenschaften, die Sie ja wahrscheinlich prinzipiell von Outlook her bereits kennen. Ich verzichte hier auf eine Beschreibung, da die Namen der Eigenschaften größtenteils selbsterklärend sind. Eine recht mager ausgefallene Referenz der Outlook-Objekte finden Sie an der Adresse msdn.microsoft.com/library/en-us/off2000/html/oltocobjectmodelapplication.asp. Da sich im Objektmodell zwischen Outlook 2000 und XP nichts Wesentliches geändert hat, können Sie diese Referenz auch auf Outlook XP anwenden.

Wenn Sie auf spezielle Ordner zugreifen wollen, müssen Sie diese einzeln ausgehend vom Stammordner referenzieren. Handelt es sich um den Unterordner eines anderen Ordners, erreichen Sie diesen über die Eigenschaft `Folders` eines `MAPIFolder`-Objekts. Das folgende Beispiel zeigt, wie Sie den Ordner *Persönliche Ordner/Kontakte/Handy-Kontakte* referenzieren.

```
string folderName1 = "Persönliche Ordner";
string folderName2 = "Kontakte";
string folderName3 = "Handy-Kontakte";

try
{
    folder = (Outlook.MAPIFolder)session.Folders.Item(folderName1);
    folder = folder.Folders.Item(folderName2);
    folder = folder.Folders.Item(folderName3);

    Console.WriteLine();
    Console.WriteLine("{0} Kontakte im Ordner {1}", folder.Items.Count,
        folder.Name);
    for (int i = 1; i <= folder.Items.Count; i++)
    {
        Outlook.ContactItem contact = folder.Items.Item(i) as
            Outlook.ContactItem;
        if (contact != null)
        {
            Console.WriteLine("{0}", contact.CompanyAndFullName);
        }
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    outlook.Quit();
    return;
}
```

Listing 478: Auslesen eines speziellen Kontakte-Ordners

Das Beispiel nutzt übrigens die Eigenschaft `CompanyAndFullName`, die den Namen der Firma und den vollen Namen des Kontakts in einer gut formatierten Form zurückgibt.

Wie in Listing 478 sollten Sie für den Fall, dass der Ordner nicht existiert, eine Ausnahmebehandlung vorsehen. Die Fehlermeldung im Fall eines nicht existenten Ordners (»Der Vorgang konnte nicht ausgeführt werden. Ein Objekt wurde nicht gefunden«) ist einigermassen aussagekräftig.

Schließlich zeige ich noch, wie Sie die einzelnen Kontakte-Ordner einer Outlook-Instanz durchgehen können, um diese mit den enthaltenen Kontakten z.B. in einem `TreeView`-Steuerelement darzustellen. Dazu benötigen Sie eine Methode zum rekursiven Durchgehen der Kontakte-Ordner:

Basics

Datum
ZeitAnwer-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

public static void ReadFolderItems(Outlook.MAPIFolder folder)
{
    // Zunächst alle Elemente auslesen
    Console.WriteLine();
    Console.WriteLine(folder.FullFolderPath);
    for (int i = 1; i <= folder.Items.Count; i++)
    {
        Outlook.ContactItem contact = folder.Items.Item(i) as
            Outlook.ContactItem;
        if (contact != null)
            Console.WriteLine(contact.CompanyAndFullName);
    }

    // Dann alle Unterordner durchgehen, die selbst wieder Kontakte-Ordner sind
    for (int i = 1; i <= folder.Folders.Count; i++)
    {
        Outlook.MAPIFolder subFolder = folder.Folders.Item(i);
        if (subFolder.DefaultItemType == Outlook.OlItemType.olContactItem)
        {
            // Methode rekursiv aufrufen
            ReadFolderItems(subFolder);
        }
    }
}

```

Listing 479: Methode zum rekursiven Durchgehen der Outlook-Kontakte-Ordner

Die Methode, der eine Referenz auf ein `MAPIFolder`-Objekt übergeben wird, liest zunächst wie bereits bekannt alle Kontakte aus diesem Ordner aus. Dann geht sie die Unterordner des Ordners durch, die über die Eigenschaft `Folders` erreichbar sind. Um zu überprüfen, ob es sich um einen Kontakte-Ordner handelt, vergleicht `ReadFolderItems` den Default-Elementtyp der einzelnen Ordner mit der Konstante `Outlook.OlItemType.olContactItem`. Dabei gehe ich davon aus, dass Ordner, deren Default-Elementtyp Kontakte sind, auch Kontakte-Ordner sind 😊. Wird ein untergeordneter Kontakte-Ordner gefunden, ruft die Methode sich mit diesem Ordner wieder auf. Im Programm müssen Sie `ReadFolderItems` lediglich mit einem Wurzel-Ordner aufrufen:

```

string rootFolderName = "Persönliche Ordner";
ReadFolderItems((Outlook.MAPIFolder)session.Folders.Item(rootFolderName));

```

Am Ende Ihrer Arbeit sollten Sie nicht vergessen, die Outlook-Instanz freizugeben, da der Prozess ansonsten bestehen bleibt und den Arbeitsspeicher belastet:

```

outlook.Quit();

```

241 Informationen über den Aufrufer in einer Methode oder einem Konstruktor ermitteln

Tritt in einer komplexen Anwendung innerhalb einer Methode ein Fehler auf, ist es häufig sinnvoll zu wissen, welche andere Methode diese Methode aufgerufen hat. Wenn der Fehler auf Ihrem Entwicklungssystem auftritt, können Sie beim Debuggen die Aufrufliste zu Rate ziehen. Tritt der Fehler aber nur auf einem Anwender-Rechner auf, wird das Debugging schon schwieriger. In diesem Fall ist es sinnvoll, Informationen über den Aufrufer der Methode auszulesen und in der Fehlermeldung auszugeben.

Dazu können Sie einfach die `StackTrace`-Eigenschaft des `Exception`-Objekts auslesen, das Sie im `catch`-Block einer Ausnahmebehandlung erhalten. Sie können aber auch direkt den Namen der aufrufenden Methode auslesen. Sinnvoll ist das ebenfalls, wenn Sie in einer Logdatei protokollieren wollen, wie oft und von welcher anderen Methode eine Methode aufgerufen wird.

Um den Namen der aufrufenden Methode herauszufinden, können Sie einfach ein neues `StackTrace`-Objekt erzeugen (aus dem Namensraum `System.Diagnostics`). Ein solches Objekt verwaltet `StackFrame`-Instanzen, die Informationen zu den einzelnen Aufrufen in der Aufrufliste enthalten. Über die `GetFrame`-Methode, der Sie einen Index übergeben, können Sie die einzelnen Stack-Frames auslesen. Über deren `GetMethod`-Methode erhalten Sie ein `MethodBase`-Objekt mit Informationen zur Methode. Über deren Eigenschaft `Name` erhalten Sie den Namen, die Eigenschaft `ReflectedType` gibt Informationen zur Klasse oder Struktur, in der die Methode gespeichert ist.

Listing 480 zeigt, wie Sie mit dieser Technik den Namen und die Klasse der aufrufenden Methode herausfinden. Das Beispiel liest den Stack-Frame mit dem Index 1 aus (der Stack-Frame mit dem Index 0 ist der der Methode `Test` selbst).

Das Beispiel erfordert den Import der Namensräume `System` und `System.Diagnostics`.

```
public static void Test()
{
    // Den Namen und die Klasse der aufrufenden Methode ermitteln
    StackTrace stackTrace = new StackTrace();
    string callerMethodName = stackTrace.GetFrame(1).GetMethod().Name;
    string callerClassName =
        stackTrace.GetFrame(1).GetMethod().ReflectedType.Name;

    Console.WriteLine("Aufrufende Methode: {0}", callerMethodName);
    Console.WriteLine("Aufrufende Klasse: {0}", callerClassName);
}
```

Listing 480: Ermittlung des Namens und des Klassennamens der aufrufenden Methode

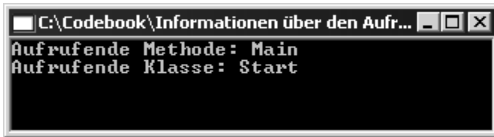


Abbildung 172: Das Beispielprogramm in Aktion

242 Parameter an Threads übergeben und Ergebnisse auslesen

Wenn Sie einen Thread programmieren wollen, müssen Sie dazu eine Methode implementieren, die der Signatur des Delegates `ThreadStart` entspricht. Diese Signatur erlaubt keine Argumente und besitzt den Rückgabotyp `void`. Oftmals müssen beim Start eines Threads aber Argumente übergeben werden. Und manchmal muss ein Ergebnis zurückgeliefert werden.

Die Lösung dieses Problems liegt in der Implementierung einer Klasse mit beliebigen Eigenschaften. Eine Methode dieser Klasse entspricht der `ThreadStart`-Signatur und wird später über eine Instanz der Klasse im Thread gestartet. Da diese Methode Zugriff auf alle Eigenschaften des Objekts besitzt, können Sie vor dem Start des Threads in den Eigenschaften Parameter definieren und nach dem Ende des Threads aus den Eigenschaften Rückgabewerte auslesen.

Das folgende Beispiel demonstriert diese Vorgehensweise an einer Primzahlberechnung. Ein Thread soll alle Primzahlen zwischen zwei gegebenen Grenzwerten ermitteln. Dazu habe ich eine Klasse `PrimeNumberCalculator` entwickelt, die in ihrer Methode `Calc` alle Zahlen zwischen den Werten der Eigenschaften `MinValue` und `MaxValue` daraufhin überprüft, ob es sich um eine Primzahl handelt. Gefundene Primzahlen werden in einem Steuerelement ausgegeben, das in der Eigenschaft `ResultControl` referenziert wird. Bei Ausgaben auf dem Steuerelement wird der Zugriff darauf für andere Threads über einen `lock`-Block gesperrt, damit keine Probleme beim gleichzeitigen Zugriff entstehen.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Threading` importieren. Der restliche Quellcode des Beispiels erfordert die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import des Namensraums `System.Windows.Forms`.

```
private class PrimeNumberCalculator
{
    public long MinValue;
    public long MaxValue;
    public Control ResultControl;

    /* Methode für den Thread */
    public void Calc()
    {
```

Listing 481: Klasse für die Ausführung eines Threads unter der Übergabe von Parametern

```

// Berechnen aller Primzahlen von MinValue bis MaxValue
for (long i = this.MinValue; i <= this.MaxValue; i++)
{
    bool isPrimeNumber = true;
    for (long j = 2; j < i; j++)
    {
        if (i % j == 0)
        {
            isPrimeNumber = false;
            break;
        }
        if (isPrimeNumber)
        {
            lock (this.ResultControl)
            {
                this.ResultControl.Text = i.ToString();
            }
        }
    }
}
}
}
}

```

Listing 481: Klasse für die Ausführung eines Threads unter der Übergabe von Parametern (Forts.)

Hinweis

Dass der »Algorithmus« zur Ermittlung der Primzahlen in diesem Beispiel weit von dem Ideal entfernt ist, ist mir übrigens bewusst 😊.

In einem Windows-Formular kann der Anwender die zwei Grenzwerte eingeben (Abbildung 173) und eine Berechnung starten.

Abbildung 173: Das Formular zur Demonstration der Übergabe von Parametern an einen Thread

In der Ereignisbehandlungsmethode des Thread-starten-Schalters wird nun eine Instanz der `PrimeNumberCalculator`-Klasse erzeugt und mit den Eingaben und einer Referenz auf das Ausgabe-Steuerelement (ein `Label`, das unter dem Schalter angelegt ist) initialisiert. Dann wird der Thread erzeugt, wobei die `Calc`-Methode des `PrimeNumberCalculator`-Objekts als Thread-

methode übergeben wird. Vor dem Start des Threads setzt `startButton_Click` dessen Eigenschaft `IsBackground` auf `true`, damit der Thread automatisch angehalten wird, wenn der Hauptthread der Anwendung beendet wird (was ansonsten nicht der Fall wäre). Dann wird der Thread nur noch gestartet.

```
private void startButton_Click(object sender, System.EventArgs e)
{
    // Instanz der PrimeNumberCalculator-Klasse erzeugen und initialisieren
    PrimeNumberCalculator pnc = new PrimeNumberCalculator();
    pnc.MinValue = Convert.ToInt64(this.minValueTextBox.Text);
    pnc.MaxValue = Convert.ToInt64(this.maxValueTextBox.Text);
    pnc.ResultControl = this.infoLabel;

    // Thread mit der Calc-Methode des PrimeNumberCalculator-Objekts starten
    Thread thread = new Thread(new ThreadStart(pnc.Calc));
    thread.IsBackground = true;
    thread.Start();
}
```

Listing 482: Starten eines Threads mit der Übergabe von Argumenten

Zum Kompilieren dieses Beispiels benötigen Sie ein Formular mit zwei TextBoxen mit Namen `minValueTextBox` und `maxValueTextBox`, einem Label mit Namen `infoLabel` und einem Schalter mit Namen `startButton`. Neben der Referenzierung der Assembly `System.Windows.Forms.dll` und der Einbindung des Namensraums `System.Windows.Forms` müssen Sie die Namensräume `System` und `System.Threading` importieren.

243 Objekte binär serialisieren

Wenn der Zustand eines Objekts zwischen zwei Programmaufrufen bestehen bleiben soll, wenn Sie ein Objekt von einem Programm zu einem anderen Programm oder System versenden wollen oder wenn das Objekt in einer Datenbank gespeichert werden soll, müssen Sie das Objekt serialisieren. Dabei werden die Daten des Objekts in einen Stream geschrieben, den Sie dann beliebig weiterverarbeiten können.

Zum Serialisieren und Deserialisieren können Sie die Klasse `BinaryFormatter` aus dem Namensraum `System.Runtime.Serialization.Formatters.Binary` verwenden. Die Methode `Serialize` serialisiert ein Objekt in einen binären Stream, die Methode `Deserialize` deserialisiert einen binären Stream in ein Objekt. Dabei werden alle Daten des Objekts – auch die privaten – in den Stream geschrieben. Im Rezept 244 finden Sie übrigens eine andere Möglichkeit, nämlich das Serialisieren nach XML.

Die einzige Voraussetzung für das Serialisieren ist, dass die Klasse des Objekts mit dem Attribut `Serializable` gekennzeichnet ist:

```
[Serializable]
public class Person
{
    public string FirstName;
    public string LastName;
    public DateTime BirthDate;
}
```

Listing 483: Klasse für serialisierbare Objekte

Objekte einer solchen Klasse können Sie dann sehr einfach serialisieren und deserialisieren. Die Methode `SerializeToFile` in Listing 484 zeigt, wie Sie ein Objekt in eine Datei serialisieren. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Runtime.Serialization`, `System.IO` importieren.

```
public static void SerializeToFile(object obj, string fileName)
{
    FileStream fileStream = null;
    try
    {
        // FileStream für die Datei erzeugen
        fileStream = new FileStream(fileName, FileMode.Create,
            FileAccess.Write);

        // Objekt serialisieren
        BinaryFormatter bf = new BinaryFormatter();
        bf.Serialize(fileStream, obj);
    }
    finally
    {
        if (fileStream != null)
            fileStream.Close();
    }
}
```

Listing 484: Methode zum Serialisieren eines Objekts in eine Datei

Das Deserialisieren eines Objekts aus einer Datei ist ebenfalls sehr einfach:

```
public static object DeserializeFromFile(string fileName)
{
    FileStream fileStream = null;
    try
    {
        // FileStream für die Datei erzeugen
        fileStream = new FileStream(fileName, FileMode.Open, FileAccess.Read);

        // Objekt deserialisieren
```

Listing 485: Deserialisieren eines Objekts aus einer Datei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

        BinaryFormatter bf = new BinaryFormatter();
        return bf.Deserialize(fileStream);
    }
    finally
    {
        if (fileStream != null)
            fileStream.Close();
    }
}

```

Listing 485: Deserialisieren eines Objekts aus einer Datei (Forts.)

Beide Methoden behandeln Ausnahmen nicht um diese direkt an den Aufrufer weiterzugeben.

Um Objekte in einer Datenbank speichern zu können, müssen Sie diese in ein Byte-Array serialisieren. Dieses Array können Sie dann in Datenbankfelder vom Typ BLOB (Binary Large Object) oder ähnliche Felder schreiben und daraus auslesen.

Zum Serialisieren in ein Byte-Array und zum Deserialisieren aus einem solchen Array können Sie ein `MemoryStream`-Objekt (aus dem Namensraum `System.IO`) verwenden. Die Methoden `SerializeToByteArray` und `DeserializeFromByteArray` in Listing 486 zeigen, wie Sie dies programmieren können. Beide Methoden erfordern den Import der Namensräume `System.Runtime.Serialization.Formatters.Binary` und `System.IO`.

```

/* Methode zum Serialisieren eines Objekts in ein Byte-Array */
public static byte[] SerializeToByteArray(object obj)
{
    MemoryStream ms = null;
    try
    {
        // MemoryStream erzeugen
        ms = new MemoryStream();

        // Objekt serialisieren
        BinaryFormatter bf = new BinaryFormatter();
        bf.Serialize(ms, obj);

        return ms.ToArray();
    }
    finally
    {
        if (ms != null)
            ms.Close();
    }
}

/* Methode zum Deserialisieren eines Objekts aus einem Byte-Array */

```

Listing 486: Methoden zum Serialisieren und Deserialisieren eines Objekts in und aus einem Byte-Array

```
public static object DeserializeFromByteArray(byte[] data)
{
    MemoryStream ms = null;
    try
    {
        // MemoryStream erzeugen
        ms = new MemoryStream(data);

        // Objekt deserialisieren
        BinaryFormatter bf = new BinaryFormatter();
        return bf.Deserialize(ms);
    }
    finally
    {
        if (ms != null)
            ms.Close();
    }
}
```

Listing 486: Methoden zum Serialisieren und Deserialisieren eines Objekts in und aus einem Byte-Array (Forts.)

Das folgende Beispiel zeigt, wie Sie mit Hilfe der Methoden `SerializeToFile` und `DeserializeFromFile` ein `Person`-Objekt in eine Datei serialisieren und wieder daraus deserialisieren. Beim Deserialisieren ist eine Typumwandlung notwendig, da `DeserializeFromFile` eine `object`-Referenz zurückgibt. Das Beispiel benötigt aufgrund der Verwendung der `Application`-Klasse die Referenzierung der Assembly `System.Windows.Forms.dll` und den Import des Namensraums `System.Windows.Forms`. Daneben müssen Sie die Namensräume `System` und `System.IO` importieren.

```
// Person-Objekt erzeugen
Person person = new Person();
person.FirstName = "Zaphod";
person.LastName = "Beeblebox";
person.BirthDate = new DateTime(1900, 1, 1);

// Objekt in eine Datei serialisieren
string fileName = Path.Combine(Application.StartupPath, "Person.dat");
SerializeToFile(person, fileName);

// Objekt aus einer Datei deserialisieren
person = (Person)DeserializeFromFile(fileName);

Console.WriteLine("{0}\r\n{1}\r\n{2}", person.FirstName, person.LastName,
    person.BirthDate.ToShortDateString());
```

Listing 487: Serialisieren eines Objekts in eine Datei

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

244 Objekte nach XML serialisieren und von XML deserialisieren

Zum Serialisieren eines Objekts können Sie neben einer binären Serialisation (siehe Rezept 243) auch eine verwenden, die als Datenformat XML verwendet. Das .NET Framework stellt dazu die Klasse `XmlSerializer` aus dem Namensraum `System.Xml.Serialization` zur Verfügung. Über die Methode `Serialize` können Sie ein Objekt in einen `Stream` oder ein `XmlWriter`- oder `TextWriter`-Objekt serialisieren, `Deserialize` deserialisiert die Daten eines `Stream`-, `XmlReader`- oder `TextReader`-Objekts in ein Objekt. Mit diesen Methoden können Sie ein Objekt sehr einfach in eine Datei oder einen XML-String schreiben.

Hinweis

Anders als bei der binären Serialisation werden dabei nur die öffentlichen Daten in das XML-Dokument geschrieben. Private und geschützte Daten werden nicht berücksichtigt. Für Eigenschaften, die mit `set-/get`-Methoden auf private Daten zugreifen, ist das kein Problem: Diese werden korrekt ausgewertet. Problematisch wird die XML-Serialisation nur dann, wenn private Daten ausschließlich über den Konstruktor oder über normale Methoden geschrieben werden und keine oder nur Eigenschaften zum Lesen dieser Daten zur Verfügung stehen. In diesem Fall wird aber keine Ausnahme erzeugt. Die Daten werden einfach nicht in das XML-Dokument geschrieben.

Eine weitere Einschränkung ist, dass das zu serialisierende Objekt einen (parameterlosen) Standardkonstruktor zur Verfügung stellen muss. Objekte, die keinen solchen besitzen (wie z.B. `Font`-Instanzen) können Sie leider nicht in XML serialisieren. Dazu finden Sie aber eine Lösung im Rezept 245.

Die Methode `SerializeToFile` in Listing 488 serialisiert ein Objekt in eine XML-Datei. Da bei XML-Dateien deren Codierung wichtig ist, wird diese neben dem Objekt und dem Dateinamen am dritten Argument übergeben. Bei der Erzeugung einer `XmlSerializer`-Instanz wird der Typ des Objekts dem Konstruktor übergeben. Die `Serialize`- und die `Deserialize`-Methode verwenden intern `Reflection` um die Eigenschaften des Objekts zu ermitteln. Deswegen muss die Klasse auch nicht wie bei der binären Serialisation mit dem Attribut `Serializable` gekennzeichnet werden.

`SerializeToFile` serialisiert das übergebene Objekt dann über eine `StreamWriter`-Instanz. Auf diese Weise kann die Codierung mit übergeben werden.

Zum Kompilieren dieser Methode (und der folgenden) müssen Sie die Namensräume `System`, `System.IO`, `System.Xml`, `System.Xml.Serialization` und `System.Text` importieren.

```
public static void SerializeToFile(object obj, string fileName,
    Encoding encoding)
{
    StreamWriter streamWriter = null;
    try
```

Listing 488: Methode zum Serialisieren eines Objekts in eine XML-Datei

```

{
    // XmlSerializer für den Typ des Objekts erzeugen
    XmlSerializer serializer = new XmlSerializer(obj.GetType());

    // Objekt über ein StreamWriter-Objekt serialisieren
    streamWriter = new StreamWriter(fileName, false, encoding);
    serializer.Serialize(streamWriter, obj);
}
finally
{
    if (streamWriter != null)
        streamWriter.Close();
}
}

```

Listing 488: Methode zum Serialisieren eines Objekts in eine XML-Datei (Forts.)

Die Methode `SerializeToXmlString` in Listing 489 serialisiert ein Objekt auf eine ähnliche Weise in einen String. Um die Codierung bestimmen zu können, wird ein `StreamWriter`-Objekt eingesetzt, das einen `MemoryStream` zur Speicherung der Daten verwendet. Die Daten dieses Streams werden schließlich in ein Byte-Array geschrieben und über die `GetString`-Methode des übergebenen `Encoding`-Objekts in einen String umgewandelt.

```

public static string SerializeToXmlString(object obj, Encoding encoding)
{
    MemoryStream memoryStream = null;
    StreamWriter streamWriter = null;
    try
    {
        // XmlSerializer für den Typ des Objekts erzeugen
        XmlSerializer serializer = new XmlSerializer(obj.GetType());

        // Objekt über ein MemoryStream-Objekt serialisieren
        memoryStream = new MemoryStream();
        streamWriter = new StreamWriter(memoryStream, encoding);
        serializer.Serialize(streamWriter, obj);

        // MemoryStream in einen String umwandeln und diesen zurückgeben
        byte[] buffer = memoryStream.ToArray();
        return encoding.GetString(buffer, 0, buffer.Length);
    }
    finally
    {
        if (memoryStream != null)
            memoryStream.Close();
        if (streamWriter != null)
            streamWriter.Close();
    }
}

```

Listing 489: Methode zum Serialisieren eines Objekts in einen XML-String

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Das Deserialisieren aus einer XML-Datei übernimmt die Methode `DeserializeFromXmlFile` in Listing 490. Da zum Deserialisieren die Codierung bekannt sein muss, wird diese am dritten Argument übergeben. `DeserializeFromXmlFile` ruft zum Deserialisieren die `Deserialize`-Methode eines `XmlSerializer`-Objekts auf. Da dabei wieder der Typ des Objekts übergeben werden muss, erwartet `DeserializeFromXmlFile` diesen am Argument `objectType`. Als Basis für die Deserialisation wird ein `StreamReader`-Objekt verwendet, da ein solches mit einem `Encoding`-Objekt initialisiert werden kann und damit die definierte Codierung berücksichtigt.

```
public static object DeserializeFromXmlFile(string fileName, Type objectType,
    Encoding encoding)
{
    StreamReader streamReader = null;
    try
    {
        // XmlSerializer für den Typ des Objekts erzeugen
        XmlSerializer serializer = new XmlSerializer(objectType);

        // Objekt über ein StreamReader-Objekt serialisieren
        streamReader = new StreamReader(fileName, encoding);
        return serializer.Deserialize(streamReader);
    }
    finally
    {
        if (streamReader != null)
            streamReader.Close();
    }
}
```

Listing 490: Methoden zum Deserialisieren einer XML-Datei

Die Methode `DeserializeFromXmlString` in Listing 491 deserialisiert auf eine ähnliche Weise wie `DeserializeFromXmlFile` einen XML-String. Als Zwischenspeicher für den deserialisierten String wird hier ein `MemoryStream`-Objekt eingesetzt, das mit dem Byte-Array initialisiert wird, das die `GetBytes`-Methode des übergebenen `Encoding`-Objekts zurückgibt.

```
public static object DeserializeFromXmlString(string xmlString,
    Type objectType, Encoding encoding)
{
    MemoryStream memoryStream = null;
    try
    {
        // XmlSerializer für den Typ des Objekts erzeugen
        XmlSerializer serializer = new XmlSerializer(objectType);

        // Objekt über ein MemoryStream-Objekt deserialisieren
        memoryStream = new MemoryStream(encoding.GetBytes(xmlString));
        return serializer.Deserialize(memoryStream);
    }
}
```

Listing 491: Methode zum Deserialisieren eines XML-Strings

```

    finally
    {
        if (memoryStream != null)
            memoryStream.Close();
    }
}

```

Listing 491: Methode zum Deserialisieren eines XML-Strings (Forts.)

Listing 492 zeigt ein `Person`-Objekt (aus der Klasse `Person`, die in Rezept 243 beschrieben wird), das über `SerializeToXmlFile` serialisiert wurde.

```

<?xml version="1.0" encoding="iso-8859-1"?>
<Person xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <FirstName>Zaphod</FirstName>
  <LastName>Beeblebrox</LastName>
  <BirthDate>1900-01-01T00:00:00.0000000+01:00</BirthDate>
</Person>

```

Listing 492: Ein über `SerializeToXmlFile` serialisiertes `Person`-Objekt

245 Font- und andere Objekte in einen String serialisieren

In vielen Anwendungen ist es notwendig, eine Schriftart, die vom Anwender ausgewählt werden kann, in einer Konfigurationsdatei oder Datenbank verwalten, um diese beim nächsten Programmstart wieder auslesen zu können. `Font`-Objekte lassen sich aber leider nicht über eine Instanz der Klasse `XmlSerializer` nach XML serialisieren, weil die `Font`-Klasse keinen öffentlichen Standardkonstruktor besitzt. Eine Serialisation in binäre Daten über ein `BinaryFormatter`-Objekt (siehe Rezept 243) ist zwar möglich, die Speicherung der binären Daten aber zu aufwändig.

Meine erste Idee zur Lösung dieses Problems, einfach ein `SoapFormatter`-Objekt (aus dem Namensraum `System.Runtime.Serialization.Formatters.Soap`) zu verwenden, das Objekte in das auf XML basierende SOAP-Format serialisiert, scheiterte daran, dass die Deserialisation aus mit unbekannten Gründen zu einer Ausnahme führte.

Um `Font`-Objekte dennoch als String speichern zu können ohne die Daten selbst auswerten zu müssen, habe ich das jeweilige Objekt einfach binär in einen `MemoryStream` serialisiert, und daraus einen Base64-String erzeugt. Die Methoden `SerializeFont` und `DeserializeFont` in Listing 493 sind das Ergebnis dieser Idee. In einer ähnlichen Form können Sie natürlich alle weiteren Objekte, die sich nicht über ein `XmlSerializer`-Objekt serialisieren lassen, in einen String umwandeln.

Zum Kompilieren dieser Methoden müssen Sie die Assembly `System.Drawing.dll` referenzieren und die Namensräume `System`, `System.IO`, `System.Runtime.Serialization.Formatters.Binary` und `System.Drawing` importieren.

```

/* Methode zum Serialisieren eines Font-Objekts in einen String */
public static string SerializeFont(Font font)
{
    // Das Objekt in einen MemoryStream serialisieren
    BinaryFormatter sf = new BinaryFormatter();
    MemoryStream ms = new MemoryStream();
    sf.Serialize(ms, font);

    // Den MemoryStream in ein Byte-Array schreiben und dieses in einen
    // Base64-String umgewandelt zurückgeben
    ms.Seek(0, SeekOrigin.Begin);
    byte[] buffer = new byte[ms.Length];
    ms.Read(buffer, 0, buffer.Length);
    ms.Close();
    return Convert.ToBase64String(buffer, 0, buffer.Length);
}

/* Methode zum Deserialisieren von Strings in ein Font-Objekt */
public static Font DeserializeFont(string fontString)
{
    // Den übergebenen Base64-String über ein Byte-Array in einen
    // MemoryStream schreiben
    byte[] buffer = Convert.FromBase64String(fontString);
    MemoryStream ms = new MemoryStream(buffer);

    // Den Stream deserialisieren und ein daraus erzeugtes Font-Objekt
    // zurückgeben
    BinaryFormatter sf = new BinaryFormatter();
    Font font = (Font)sf.Deserialize(ms);
    ms.Close();
    return font;
}

```

Listing 493: Methoden zum Serialisieren und Deserialisieren von Font-Objekten

246 Die Anzahl der Datensätze ermitteln, die eine Abfrage in einer Datenbank ergibt

Wenn Sie Daten über ein `DataReader`-Objekt abfragen, können Sie nicht ermitteln, wie viele Datensätze von der Abfrage zurückgegeben wurden, da ein `DataReader`-Objekt keine entsprechende Eigenschaft besitzt. Die Eigenschaft `RecordsAffected`, die eigentlich die Anzahl der betroffenen Datensätze (allerdings laut Dokumentation nur für Einfüge-, Aktualisierungs- und Löschaktionen) verwaltet, gibt beim Lesen von Daten immer -1 zurück.

Sie können aber recht einfach über die SQL-Funktion `COUNT` ermitteln, wie viele Datensätze von einer Abfrage betroffen sind. Idealerweise fragen Sie die Anzahl vor der eigentlichen Abfrage ab, da viele Datenbanksysteme `COUNT` nur dann in einer Abfrage mit Tabellenfeldern erlauben, wenn diese Felder in einer `GROUP BY`-Klausel angegeben werden. `COUNT` würde sich dann nicht auf die Gesamtzahl der Datensätze beziehen, sondern auf die Datensätze einer Gruppe. Das Ergebnis wäre in diesem Fall falsch.

Listing 494 zeigt das Abfragen der Anzahl der Datensätze am Beispiel der *Products*-Tabelle der *Northwind*-Datenbank im SQL Server. Da die Abfrage mit `COUNT(*)` immer nur und genau einen Datensatz ergibt, fragt das Beispiel die Anzahl über die `ExecuteScalar`-Methode eines `SqlCommand`-Objekts ab. Diese Methode ergibt den Wert der ersten Spalte des ersten Datensatzes als `object`-Typ.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System`, `System.Data` und `System.Data.SqlClient` importieren. Den Verbindungsstring, der bei der Erzeugung des `SqlConnection`-Objekts übergeben wird, müssen Sie natürlich an Ihren Server anpassen.

```
// Verbindung zur Northwind-Datenbank auf dem lokalen SQL Server aufbauen
SqlConnection connection =
    new SqlConnection("Server=(local);Database=Northwind;" +
        "Trusted_Connection=Yes");
connection.Open();

// Anzahl der Produkte abfragen, die der Kategorie 1 angehören
SqlCommand command = new SqlCommand("SELECT COUNT(*) AS Count " +
    "FROM Products WHERE CategoryId = 1", connection);

Console.WriteLine("{0} Datensätze", command.ExecuteScalar());
```

Listing 494: Abfragen der Anzahl der Datensätze, die eine Abfrage ergibt

247 Datenbanken erzeugen

ADO.NET bietet einen hervorragenden Support zur Abfrage und zur Bearbeitung von Datenbanken. Was jedoch fehlt, ist die Möglichkeit, Datenbanken und die darin enthaltenen Tabellen zu erzeugen. Dazu können Sie aber einfach SQL verwenden. Das folgende Beispiel erzeugt über SQL eine Datenbank *Bookstore* im lokalen SQL Server. Dabei werden drei Tabellen *Authors*, *Books* und *BookAuthors* erzeugt und miteinander in Beziehung gesetzt. Das Schema dieser Datenbank wird in Abbildung 174 dargestellt.

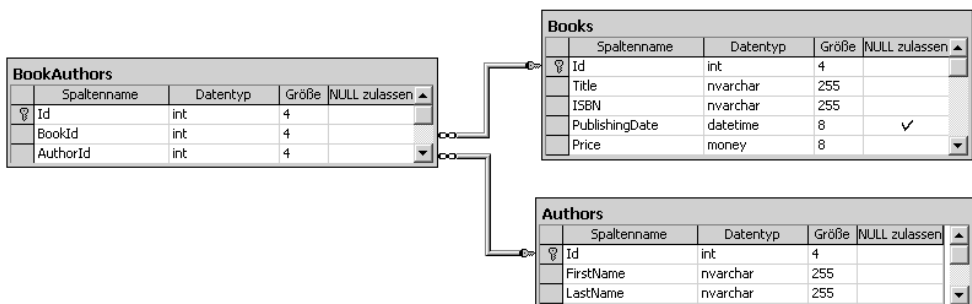


Abbildung 174: Ein SQL-Server-Diagramm der erzeugten Datenbank

Tipp

An der Adresse www.juergen-bayer.net/artikel/index.html finden Sie u.a. den Artikel »SQL/92, SQL:1999 und TSQL«, der den aktuellen SQL-Standard ausführlich beschreibt, und den Artikel »SQL-Server-2000«, der den Umgang mit dem SQL Server 2000 erläutert. In diesen Artikeln können Sie alles Wichtige zum Erzeugen und Verwalten von Datenbanken nachlesen.

Zum Kompilieren des Beispielprogramms müssen Sie die Namensräume `System`, `System.Data` und `System.Data.SqlClient` importieren. Im Verbindungsstring müssen Sie natürlich Ihren Server angeben.

Zuerst wird eine Verbindung zum SQL Server aufgebaut. Dabei geben Sie keine Datenbank an:

```
SqlConnection connection =  
    new SqlConnection("Server=(local);Trusted_Connection=Yes");  
connection.Open();
```

Listing 495: Aufbau einer Verbindung zum lokalen SQL Server

Dann wird zunächst die Datenbank angelegt. Dazu verwenden Sie die SQL-Anweisung `CREATE DATABASE`:

```
string databaseName = "Bookstore";  
string sql = "CREATE Database " + databaseName;  
SqlCommand command = new SqlCommand(sql, connection);  
command.ExecuteNonQuery();
```

Listing 496: Anlegen der Datenbank

Bei vielen Datenbanksystemen wie dem SQL Server erlaubt `CREATE DATABASE` noch spezielle Einstellungen, wie z.B. die Festlegung, in welchen Dateien die Datenbank gespeichert werden soll. Lesen Sie diese in der Dokumentation (oder für den SQL Server im oben genannten Artikel) nach. Einige wenige Datenbanksysteme wie z.B. Oracle kennen die `CREATE DATABASE`-Anweisung nicht und verwenden stattdessen `CREATE SCHEMA`.

Dann müssen Sie die Verbindung schließen und zu der neuen Datenbank wieder aufbauen:

```
connection.Close();  
connection =  
    new SqlConnection("Server=(local);Database=" + databaseName + ";" +  
        "Trusted_Connection=Yes");  
connection.Open();
```

Listing 497: Aufbau einer Verbindung zur neuen Datenbank

Nun können Sie die einzelnen Tabellen erzeugen. Dazu verwenden Sie die `CREATE TABLE`-Anweisung. Ich schreibe die SQL-Befehle zum Erzeugen der Tabellen zunächst in einzelne String-Variablen, um die Befehle später alle gemeinsam ausführen zu können:

```
string sql1 = "CREATE TABLE Authors (" +  
    "Id int NOT NULL PRIMARY KEY IDENTITY," +  
    "FirstName nvarchar(255) NOT NULL," +  
    "LastName nvarchar(255) NOT NULL)";  
  
string sql2 = "CREATE TABLE Books (" +  
    "Id int NOT NULL PRIMARY KEY IDENTITY," +  
    "Title nvarchar(255) NOT NULL," +  
    "ISBN nvarchar(255) NOT NULL," +  
    "PublishingDate datetime," +  
    "Price money NOT NULL DEFAULT 0)";  
  
string sql3 = "CREATE TABLE BookAuthors (" +  
    "Id int NOT NULL PRIMARY KEY," +  
    "BookId int NOT NULL," +  
    "AuthorId int NOT NULL)";
```

Listing 498: Erzeugen von Tabellen

In den Klammern geben Sie jeweils die zu erzeugenden Spalten an. Eine Spaltenbeschreibung beginnt mit dem Namen der Spalte gefolgt vom Datentyp. Die Datentypbezeichner sind natürlich datenbankabhängig. Der SQL Server kennt eine Vielzahl an Datentypen, die Sie in der Dokumentation oder im oben genannten Artikel nachlesen können. Die wichtigsten sind:

- ▶ `int`: 32-Bit-Integerwert,
- ▶ `float[n]`: Fließkommazahl mit 1 bis 53 Dezimalstellen, die in *n* spezifiziert werden können. Geben Sie *n* nicht an, werden 24 Dezimalstellen verwendet,
- ▶ `nvarchar(n)`: Unicode-Zeichen mit einer Maximalanzahl, die in *n* angegeben ist. Sie können bis zu 4000 Zeichen in einem solchen Feld verwalten. Da nur die tatsächlich benutzten Zeichen gespeichert werden, spielt die Angabe der Maximalgröße für den Speicherverbrauch keine Rolle,
- ▶ `datetime`: Datums-/Zeitwert mit der Basis 1.1.1900,
- ▶ `money`: Währungsdaten.

Über den Zusatz `NOT NULL` legen Sie fest, dass das Feld keine Nullwerte erlaubt. Der Zusatz `DEFAULT` kennzeichnet Defaultwerte, die bei neuen Datensätzen automatisch eingetragen werden, wenn das Feld nicht gefüllt wird. `PRIMARY KEY` legt die Spalte als Primärschlüssel fest. Über `IDENTITY` können Sie bestimmen, dass der Wert dieser Spalte automatisch beim Einfügen von neuen Datensätzen um 1 erhöht in die Spalte geschrieben wird (damit erzeugen Sie so genannte »Autowert«-Spalten).

Basics

Datum
ZeitAnwen-
dungeDatei-
system

Dateie

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Nach der Definition der SQL-Strings zum Erzeugen der Tabellen legt das Beispiel die Beziehungen zwischen diesen fest. Dazu verwenden Sie den SQL-Befehl `ALTER TABLE` mit dem Unterbefehl `ADD CONSTRAINT`:

```
string sql4 = "ALTER TABLE BookAuthors " +
    "ADD CONSTRAINT FK_Books FOREIGN KEY " +
    "(BookId) REFERENCES Books(Id)";

string sql5 = "ALTER TABLE BookAuthors " +
    "ADD CONSTRAINT FK_Authors FOREIGN KEY " +
    "(AuthorId) REFERENCES Authors(Id)";
```

Listing 499: Festlegen der Beziehungen zwischen den Tabellen

Der Zusatz `FOREIGN KEY` sagt aus, dass es sich um einen Fremdschlüssel handelt (der eben für Beziehungen gedacht ist). Hinter `FOREIGN KEY` wird die Spalte angegeben, auf die sich der Fremdschlüssel bezieht, dann folgt `REFERENCES` mit der Angabe der referenzierten Tabelle und der referenzierten Spalte.

Schließlich müssen Sie die einzelnen Strings nur noch zusammensetzen, wobei Sie beim SQL Server einen Zeilenumbruch als Trennung einbauen müssen. Die meisten anderen Datenbanksysteme erfordern ein Semikolon als Trennzeichen. Dann können Sie den SQL-Befehl ausführen (und hoffen, dass dessen Syntax korrekt ist):

```
sql = sql1 + "\r\n" + sql2 + "\r\n" + sql3 + "\r\n" +
    sql4 + "\r\n" + sql5;
command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();

connection.Close();
```

Listing 500: Ausführen der SQL-Befehle zur Erzeugung der Datenbank

248 Abfragen der automatisch vergebenen Id eines neuen Datensatzes

Viele Datenbanksysteme erlauben es, eine Spalte in einer Tabelle als Identity- oder Autowert-Spalte zu deklarieren. Meist wird dazu die Primärschlüssel-Spalte verwendet. Beim SQL Server setzen Sie dazu den Zusatz `IDENTITY` für die betreffende Spalte bei der Erzeugung der Tabelle ein. Der numerische Wert dieser Spalte wird dann beim Einfügen von Datensätzen automatisch um einen bestimmten Inkrementwert (meist 1) erhöht.

Solche Spalten sind in der Praxis immer problematisch: Wenn Datensätze in verknüpften Tabellen eingefügt werden, muss zunächst der Datensatz in der Mastertabelle erzeugt werden. Danach können Sie die Datensätze in der Detailtabelle erzeugen. Das Problem ist, dass Sie dann den Primärschlüsselwert des neuen Master-Datensatzes benötigen.

Beim SQL Server können Sie den Wert eines `IDENTITY`-Feldes nach dem Hinzufügen eines Datensatzes über eine Abfrage der globalen Variablen `@@IDENTITY` ermitteln. Als Beispiel verwende ich die Datenbank, die im Rezept 247 erzeugt wurde. Das Beispiel setzt SQL zum Einfügen des Datensatzes ein. Sie können die `@@IDENTITY`-Variable aber auch nach dem Hinzufügen eines Datensatzes über ein `DataSet` oder `DataTable` abfragen, wobei Sie natürlich darauf achten müssen, dass Sie im `DataSet`- bzw. `DataTable`-Objekt nicht gleich mehrere Master-Datensätze ändern und in die Datenbank schreiben.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.Data` und `System.Data.SqlClient` importieren. Den Verbindungsstring, der bei der Erzeugung des `SqlConnection`-Objekts übergeben wird, müssen Sie natürlich an Ihren Server anpassen.

```
// Verbindung zur Bookstore-Datenbank auf dem lokalen SQL Server aufbauen
SqlConnection connection = new SqlConnection("Server=(local);" +
    "Database=Bookstore;Trusted_Connection=Yes");
connection.Open();

// Autor hinzufügen
string sql = "INSERT INTO Authors (FirstName, LastName) " +
    "VALUES ('Matt', 'Ruff')";
SqlCommand command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();

// Den Wert der @@IDENTITY-Variablen auslesen
sql = "SELECT @@IDENTITY";
command = new SqlCommand(sql, connection);
int identityValue = Convert.ToInt32(command.ExecuteScalar());

Console.WriteLine("Id des neuen Autors: {0}", identityValue);
```

Listing 501: Abfragen der `@@IDENTITY`-Variable zur Ermittlung der Id eines neuen Datensatzes

Andere Datenbanksysteme kennen u.U. auch eine ähnliche Variable, die Sie genauso einfach abfragen können. Lesen Sie in der Dokumentation des Datenbanksystems nach.

249 Bilder und andere binäre Daten in einer Datenbank verwalten

Die meisten Datenbanksysteme kennen einen Feldtyp, der es erlaubt, binäre Daten zu speichern. Beim SQL Server heißt dieser Typ (etwas irreführend) `image`. Im Allgemeinen werden solche Felder allerdings als `BLOB`-Felder (Binary Large Object) bezeichnet.

In einem `BLOB`-Feld können Sie beliebige binäre Daten ablegen. Dazu schreiben Sie einfach ein `Byte`-Array in das Feld. Beim Lesen erhalten Sie ebenfalls ein `Byte`-Array zurück, das Sie dann über ein `Stream`-Objekt in eine Datei oder – wenn es sich um ein Bild handelt – in ein `Bitmap`-Objekt schreiben können.

Basics

Datum
ZeitAnwei-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

Das Beispiel ab Listing 502 zeigt, wie Sie damit umgehen. Das Beispiel basiert auf einer Datenbank, die folgendermaßen erzeugt wird:

```
// Verbindung zum lokalen SQL Server aufbauen und Datenbank erzeugen
SqlConnection connection = new SqlConnection("Server=(local);" +
    "Trusted_Connection=Yes");
connection.Open();

string sql = "CREATE DATABASE Persons";
SqlCommand command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();
connection.Close();

connection = new SqlConnection("Server=(local);" +
    "Database=Persons;Trusted_Connection=Yes");
connection.Open();

sql = "CREATE TABLE Persons (Id int PRIMARY KEY " +
    "NOT NULL IDENTITY, FirstName nvarchar(255), " +
    "LastName nvarchar(255), Picture image)";
command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();
```

Listing 502: Erzeugen der Beispieldatenbank

Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System`, `System.Data`, `System.Data.SqlClient`, `System.Drawing`, `System.Drawing.Imaging` und `System.IO` einbinden.

Die Datenbank besteht lediglich aus einer Tabelle mit den Feldern *Id*, *FirstName*, *LastName* und *Picture*. Das Feld *Id* wird beim Hinzufügen von Datensätzen automatisch um 1 erhöht. Das Feld *Picture* ist vom Typ `image` und verwaltet deswegen binäre Daten.

Datensätze hinzufügen können Sie vereinfacht über ein `DataSet`- oder ein `DataTable`-Objekt. Der Grund dafür liegt darin, dass dies die binären Daten automatisch korrekt umsetzt. In einem SQL-Befehl müssten Sie ansonsten erst herausfinden, wie binäre Daten dargestellt werden müssen.

Das folgende Beispiel liest eine Bitmap-Datei ein und speichert diese mit dem Vor- und dem Nachnamen eines Autors in der *Persons*-Tabelle:

```
// Bitmap einlesen
string fileName = "C:\\Bilder\\Zaphod.bmp";
Bitmap bitmap = new Bitmap(fileName);

// Verbindung zur Persons-Datenbank auf dem lokalen SQL Server aufbauen
SqlConnection connection = new SqlConnection("Server=(local);" +
    "Database=Persons;Trusted_Connection=Yes");
connection.Open();
```

Listing 503: Speichern von binären Daten in einer Datenbanktabelle

```
// Das Tabellen-Schema in ein DataTable-Objekt lesen
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Persons",
    connection);
new SqlCommandBuilder(adapter);
DataTable dataTable = new DataTable();
adapter.FillSchema(dataTable, SchemaType.Source);

// DataRow erzeugen und mit den Textdaten füllen
DataRow newRow = dataTable.NewRow();
newRow["FirstName"] = this.authorFirstNameTextBox.Text;
newRow["LastName"] = this.authorLastNameTextBox.Text;

// Das Bitmap in ein Byte-Array umwandeln und speichern
MemoryStream ms = new MemoryStream();
bitmap.Save(ms, ImageFormat.Bmp);
byte[] buffer = ms.ToArray();
newRow["Picture"] = buffer;

// Datensatz anfügen und Tabelle aktualisieren
dataTable.Rows.Add(newRow);
adapter.Update(dataTable);
```

Listing 503: Speichern von binären Daten in einer Datenbanktabelle (Forts.)

Beim Füllen des `DataTable`-Objekts nutze ich nicht die `Fill`-Methode, sondern `FillSchema`. Damit werden nicht die Datensätze eingelesen (was wir hier nicht benötigen), sondern nur das Schema der Tabelle. Die Performance wird damit erheblich erhöht.

Das Bild wird dann über ein `MemoryStream`-Objekt in ein `Byte-Array` und dieses in die neue Zeile geschrieben. Und schon ist das Bild, nach dem Aufruf der `Update`-Methode des `DataAdapter`-Objekts, in der Datenbank.

Beim Lesen erhalten Sie wieder ein `Byte-Array`, das Sie entsprechend umsetzen müssen. Das folgende Beispiel ist in einem Windows-Formular implementiert. Es liest die Daten der Person ein, deren Id in der `TextBox` `personIdTextBox` eingegeben wurde, und gibt den Namen im `Label` `infoLabel` und das Bild in der `PictureBox` `personPictureBox` aus.

Zum Lesen der binären Bilddaten setzt der Programmcode ein `MemoryStream`-Objekt ein, das mit dem eingelesenen `Byte-Array` initialisiert wird. Über diesen Stream wird dann ein `Bitmap`-Objekt erzeugt und in einer `PictureBox` dargestellt.

```
// Verbindung zur Persons-Datenbank auf dem lokalen SQL Server aufbauen
SqlConnection connection = new SqlConnection("Server=(local);" +
    "Database=Persons;Trusted_Connection=Yes");
connection.Open();

// Person einlesen
int personId = Convert.ToInt32(this.personIdTextBox.Text);
```

Listing 504: Lesen von binären Daten

Basics

Datum
ZeitAnwen-
dungenDatei-
system

Datei

XML

System

Interne

Formu-
lare

Benutz

Multi-
mediaAutom-
tionSons-
tiges

```

string sql = "SELECT * FROM Persons WHERE Id = " + personId;
SqlCommand command = new SqlCommand(sql, connection);
SqlDataReader reader = command.ExecuteReader();
if (reader.Read())
{
    // Textdaten einlesen
    infoLabel.Text = (string)reader["FirstName"] + " " +
        (string)reader["LastName"];

    // Binäre Daten einlesen
    byte[] imageData = (byte[])reader["Picture"];
    if (imageData != null)
    {
        MemoryStream ms = new MemoryStream(imageData, 0, imageData.Length);
        this.personPictureBox.Image = new Bitmap(ms);
        ms.Close();
    }
    else
        this.personPictureBox.Image = null;
}
else
{
    MessageBox.Show("Person nicht gefunden", Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

Listing 504: Lesen von binären Daten (Forts.)

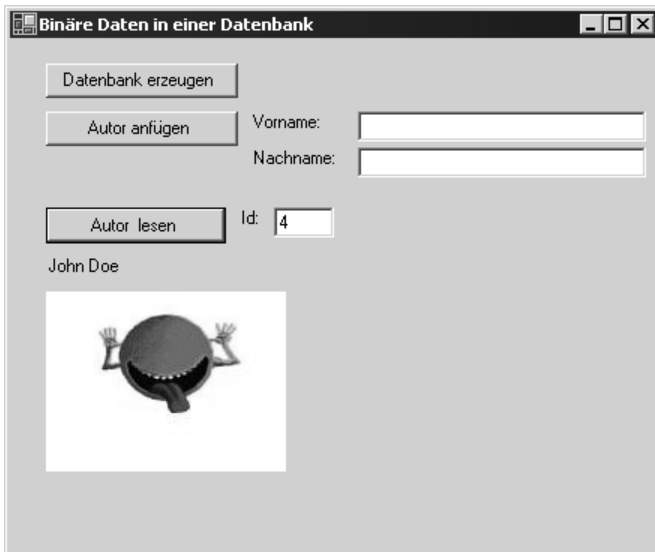
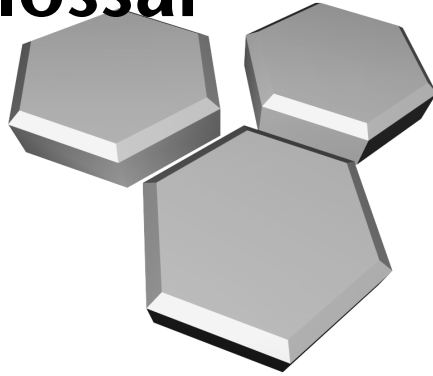


Abbildung 175: Das Beispielprogramm hat die Daten und das Bild eines Autors gelesen

TEIL III

Glossar



Referenz

1 Sprachreferenz

Verzweigungen und Schleifen

if

```
if (Bedingung)  
    Anweisungsblock1  
[else  
    Anweisungsblock2]
```

switch

```
switch (Prüfausdruck)  
{  
    case Wert1:  
        /* Anweisungen für den Fall 1 */  
        Sprunganweisung;  
  
    [case Wert2:  
        /* Anweisungen für den Fall 2 */  
        Sprunganweisung;]  
  
    [...]  
  
    [default:  
        /* Anweisungen, die ausgeführt werden sollen,  
         * wenn keiner der Fälle eingetreten ist*/  
        Sprunganweisung;]  
}
```

while

```
while (Bedingung)  
{  
    Anweisungen  
}
```

do

```
do
    Anweisungen
while (Bedingung);
```

for

```
for ([Initialisierungsausdruck]; [Bedingung]; [Zählausdruck])
    Anweisungen
```

Sprunganweisungen

Sprunganweisung	Bedeutung
Break	Sprung aus dem Block
Continue	Sprung zum nächsten Fall
Default	Sprung zum default-Block
goto <i>Label</i>	Sprung zu einem Label, das am Anfang einer Zeile mit <i>Label</i> : definiert ist. Verwenden Sie diese veraltete Anweisung besser nicht.
Return	Sprung aus der Methode heraus

Tabelle 1: Die C#-Sprunganweisungen

Escape-Sequenzen

Escape-Sequenz	Bedeutung
\'	Apostroph
\«	Anführungszeichen
\\	Backslash
\0	Null-Zeichen
\a	Warnung (alert). Erzeugt einen Piepston.
\b	Backspace
\f	Seitenvorschub (Form Feed)
\n	Zeilenumbruch (New Line)
\r	Wagenrücklauf (Carriage Return)
\t	horizontaler Tabulator
\v	vertikaler Tabulator

Tabelle 2: Die C#-Escape-Sequenzen

Datentypen

Die Standarddatentypen

C#-Datentyp	CLR-Datentyp	Größe	Wertebereich
Sbyte	SByte	8 Bit	-128 bis 127
Byte	Byte	8 Bit	0 bis 255
Char	Char	16 Bit	ein beliebiges Unicode-Zeichen
Short	Int16	16 Bit	-32.768 bis 32.767
Ushort	UInt16	16 Bit	0 bis 65.535
Int	Int32	32 Bit	-2.147.483.648 bis 2.147.483.647
UInt	UInt32	32 Bit	0 bis 4.294.967.295
Long	Int64	64 Bit	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807
Ulong	UInt64	64 Bit	0 bis 18.446.744.073.709.551.615
Float	Single	32 Bit	$\pm 1.5 \times 10^{-45}$ bis $\pm 3.4 \times 10^{38}$ mit maximal 7 Dezimalstellen
Double	Double	64 Bit	$\pm 5.0 \times 10^{-324}$ bis $\pm 1.7 \times 10^{308}$ mit maximal 15-16 Dezimalstellen
decimal	Decimal	128 Bit	1.0×10^{-28} bis 7.9×10^{28} mit maximal 28-29 Dezimalstellen
Bool	Boolean	8 Bit	true oder false
String	String	variabel	beliebige Unicode-Zeichenketten

Tabelle 3: Die C#-Standarddatentypen

Typkennzeichen für Literale

Suffix	Datentyp
L	long
U	unit oder ulong, je nach Größe des Werts
UL	ulong
D	double
F	float
M	decimal

Tabelle 4: Die Suffixe für C#-Literele

Zeichen für Formatierungen

Zahlformatierungen

Format	Erläuterung	Beispielwert	Ergebnis
C	Währung mit Standard-Dezimalstellen	100.955	»100,96 €«
C4	Währung mit festen Dezimalstellen	100.955	»100,9550 €«
F2	Zahl mit festen Dezimalstellen	100.955	»100,96«
F4	Zahl mit festen Dezimalstellen	100.955	»100,9550«
X	hexadezimaleres Format	255	»FF«
0.00	benutzerdefiniert	0.345	»0,35«
0.00	benutzerdefiniert	12.345	»12,35«
0.00	benutzerdefiniert	1234.567	»1234,57«
#,#0.00	benutzerdefiniert	1234.567	»1.234,57«
#,#.00	benutzerdefiniert	0.567	»,57«

Tabelle 5: Die wichtigsten Zahlformatierungen

Datumsformatierungen

Format	Erläuterung	Beispiel-Ergebnis
D	langes Datumsformat	»Samstag, 31. Dezember 2005«
D	kurzes Datumsformat	»31.12.2005«
G	kurzes Datumsformat mit kurzer Zeit	»31.12.2005 12:30«
T	kurzes Zeitformat	»12:30«
T	langes Zeitformat	»12:30:00«
U	ISO-Format	»2005-12-31 12:30:00Z«
dd.MM.yy	benutzerdefiniert	»31.12.05«
dddd, dd. MMMM yyyy	benutzerdefiniert	»Samstag, 31. Dezember 2005«
yyyy-MM-dd	benutzerdefiniert	»2005-12-31«
hh:mm:ss	benutzerdefiniert	»12:30:00«

Tabelle 6: Die wichtigsten Datumsformate

Operatoren

Arithmetische Operatoren

Operator	Bedeutung
()	Klammern; Verschieben der Rechenpriorität
+	Addition zweier Operanden
++	Addition eines Operanden mit 1
-	Subtraktion zweier Operanden
--	Subtraktion von 1 von einem Operanden
*	Multiplikation
/	Division
%	Modulo-Division

Tabelle 7: Die arithmetischen Operatoren von C#

Bitweise Operatoren

Operator	Bedeutung
&	AND
	OR
^	XOR
~	Komplementär-Operation
<<	Linksverschiebung
>>	Rechtsverschiebung

Tabelle 8: Die bitweisen Operatoren von C#

Zuweisungsoperatoren

Operator	Bedeutung
=	Einfache Zuweisung
+=	Additionszuweisung. $i += 1$ entspricht $i = i + 1$.
-=	Subtraktionszuweisung. $i -= 1$ entspricht $i = i - 1$.
*=	Multiplikationszuweisung. $i *= 1$ entspricht $i = i * 1$.
/=	Divisionszuweisung. $i /= 1$ entspricht $i = i / 1$.
%=	Modulo-Divisionszuweisung. $i \% = 1$ entspricht $i = i \% 1$.
&=	Bitweise AND-Zuweisung. $i \&= 1$ entspricht $i = i \& 1$.
=	Bitweise OR-Zuweisung. $i = 1$ entspricht $i = i 1$.

Tabelle 9: Die Zuweisungsoperatoren von C#

Operator	Bedeutung
<code>^=</code>	Bitweise XOR-Zuweisung. <code>i ^= 1</code> entspricht <code>i = i ^ 1</code> .
<code><<=</code>	Linksverschiebungszuweisung. <code>i <<= 1</code> entspricht <code>i = i << 1</code> .
<code>>>=</code>	Rechtsverschiebungszuweisung. <code>i >>= 1</code> entspricht <code>i = i >> 1</code> .

Tabelle 9: Die Zuweisungsoperatoren von C# (Forts.)

Vergleichsoperatoren

Operator	Beschreibung	gilt für Datentyp
<code>==</code>	Gleichheit	alle
<code>!=</code>	Ungleichheit	alle
<code><</code>	Kleiner	nur für numerische Typen und Aufzählungen (Enums)
<code>></code>	Größer	nur für numerische Typen und Aufzählungen
<code><=</code>	Kleiner/Gleich	nur für numerische Typen und Aufzählungen
<code>>=</code>	Größer/Gleich	nur für numerische Typen und Aufzählungen

Tabelle 10: Die Vergleichsoperatoren von C#

Logische Operatoren

Operator	Beschreibung
<code>&&</code>	Logisches AND. Der rechte und der linke Ausdruck müssen <code>true</code> ergeben, damit der Gesamtausdruck <code>true</code> ergibt.
<code> </code>	Logisches Or. Wenn einer der Ausdrücke <code>true</code> ergibt, resultiert <code>true</code> für den Gesamtausdruck.
<code>!</code>	Logisches Not. Der Ausdruck, der rechts von diesem Operator steht, wird negiert. Aus <code>true</code> wird <code>false</code> und aus <code>false</code> wird <code>true</code> .

Tabelle 11: Die logischen Operatoren von C#

Spezielle Operatoren

Operator	Beschreibung
<code>?:</code>	Über den Bedingungsoperator <code>?:</code> können Sie einen Ausdruck schreiben, der eine Bedingung überprüft und jeweils einen anderen Wert ergibt, wenn die Bedingung wahr oder falsch wird: <i>Bedingung</i> ? <i>Ausdruck1</i> : <i>Ausdruck2</i>
<code>typeof(Typ)</code>	ermittelt Typinformationen zu einem Typ in Form eines <code>Type</code> -Objekts.
<code>is</code>	überprüft, ob der linke Operand denselben Typ besitzt wie der rechte.

Tabelle 12: Die speziellen C#-Operatoren

2 Referenz der wichtigsten Basisklassen

object

Methoden

Methode	Beschreibung
<code>bool Equals(object obj)</code>	vergleicht die Daten eines Objekts mit den Daten eines anderen Objekts auf Gleichheit (d.h. dass die in den Objekten gespeicherten Werte gleich sind).
<code>int GetHashCode()</code>	ermittelt einen Hash-Code für das Objekt. Ein Hash-Code ist ein aus den Daten des Objekts ermittelter Code, der das Objekt in verkürzter Form identifiziert. Dieser Code wird beispielsweise in einer Auflistung gespeichert, die in der Lage ist, die Elemente ihrer Liste mit einem Schlüssel zu assoziieren. Statt den eigentlichen Wert des Schlüssels in der Liste zu speichern, verwendet die Auflistung den Hashcode des als Schlüssel übergebenen Typs. So können dann auch beliebige Objekte als Schlüssel verwendet werden.
<code>Type GetType()</code>	ermittelt den Typ des Objekts. Gibt ein Objekt der Klasse <code>Type</code> zurück. Aus diesem Objekt können Sie sehr viele Informationen auslesen. Die Eigenschaft <code>Name</code> liefert z.B. den Namen des Typs, die Eigenschaft <code>Namespace</code> gibt den Bezeichner des Namensraums zurück, in dem der Typ verwaltet wird.
<code>string ToString()</code>	Diese Methode gibt die Daten des Objekts als Zeichenkette zurück.

Tabelle 13: Die Instanzmethoden des Basistyps `object`

Zusätzliche Methoden und Eigenschaften der Standarddatentypen

Instanzmethoden

Methode	Beschreibung
<code>int CompareTo(object value)</code>	vergleicht einen Typ mit einem anderen. Rückgabe: < 0 : der Typ ist kleiner als der andere, 0 : beide Typen sind gleich, > 0 : der Typ ist größer als der andere.
<code>TypeCode GetTypeCode()</code>	ermittelt den Datentyp des Typs. Gibt einen Wert des Typs <code>TypeCode</code> zurück. <code>TypeCode</code> ist eine Aufzählung von Konstanten, die so benannt sind wie die Typen. <code>TypeCode.Byte</code> definiert z.B. einen <code>Byte</code> -Typ.

Tabelle 14: Die zusätzlichen Methoden der C#-Standarddatentypen

Klasseneigenschaften und -methoden

Eigenschaft/Methode	Beschreibung
<code>MinValue</code>	liefert den kleinsten speicherbaren Wert.
<code>MaxValue</code>	liefert den größten speicherbaren Wert.

Tabelle 15: Die wichtigsten Klasseneigenschaften und -methoden der Standarddatentypen

Eigenschaft/Methode	Beschreibung
Typ Parse(string s [...])	Über diese Methode können Sie einen String in den Typ umwandeln, auf den Sie die Methode anwenden. Parse erlaubt bei numerischen Typen zusätzlich die Angabe der zu verwendenden Kultur.
bool TryParse(string s , NumberStyles style , IFormatProvider provider , out double result)	Über diese Methode können Sie überprüfen, ob eine Umwandlung eines Strings in einen speziellen Typ möglich ist.

Tabelle 15: Die wichtigsten Klasseneigenschaften und -methoden der Standarddatentypen (Forts.)

Convert-Klasse

Methode	konvertiert in
bool ToBoolean(...)	Bool
byte ToByte(...)	Byte
char ToChar(...)	Char
DateTime ToDateTime(...)	DateTime
decimal ToDecimal(...)	Decimal
double ToDouble(...)	Double
short.ToInt16(...)	Short
int.ToInt32(...)	Int
long.ToInt64(...)	Long
sbyte ToSByte(...)	Sbyte
float ToSingle(...)	Float
string ToString(...)	String
ushort.ToInt16(...)	Ushort
uint.ToInt32(...)	Uint
ulong.ToInt64(...)	Ulong
byte[] FromBase64String(string s)	konvertiert den übergebenen String in ein Byte-Array und nimmt dabei eine Base-64-Decodierung vor.
string ToBase64String(byte[] inArray) string ToBase64String(byte[] inArray, int offset, int length);	konvertiert die im übergebenen Byte-Array definierten Zeichen in einen Base-64-codierten String.
bool IsDBNull(object value)	überprüft, ob der übergebene Wert dem Nullwert entspricht, der in leeren Feldern von Datenbanktabellen gespeichert ist.

Tabelle 16: Die wichtigsten Klassenmethoden der Klasse Convert

Array-Klasse

Instanzeigenschaften und -methoden

Eigenschaft / Methode	Beschreibung
<code>object Array Clone()</code>	erzeugt eine Kopie eines Arrays.
<code>void CopyTo(Array array , int index)</code>	kopiert den Inhalt eines Arrays in ein anderes Array. Am Argument <code>index</code> geben Sie den Index des Zielarrays an, ab dem das Kopieren beginnen soll.
<code>int GetLength(int dimension)</code>	ermittelt die Anzahl der Elemente in einer gegebenen Dimension.
<code>void Initialize()</code>	Über diese Methode können Sie die einzelnen Elemente eines Arrays auf einen Standardwert (in der Regel ein Leerwert) initialisieren. <code>Initialize</code> ruft dazu einfach den Standardkonstruktor der gespeicherten Typen auf.
<code>Length</code>	Diese Eigenschaft gibt die Gesamtanzahl der gespeicherten Elemente zurück.
<code>Rank</code>	Über diese Eigenschaft können Sie die Gesamtanzahl der Dimensionen ermitteln.

Tabelle 17: Die wichtigsten Instanzeigenschaften und -methoden der Array-Klasse

Klassenmethoden

Methode	Beschreibung
<code>int BinarySearch(Array array, object value [, IComparer comparer])</code>	Über diese Methode können Sie die Position eines Wertes im Array über die schnelle binäre Suche ermitteln. Das Array muss dazu sortiert sein. Wenn Sie am dritten Argument einen Index angeben, durchsucht die Methode ab diesem Index die am Argument <code>length</code> angegebene Anzahl Elemente.
<code>int BinarySearch(Array array, int index, int length, object value [, IComparer comparer])</code>	Geben Sie keinen Index an, wird das ganze Array durchsucht. Am letzten Argument können Sie ein Objekt übergeben, das die Schnittstelle <code>IComparer</code> implementiert, über deren <code>Compare</code> -Methode der Vergleich der Elemente erfolgt. So können Sie benutzerdefinierte Suchen programmieren. Die Standarddatentypen implementieren diese Schnittstelle bereits, weswegen Sie diese auch ohne <code>IComparer</code> -Objekt durchsuchen können.
<code>int IndexOf(Array array, object value [, int startIndex [, int count])</code>	Über <code>IndexOf</code> können Sie ein Array sequenziell (und damit langsam) durchsuchen.

Tabelle 18: Die wichtigsten Klassenmethoden der Array-Klasse

Methode	Beschreibung
<code>int LastIndexOf(Array array, object value [, int startIndex] [, int count])</code>	<code>LastIndexOf</code> durchsucht ein Array sequenziell, ermittelt aber den Index des letzten gespeicherten Werts, der zum Suchwert identisch ist.
<code>Array Sort(Array array [, int index , int length] [, IComparer comparer])</code> <code>Array Sort(Array array, IComparer comparer)</code> <code>Array Sort(Array keys, Array items [, int index, int length] [, IComparer comparer])</code> <code>Array Sort(, Array keys, Array items, IComparer comparer)</code>	Über diese Methode können Sie ein Array ganz oder teilweise sortieren. Die gespeicherten Typen müssen dazu die <code>IComparer</code> -Schnittstelle implementieren, was bei den meisten Typen bereits der Fall ist. Für eine benutzerdefinierte Sortierung können Sie ein separates Objekt übergeben, das diese Schnittstelle implementiert.

Tabelle 18: Die wichtigsten Klassenmethoden der `Array`-Klasse (Forts.)

String-Klasse

Instanzeigenschaften

Eigenschaft	Beschreibung
<code>Length</code>	liefert die Länge des Strings.

Tabelle 19: Die einsame Instanzeigenschaft der `string`-Klasse von C#

Instanzmethoden

Methode	Beschreibung
<code>bool EndsWith(string value)</code>	ermittelt, ob der String mit dem am Argument <code>value</code> übergebenen String endet.
<code>int IndexOf({string char} value [, int startIndex] [, int count])</code>	Mit dieser Methode können Sie den Index eines Zeichens oder Strings ermitteln. Wenn Sie <code>startIndex</code> angeben, wird ab diesem Index gesucht, ansonsten ab dem ersten Zeichen. Die Indizierung beginnt, wie in C# üblich, bei 0. Geben Sie <code>count</code> an, werden nur die Anzahl Zeichen durchsucht, die Sie hier angegeben haben. Wird das Zeichen bzw. der String nicht gefunden, resultiert der Wert -1.

Tabelle 20: Die wichtigsten Instanzmethoden der `string`-Klasse

Methode	Beschreibung
<code>int IndexOfAny(char[] anyOf[, int startIndex[, int count]])</code>	Über diese Methode können Sie den Index eines von mehreren Zeichen ermitteln, die Sie am ersten Argument als <code>char</code> -Array übergeben.
<code>string Insert(int startIndex, string value)</code>	fügt einen String an der angegebenen Position ein und gibt diesen String zurück.
<code>int LastIndexOf({string char} value[, int startIndex[, int count]])</code>	ermittelt ähnlich <code>IndexOf</code> den Index eines Zeichens oder eines Strings. <code>LastIndexOf</code> ermittelt aber den Index des letzten Vorkommens.
<code>int LastIndexOfAny(char[] anyOf[, int startIndex[, int count]])</code>	ermittelt ähnlich <code>IndexOfAny</code> den Index eines von mehreren Zeichen. <code>LastIndexOfAny</code> ermittelt aber den Index des letzten Vorkommens.
<code>string PadLeft(int totalWidth[, char paddingChar])</code>	füllt einen String nach links mit Leerzeichen oder dem am Argument <code>paddingChar</code> angegebenen Zeichen auf, bis die an <code>totalWidth</code> angegebene Länge erreicht ist, und gibt diesen String zurück.
<code>string PadRight(int totalWidth[, char paddingChar])</code>	füllt einen String nach rechts auf.
<code>string Remove(int startIndex, int count)</code>	gibt den um einen Teilstring reduzierten String zurück.
<code>string Replace(char oldChar, char newChar)</code> <code>string Replace(string oldValue, string newValue)</code>	ersetzt alle Vorkommen eines Zeichens oder Strings in einem String durch ein anderes Zeichen bzw. einen anderen String und gibt diesen String zurück.
<code>string[] Split(char[] separator, int count)</code> <code>string[] Split(params char[] separator)</code>	splittet einen String an den am ersten Argument übergebenen Trennzeichen auf. <code>Split</code> gibt ein Array aus Strings mit den ermittelten Teilstrings zurück. So können Sie einen String sehr einfach in einzelne Teilstrings zerlegen. Am ersten Argument übergeben Sie ein Array aus <code>char</code>-Werten mit den möglichen Trennzeichen. Alternativ können Sie hier auch ein leeres Array oder <code>null</code> übergeben. C# verwendet dann ein Komma und verschiedene Escape-Sequenzen (<code>\n</code>, <code>\r</code>, <code>\t</code>) als Trennzeichen. Am zweiten Argument können Sie festlegen, wie viele Teilstrings maximal zurückgegeben werden.
<code>bool StartsWith(string value)</code>	ermittelt, ob der String mit dem am Argument <code>value</code> übergebenen String beginnt.
<code>string Substring(int startIndex[, int length])</code>	ermittelt einen Teilstring. Wenn Sie <code>length</code> angeben, wird der Teilstring ab dem an <code>startIndex</code> angegebenen Index mit der angegebenen Länge zurückgegeben, ansonsten bis zum Ende des Strings.

Tabelle 20: Die wichtigsten Instanzmethoden der `string`-Klasse (Forts.)

Methode	Beschreibung
<code>char[] ToCharArray([int startIndex, int length]);</code>	gibt den (Teil-)String als <code>char</code> -Array zurück.
<code>string ToLower([CultureInfo culture])</code> <code>string ToUpper([CultureInfo culture])</code>	gibt den String in Klein- bzw. Großschreibung zurück. Sie können ein Objekt der Klasse <code>System.Globalization.CultureInfo</code> übergeben, um die bei der Konvertierung zu verwendende Kultur zu definieren (in verschiedenen Kulturen werden Zeichen unterschiedlich konvertiert). Übergeben Sie kein Objekt, verwendet <code>ToLower</code> die aktuelle Kultur.
<code>string Trim([params char[] trimChars])</code> <code>string TrimEnd([params char[] trimChars])</code> <code>string TrimStart([params char[] trimChars])</code>	Über diese Methode können Sie bestimmte Zeichen aus dem String entfernen, die rechts und/oder links stehen. Übergeben Sie keine Zeichen, werden alle Whitespace-Zeichen (das sind Zeichen, die bei der Ausgabe nicht sichtbar sind) entfernt. Zu diesen Zeichen gehört z. B. das Leerzeichen, aber auch das Carriage Return. Alternativ können Sie am ersten Argument ein Array mit Zeichen übergeben, die entfernt werden sollen.

Tabelle 20: Die wichtigsten Instanzmethoden der `string`-Klasse (Forts.)

Klassenmethoden

Methode	Beschreibung
<code>int Compare(string strA, string strB, [bool ignoreCase], [CultureInfo culture])</code> <code>int Compare(string strA, int indexA, string strB, int indexB, [int length], [bool ignoreCase], [CultureInfo culture])</code>	vergleicht einen String mit einem anderen. Per Voreinstellung wird dabei Groß- und Kleinschreibung berücksichtigt (binärer Vergleich). Wenn Sie am dritten Argument <code>true</code> angeben, erfolgt der Vergleich ohne Berücksichtigung der Groß- und Kleinschreibung. In einer anderen Variante können Sie neben den zu vergleichenden Strings noch einen Index angeben, ab dem verglichen werden soll. Die Rückgabe entspricht der bei allgemeinen Vergleichsmethoden: Ein Wert kleiner als 0 bedeutet, dass String <i>A</i> kleiner ist als String <i>B</i> . Bei 0 sind beide Strings gleich groß. Ein Wert größer als 0 bedeutet, dass String <i>A</i> größer ist als String <i>B</i> . Beim Vergleich werden Kulturinformationen berücksichtigt. Wenn Sie nicht die Systeminformationen verwenden wollen, können Sie am letzten Argument spezielle Kulturinformationen angeben.
<code>int CompareOrdinal(string strA, string strB)</code> <code>int CompareOrdinal(string strA, int indexA, string strB, int indexB, int length)</code>	vergleicht ähnlich <code>Compare</code> , aber ohne Berücksichtigung der Kultur und mit Berücksichtigung der Groß-/Kleinschreibung (rein binärer Vergleich).

Tabelle 21: Die wichtigsten Klassenmethoden der `string`-Klasse

Methode	Beschreibung
<code>string Copy(string str)</code>	gibt eine neue Instanz eines Strings mit der Kopie des übergebenen Strings zurück. Beachten Sie, dass eine einfache Zuweisung wie <code>s1 = s2</code> zunächst keine neue Instanz erzeugt, sondern beide Referenzen auf dieselbe Instanz zeigen lässt (erst bei Veränderung eines der beiden Strings wird eine separate Instanz erzeugt). <code>Copy</code> erzeugt dagegen immer eine neue Instanz.
<code>string Format(string format, object arg0 [, object arg1] [, object arg2])</code> <code>string Format(string format, params object[] args)</code> <code>string Format(IFormatProvider provider, string format, params object[] args)</code>	formatiert den am Argument <code>format</code> angegebenen String unter Berücksichtigung der Werte, die an <code>args0</code> , <code>args1</code> und <code>args2</code> bzw. an <code>args</code> (als Array) angegeben sind. Alternativ können Sie am ersten Argument einen Format-Provider übergeben.
<code>string Join(string separator, string[] value [, int startIndex, int count])</code>	<code>Join</code> erzeugt aus einem String-Array einen einzelnen String. Die im Array gespeicherten Strings werden durch den angegebenen Separator getrennt. Über <code>startIndex</code> können Sie einen Index angeben, ab dem das Array berücksichtigt werden soll. <code>count</code> gibt dann die Anzahl der Elemente an, die im String erscheinen sollen.

Tabelle 21: Die wichtigsten Klassenmethoden der `string`-Klasse (Forts.)

DateTime-Struktur

Instanzmethoden

Methode	Beschreibung
<code>DateTime Add(TimeSpan value)</code> <code>DateTime AddTicks(long value)</code> <code>DateTime AddMilliseconds(double value)</code> <code>DateTime AddSeconds(double value)</code> <code>DateTime AddMinutes(double value)</code>	gibt ein um die angegebene Anzahl der jeweiligen Einheit addiertes Datum als <code>DateTime</code> -Objekt zurück. Sie können auch negative Werte übergeben um einem Datum einen Wert abzuziehen.

Tabelle 22: Die wichtigsten Instanzmethoden der `DateTime`-Struktur

Methode	Beschreibung
DateTime AddHours(double value) DateTime AddDays(double value) DateTime AddMonths(double value) DateTime AddYears(double value)	
string ToLongDateString() string ToLongTimeString() string ToShortDateString() string ToShortTimeString()	Diese Methoden geben das Datum als kurzen oder langen Datums-/ Zeitstring zurück.

Tabelle 22: Die wichtigsten Instanzmethoden der DateTime-Struktur (Forts.)

Instanzeigenschaften

Eigenschaft	Bedeutung
Day, Hour, Minute, Month, Second, Year	Über diese Eigenschaften können Sie die Teile des Datums ermitteln.
DayOfWeek	gibt den Wochentag als DayOfWeek-Typ zurück. DayOfWeek ist eine Aufzählung mit den Werten Sunday (0), Monday (1), Tuesday (2), Wednesday (3), Thursday (4), Friday (5) und Saturday (6).
DayOfYear	gibt die Nummer des Tages bezogen auf das Jahr als int-Wert zurück.

Tabelle 23: Die wichtigsten Instanzeigenschaften der DateTime-Struktur

Klassenmethoden

Methode	Beschreibung
int DaysInMonth(int year, int month)	ermittelt die Anzahl der Tage eines Monats.
bool IsLeapYear(int year)	ermittelt, ob ein Jahr ein Schaltjahr ist.
DateTime Parse(string s [...])	Über diese Methode können Sie einen String in einen DateTime-Wert umwandeln. Neben dem String können Sie an den zusätzlichen Argumenten die zu verwendenden Kulturinformationen des Datums definieren. Das Datum darf in einer beliebigen, zur Kultur passenden Form übergeben werden.

Tabelle 24: Die wichtigsten Klassenmethoden der DateTime-Struktur

Methode	Beschreibung
<code>DateTime.ParseExact(string s, string format, ...)</code> <code>DateTime.ParseExact(string s, string[] formats, ...)</code>	<code>ParseExact</code> wandelt einen String in einen <code>DateTime</code> -Wert um. Der String muss dem am zweiten Argument übergebenen Format entsprechen. Diese Methode erwartet an den weiteren Argumenten Informationen zur verwendeten Kultur.

Tabelle 24: Die wichtigsten Klassenmethoden der `DateTime`-Struktur (Forts.)

Klasseneigenschaften

Eigenschaft	Bedeutung
<code>Now</code>	gibt ein <code>DateTime</code> -Objekt mit dem aktuellen Datum inklusive Zeit zurück.
<code>Today</code>	gibt ein <code>DateTime</code> -Objekt mit dem aktuellen Datum ohne Zeit zurück.

Tabelle 25: Die wichtigsten Klasseneigenschaften der `DateTime`-Struktur

Math-Klasse

Methode	Beschreibung
<code>Abs</code>	liefert den Absolutwert (Zahl ohne Vorzeichen) der übergebenen Zahl.
<code>Cos</code> , <code>Sin</code> , <code>Tan</code> , <code>Acos</code> , <code>Asin</code> , <code>Atan</code>	berechnen Kosinus, Sinus, Tangens, Arcuskosinus, Arcussinus und Arcustangens. Das übergebene Argument wird nicht in Grad, sondern in Bogenmaß übergeben. Ein Bogenmaß entspricht: $\text{Grad} * \text{Math.PI} / 180$.
<code>Exp</code>	liefert den Exponenten einer Zahl (Umkehrfunktion zu <code>Log</code>).
<code>Log</code>	liefert den natürlichen Logarithmus der übergebenen Zahl.
<code>Max</code> , <code>Min</code>	liefern den kleineren bzw. größeren von zwei übergebenen Werten.
<code>PI</code>	Diese Eigenschaft gibt den annähernd genauen Wert von <code>PI</code> zurück.
<code>Pow</code>	liefert die Potenz einer Zahl. Am ersten Argument übergeben Sie die Basis, am zweiten den Exponenten. <code>Pow(2,3)</code> bedeutet so viel wie 2^3 .

Tabelle 26: Die wichtigsten Methoden und Eigenschaften der `Math`-Klasse

Methode	Beschreibung
<pre>decimal Round(decimal d [, int decimals]) double Round(double value [, int decimals])</pre>	<code>Round</code> rundet eine Zahl auf eine optional anzugebende Anzahl Dezimalstellen, allerdings leider nicht so, wie wir es gewohnt sind. Am ersten Argument geben Sie die zu rundende Zahl an. Das zweite Argument bestimmt, auf wie viele Stellen hinter dem Komma gerundet wird. Leider rundet <code>Round</code> mathematisch (und nicht kaufmännisch). Beim mathematischen Runden wird, wenn der Ziffer, auf die gerundet wird, genau eine 5 folgt, aufgerundet, wenn diese Ziffer ungerade ist, und abgerundet, wenn diese Ziffer gerade ist. So rundet <code>Round(1.35, 1)</code> auf 1,4 und <code>Round(1.45, 1)</code> ebenfalls auf 1,4. <code>Round(1.45001, 1)</code> rundet allerdings auf 1.5. So ist sichergestellt, dass beim Runden nicht ein Ungleichgewicht zu den aufgerundeten Zahlen hin entsteht (wie es beim kaufmännischen Runden der Fall ist).
<code>Sqrt</code>	liefert die Quadratwurzel der übergebenen Zahl.

Tabelle 26: Die wichtigsten Methoden und Eigenschaften der `Math`-Klasse (Forts.)

Basiseigenschaften und -methoden der Auflistungsklassen

ICollection-Schnittstelle

Eigenschaft / Methode	Beschreibung
<code>object this [int index]</code>	Über den Indizierer erhalten Sie Zugriff auf ein Element.
<code>int Add(object value)</code>	fügt ein Element an die Liste an. Gibt den Index zurück, an dem das Element angefügt wurde.
<code>void Clear()</code>	entfernt alle Elemente aus der Liste.
<code>bool Contains(object value)</code>	ermittelt, ob ein Element mit einem angegebenen Wert existiert.
<code>int IndexOf(object value)</code>	ermittelt den Index eines Elements mit einem bestimmten Wert.
<code>void Insert(int index, object value)</code>	fügt ein Element in die Liste ein.
<code>void Remove(object value)</code>	löscht das erste Element, das den angegebenen Wert besitzt.
<code>void RemoveAt(int index)</code>	löscht das Element, das am angegebenen Index gespeichert wird.

Tabelle 27: Die wichtigsten Eigenschaften und Methoden der `ICollection`-Schnittstelle

IDictionaryEnumerator-Schnittstelle

Eigenschaft	Beschreibung
Entry	gibt ein DictionaryEntry-Objekt zurück, das im Wesentlichen den Schlüssel und den Wert eines Elements speichert.
Key	gibt den Schlüssel des aktuellen Elements zurück.
Value	gibt den Wert des aktuellen Elements zurück.

Tabelle 28: Die Eigenschaften der IDictionaryEnumerator-Schnittstelle

3 Referenz der wichtigsten der im Buch verwendeten Klassen

MailMessage

Eigenschaften

Eigenschaft	Beschreibung
Attachments	referenziert eine normale Auflistung (die die Schnittstelle <code>IList</code> implementiert). Über diese Auflistung können Sie der Mail theoretisch beliebige Objekte hinzufügen, was aber dann zu einer Ausnahme führt. Sie sollten immer eine Instanz der Klasse <code>MailAttachment</code> verwenden, um der Mail Anhänge anzufügen. Stellt sich nur die Frage, warum der Entwickler der <code>MailMessage</code>-Klasse hier nicht typischer programmiert hat. Möglicherweise können Sie auch ein <code>Stream</code>-Objekt oder Ähnliches übergeben, aber das ist leider nicht dokumentiert. Die <code>MailAttachment</code>-Klasse besitzt nur zwei wichtige Eigenschaften: ► Encoding: spezifiziert die Codierung des Anhangs mit den Werten der <code>MailEncoding</code>-Aufzählung: <code>Base64</code> und <code>UUEncode</code>. ► Filename: spezifiziert den Dateinamen der anzuhängenden Datei. Im Konstruktor müssen Sie dieser Klasse den Dateinamen übergeben und können optional gleich auch die Codierung festlegen (die aber wohl eher unwichtig ist, <code>Base64</code> ist der aktuelle Standard).
Bcc	spezifiziert die Adressen der Empfänger einer Kopie der Mail. Kopien haben den Sinn, dass der Empfänger weiß, dass er nicht auf die Mail antworten muss. Der Sender erhält aber im Gegensatz zu einer Kopie (<code>Cc</code>) keine Fehlermeldungs-Mail zurück, wenn einer der Kopienempfänger nicht erreichbar ist. Mehrere Adressen werden durch ein Semikolon getrennt angegeben.
Body	verwaltet den eigentlichen Text der Nachricht.

Tabelle 29: Die Eigenschaften der MailMessage-Klasse

Eigenschaft	Beschreibung
BodyEncoding	referenziert ein <code>Encoding</code> -Objekt mit Informationen zur Zeichencodierung. Sie können die statische Methode <code>System.Text.Encoding.GetEncoding({string name} int codepage)</code> verwenden, um ein <code>Encoding</code> -Objekt mit einer speziellen Codierung zu erzeugen (wobei <i>name</i> einer der standardisierten Zeichensatznamen wie »UTF-8«, »windows-1252« oder »unicode« und <i>codepage</i> die entsprechende Codepage-Nummer ist). Alternativ können Sie auch die statischen Eigenschaften <code>ASCII</code> , <code>BigEndianUnicode</code> , <code>Default</code> , <code>Unicode</code> , <code>UTF7</code> und <code>UTF8</code> der <code>Encoding</code> -Klasse zur Erzeugung eines solchen Objekts einsetzen. Der Defaultwert ist scheinbar die Einstellung des Servers, was aber leider nicht dokumentiert ist. Unter Windows ist das der Zeichensatz »Windows-1252«.
BodyFormat	spezifiziert mit einem der Werte der <code>MailFormat</code> -Aufzählung das Format der Mail: <code>Html</code> oder <code>Text</code> . Die Voreinstellung ist <code>Text</code> .
Cc	spezifiziert die Adressen der Empfänger einer Kopie der Mail. Kopien haben den Sinn, dass der Empfänger weiß, dass er nicht auf die Mail antworten muss. Der Sender erhält aber im Gegensatz zu einer Blindkopie (<code>Bcc</code>) eine Fehlermeldungs-Mail zurück, wenn einer der Kopienempfänger nicht erreichbar ist. Mehrere Adressen werden durch ein Semikolon getrennt angegeben.
From	gibt die Rücksende-Adresse (die des Senders) an.
Headers	In dieser Auflistung vom Typ <code>IDictionary</code> können Sie eigene Mail-Header ablegen.
Priority	gibt die Priorität der Mail mit den Werten der <code>MailPriority</code> -Aufzählung an: <code>High</code> , <code>Low</code> oder <code>Normal</code> . Die Priorität soll dem Empfänger eine Information über die Wichtigkeit der Mail übermitteln.
Subject	spezifiziert den Betreff.
To	verwaltet die Empfängeradressen. Mehrere Adressen werden durch Semikola getrennt.
UrlContentBase	definiert den <code>Content-Base</code> -Header der Mail, der die Basis-URL für alle in der Mail relativ angegebenen URLs verwaltet.
UrlContent-Location	definiert den <code>Content-Location</code> -Header der Mail.

Tabelle 29: Die Eigenschaften der `MailMessage`-Klasse (Forts.)

Regex (Reguläre Ausdrücke)

Die `Regex`-Klasse aus dem Namensraum `System.Text.RegularExpressions` ermöglicht die Arbeit mit regulären Ausdrücken.

Konstruktor

Konstruktor	Bedeutung
<code>Regex()</code>	erzeugt ein <code>Regex</code> -Objekt ohne Muster und Optionen.
<code>Regex(string pattern)</code>	erzeugt ein <code>Regex</code> -Objekt mit Muster und ohne spezielle Optionen.
<code>Regex(string pattern, RegexOptions options)</code>	erzeugt ein <code>Regex</code>-Objekt mit Muster und Optionen. Das zweite Argument bestimmt mit einem oder mehreren Werten der <code>RegexOptions</code>-Aufzählung, welche Optionen bei der Auswertung des regulären Ausdrucks angewendet werden. Die wichtigsten Konstanten dieser Aufzählung sind: ▶ <code>IgnoreCase</code>: Bestimmt, dass bei der Suche die Groß-/Kleinschreibung nicht berücksichtigt wird, ▶ <code>Multiline</code>: Bestimmt, dass nicht nur in der ersten Zeile eines mehrzeiligen Strings gesucht wird, sondern in allen Zeilen (Voreinstellung), ▶ <code>RightToLeft</code>: Gibt an, dass nicht von links nach rechts, sondern von rechts nach links gesucht wird, ▶ <code>Singleline</code>: Bestimmt, dass nur in der ersten Zeile eines mehrzeiligen Strings gesucht wird.

Tabelle 30: Die Konstruktor der `Regex`-Klasse

Klassenmethoden

Methode	Beschreibung
<code>string Escape(string str)</code>	Diese Methode stellt allen Zeichen mit einer Sonderbedeutung für reguläre Ausdrücke (<code>\</code> , <code>{</code> , <code>[</code> etc.) einen Backslash voran, sodass deren Bedeutung aufgehoben wird.
<code>string Unescape(string str)</code>	Diese Methode hebt alle Escape-Sequenzen im String wieder auf. Aus <code>\\</code> wird z.B. <code>\</code> , aus <code>\{</code> wird <code>{</code> etc.

Tabelle 31: Die wichtigen statischen Methoden der `Regex`-Klasse

Instanzeigenschaften

Eigenschaft	Beschreibung
<code>Options</code>	Aus dieser Eigenschaft können Sie die Optionen des regulären Ausdrucks auslesen (aber nicht setzen). <code>Options</code> gibt ein Bitfeld mit Werten der <code>RegexOptions</code> -Aufzählung zurück (siehe bei den Konstruktoren in Tabelle 30).
<code>RightToLeft</code>	liefert eine Information darüber, ob der String von rechts nach links durchsucht wird (schreibgeschützt).

Tabelle 32: Die Instanzeigenschaften der `Regex`-Klasse

Instanzmethoden

Methode	Beschreibung
<code>string[] GetGroupNames()</code>	liefert die Namen aller Gruppen (die im regulären Ausdruck über runde Klammern spezifiziert wurden).
<code>int[] GetGroupNumbers()</code>	liefert die Nummern aller Gruppen (die im regulären Ausdruck über runde Klammern spezifiziert wurden).
<code>string GroupNameFromNumber(int i)</code>	ergibt den Namen der Gruppe mit der angegebenen Nummer.
<code>int GroupNumberFromName(string name)</code>	ergibt die Nummer der Gruppe mit dem angegebenen Namen.
<code>bool IsMatch(string input, [int startat])</code> <code>bool IsMatch(string input, string pattern)</code> <code>bool IsMatch(string input, string pattern, RegexOptions options)</code>	Diese Methode überprüft, ob ein Muster in einem String vorkommt. Sie können am zweiten Argument angeben, ab welchem Index im String gesucht werden soll. Alternativ können Sie am zweiten Argument auch ein anderes Muster übergeben als das, mit dem das <code>Regex</code> -Objekt beim Erzeugen initialisiert wurde. Am dritten Argument können Sie einen oder mehrere Konstanten der Aufzählung <code>RegexOptions</code> übergeben. Mit der Konstante <code>IgnoreCase</code> erreichen Sie, dass die Groß-/ Kleinschreibung ignoriert wird. <code>Multiline</code> bewirkt, dass auch in allen der ersten Zeile folgenden Zeilen gesucht wird.
<code>Match Match(string input, [int startat])</code> <code>Match Match(string input, string pattern)</code> <code>Match Match(string input, int beginning, int length)</code> <code>Match Match(string input, string pattern, RegexOptions options)</code>	Diese Methode durchsucht einen String auf das im <code>Regex</code> -Objekt definierte oder übergebene Muster und gibt ein <code>Match</code> -Objekt für die erste Fundstelle zurück. Über die Eigenschaft <code>Success</code> dieses Objekts können Sie überprüfen, ob das Muster im String gefunden wurde. Die Eigenschaft <code>Index</code> gibt eine Information über die Position der Fundstelle, <code>Value</code> enthält den Wert. In der letzten Variante der Methode können Sie am dritten Argument einen oder mehrere Konstanten der Aufzählung <code>RegexOptions</code> übergeben. Die Bedeutung ist dieselbe wie <code>IsMatch</code> .

Tabelle 33: Die Instanzmethoden der `Regex`-Klasse

Methode	Beschreibung
<pre> MatchCollection Matches(string input [, int startat]) MatchCollection Matches(string input, string pattern) MatchCollection Match(string input, int beginning, int length) MatchCollection Matches(string input, string pattern, RegexOptions options) </pre>	<code>Matches</code> durchsucht einen String auf das im <code>Regex</code>-Objekt definierte oder übergebene Muster und gibt ein <code>MatchCollection</code>-Objekt zurück, das <code>Match</code>-Objekte für alle Fundstellen referenziert. Die Eigenschaft <code>Index</code> der <code>Match</code>-Objekte gibt eine Information über die Position der Fundstelle, <code>Value</code> enthält den Wert. In der letzten Variante der Methode können Sie am dritten Argument einen oder mehrere Konstanten der Aufzählung <code>RegexOptions</code> übergeben. Die Bedeutung ist dieselbe wie bei <code>IsMatch</code>.
<pre> string Replace(...) </pre>	Über diese mehrfach überladene Methode können Sie in einem String sehr flexibel alle oder nur eine bestimmte Anzahl an Fundstellen ersetzen. Neben dem String, einem optionalen Muster (das aber auch im <code>Regex</code>-Objekt definiert sein kann), einer Startposition und einer Ersetzungsanzahl können Sie einen Wert der Aufzählung <code>RegexOptions</code> übergeben. Mit der Konstante <code>IgnoreCase</code> erreichen Sie, dass die Groß-/Kleinschreibung ignoriert wird, <code>Multiline</code> bewirkt, dass auch in allen der ersten Zeile folgenden Zeilen gesucht wird. Interessant ist auch, dass Sie einigen Varianten einen Delegate vom Typ <code>MatchEvaluator</code> übergeben können. Die Signatur dieses Delegates ist die folgende: <pre>string MatchEvaluator(Match match)</pre> Das in einer entsprechenden Methode übergebene <code>Match</code>-Objekt enthält Informationen über die Fundstelle. Über die Eigenschaft <code>Value</code> können Sie den Wert der Fundstelle auslesen. Den veränderten Wert geben Sie dann zurück, damit <code>Replace</code> die Fundstelle durch den neuen Wert ersetzt. So können Sie sehr komplexe Ersetzungen programmieren.

Tabelle 33: Die Instanzmethoden der `Regex`-Klasse (Forts.)

Methode	Beschreibung
<pre>string[] Split(string input, int count) string[] Split(string input, string pattern) string[] Split(string input, int count, int startat) string[] Split(string input, string pattern, RegexOptions options)</pre>	Diese Methode trennt einen String an allen Stellen auf, die dem Muster des regulären Ausdrucks entsprechen. Am Argument <code>count</code> können Sie die Maximalanzahl der Trennungen angeben, am Argument <code>startat</code> den Index, ab dem gesucht werden soll. In der letzten Variante der Methode können Sie am dritten Argument einen oder mehrere Konstanten der Aufzählung <code>RegexOptions</code> übergeben. Die Bedeutung ist dieselbe wie bei <code>IsMatch</code>.

Tabelle 33: Die Instanzmethoden der `Regex`-Klasse (Forts.)

Zeichenklassen

Zeichenklassen definieren einzelne Zeichen im Muster.

Zeichenklasse	Beschreibung
.	Der Punkt steht für alle Zeichen außer <code>\n</code> .
[<i>Zeichenliste</i>]	steht für eines der Zeichen, die in der Zeichenliste eingetragen sind. Die Liste darf Bereiche enthalten, beispielsweise <code>a-z</code> für die Zeichen <code>a</code> bis <code>z</code> , und einzelne Zeichen. Die Zeichenklasse <code>[a-zA-ZäöüßÄÖÜ]</code> lässt z.B. alle normalen deutschen Wortzeichen zu.
[<i>^Zeichenliste</i>]	steht für alle Zeichen, die nicht in der Zeichenliste angegeben sind.
\b	steht für eine Wort-Grenze. Eine Wort-Grenze ist eine Position an einem Wort-Trennzeichen (Leerzeichen, Bindestrich, Zeilenumbruch etc.), das direkt an ein Wort angrenzt, oder der Beginn oder das Ende einer Zeile. Wenn Sie beispielsweise nach Wörtern suchen, die aus drei Ziffern bestehen, verwenden Sie das folgende Muster: <code>\b\d{3}\b</code> .
\B	steht für eine Nicht-Wort-Grenze. Wenn Sie beispielsweise nach einem Zeichen <code>x</code> suchen, das nicht am Anfang eines Wortes steht, verwenden Sie das folgende Muster: <code>\Bx</code> .
\d	steht für eine Dezimalziffer (0-9).
\D	steht für ein Zeichen, das keine Dezimalziffer ist.
\s	steht für ein Whitespace-Zeichen (Leerzeichen, Tabulator, <code>\r</code> , <code>\n</code> etc.).
\S	steht für ein Zeichen, das kein Whitespace-Zeichen ist.

Tabelle 34: Die wichtigsten Zeichenklassen der regulären Ausdrücke

Zeichenklasse	Beschreibung
<code>\w</code>	spezifiziert ein Wort-Zeichen (u.a. a-z, A-Z, 1-0 und der Unterstrich). <code>C#</code> berücksichtigt dabei alle Wort-Zeichen, die im Unicode-Standard als solche gekennzeichnet sind, auch die von Kulturen mit einer anderen Schriftsprache als der unseren.
<code>\W</code>	spezifiziert ein Zeichen, das kein Wort-Zeichen ist.
<code>\xhh</code>	spezifiziert ein Zeichen über einen zweistelligen hexadezimalen ASCII-Zeichencode.
<code>\uhhhh</code>	spezifiziert ein Zeichen über einen zweistelligen hexadezimalen Unicode-Zeichencode.

Tabelle 34: Die wichtigsten Zeichenklassen der regulären Ausdrücke (Forts.)

Quantifizierer

Quantifizierer werden rechts neben einem Zeichen, einer Zeichenklasse oder einer Zeichengruppe im Muster eingetragen und geben die erforderliche und/oder mögliche Anzahl des links spezifizierten Zeichens an.

Quantifizierer	Beschreibung
<code>*</code>	gibt an, dass das auf der linken Seite spezifizierte Zeichen keinmal, einmal oder beliebig oft in Folge vorkommen kann. <code>/\d*/</code> steht beispielsweise für beliebig viele Dezimalziffern. Verwechseln Sie diesen Quantifizierer nicht mit dem Wildcard, der u.a. in Windows verwendet wird, um beliebig viele Zeichen zu spezifizieren.
<code>+</code>	gibt an, dass das auf der linken Seite spezifizierte Zeichen einmal vorkommen muss und beliebig oft in Folge vorkommen kann. Ist identisch mit <code>{1,}</code> .
<code>?</code>	gibt an, dass das auf der linken Seite spezifizierte Zeichen keinmal oder einmal vorkommen darf. Ist identisch mit <code>{0,1}</code> .
<code>{n}</code>	gibt an, dass das auf der linken Seite spezifizierte Zeichen genau <i>n</i> -mal hintereinander vorkommen muss. <code>\d{5}</code> steht z.B. für »genau 5 Dezimalziffern«.
<code>{min,}</code>	spezifiziert mindestens <i>min</i> Zeichen.
<code>{min, max}</code>	spezifiziert mindestens <i>min</i> und maximal <i>max</i> Zeichen.

Tabelle 35: Die Quantifizierer der regulären Ausdrücke

Gruppierungen

Reguläre Ausdrücke erlauben das Gruppieren von einzelnen Zeichen, sodass Sie beispielsweise Quantifizierer auf eine Gruppe anwenden können. Gleichzeitig können Sie mit Gruppierungen auch Bedingungen definieren und für spezielle Suchtechniken festlegen, welche Fundstellen in das Ergebnis übertragen werden.

Gruppierung	Beschreibung
(<i>Teilmuster</i>)	Diese Gruppierung besitzt zwei Bedeutungen: Zum einen wird das Teilmuster gruppiert, sodass Sie z. B. Quantifizierer auf das Teilmuster anwenden oder das logische Oder () verfeinern können. <code>(\w\d){1,}</code> steht beispielsweise für eine Kombination aus einer Dezimalziffer und einem Wortzeichen, die einmal oder beliebig oft vorkommen darf. <code>x(\d{3} ###)</code> steht für »x gefolgt von einer dreistelligen Ziffer oder gefolgt von ###«. Zum anderen wird die Fundstelle in ein internes Array geschrieben, sodass Sie sich in Rückreferenzen darauf beziehen können. Diese Teil-Fundstellen können Sie auch über die <code>Groups</code> -Eigenschaft eines <code>Match</code> -Objekts (das Sie z. B. über den Aufruf der <code>Matches</code> -Methode erhalten haben) auslesen.
(?: <i>Teilmuster</i>)	gruppiert ein Teilmuster. Die Fundstelle des Teilmusters wird aber nicht in das interne Array übertragen.
<code>x(?:y)</code>	Dieses Muster bewirkt, dass das Teilmuster <code>x</code> nur dann zum Erfolg führt, wenn das Teilmuster <code>y</code> direkt folgt. So können Sie Teilstrings finden, denen ein anderer String direkt angehängt ist. Das Muster <code>/\d{3}(?=x)/</code> findet z. B. alle dreistelligen Zahlen, denen ein <code>x</code> folgt. Im String »123a 456x 789x« würden beispielsweise die Teilstrings »456« und »789« gefunden werden, nicht aber »123«.
<code>x(!y)</code>	bewirkt, dass nur die Fundstellen in das Ergebnis übernommen werden, die dem Muster <code>x</code> entsprechen und denen nicht ein String folgt, der dem Muster <code>y</code> entspricht.

Tabelle 36: Die Gruppierungen der regulären Ausdrücke

Smtplib

Klasseneigenschaften

Eigenschaft	Beschreibung
<code>Smtplib</code>	gibt den zu verwendenden SMTP-Server an. Ein Leerstring steht für den lokalen Server.

Tabelle 37: Die statische Eigenschaft der Smtplib-Klasse

Klassenmethoden

Methode	Beschreibung
<code>void Send(MailMessage message)</code>	sendet eine E-Mail unter Übergabe eines <code>MailMessage</code> -Objekts. Mit dieser Variante der <code>Send</code> -Methode können Sie auch spezielle Eigenschaften wie <code>Cc</code> und <code>Bcc</code> definieren und Dateien an die Mail anhängen.

Tabelle 38: Die statischen Methoden der Smtplib-Klasse

Methode	Beschreibung
<code>void Send(string from, string to, string subject, string messageText)</code>	sendet eine einfache Mail ohne Cc und Bcc und ohne Dateianhänge.

Tabelle 38: Die statischen Methoden der *SmtpMail*-Klasse (Forts.)

XmlDocument

Die Klasse `XmlDocument` repräsentiert ein komplettes XML-Dokument. `XmlDocument` ist von `XmlNode` abgeleitet und erbt deswegen alle Eigenschaften und Methoden dieser Klasse. Darum beschreibe ich auch nur die wenigen Methoden, die in dieser Klasse zusätzlich definiert sind.

Instanzmethoden

Methode	Beschreibung
<code>XmlAttribute CreateAttribute(string name)</code> <code>XmlAttribute CreateAttribute(string qualifiedName, string namespaceURI)</code> <code>XmlAttribute CreateAttribute(string prefix, string localName, string namespaceURI)</code>	erzeugt ein Attribut. Sie können einen Namen angeben, der im ersten Teil auch einen Namensraum-Präfix enthalten kann. Namensraum-Präfix und Name werden dann durch einen Doppelpunkt getrennt. Neben diesem Namen können Sie am zweiten Argument optional auch einen Namensraum angeben, dem das Attribut zugeordnet werden soll. Die dritte Variante der <code>CreateAttribute</code> -Methode erwartet einen Namensraum-Präfix, den lokalen Namen und den Namensraum des Attributs.
<code>XmlComment CreateComment(string data)</code>	erzeugt einen XML-Kommentar.
<code>XmlElement CreateElement(string name)</code> <code>XmlElement CreateElement(string qualifiedName, string namespaceURI)</code> <code>XmlElement CreateElement(string prefix, string localName, string namespaceURI)</code>	erzeugt ein XML-Element. Die Argumente entsprechend denen von <code>CreateAttribute</code> .

Tabelle 39: Wichtige Instanzmethoden der *XmlDocument*-Klasse (außer die von *XmlNode* geerbten)

Methode	Beschreibung
<code>XmlText CreateTextNode(string text)</code>	erstellt ein Text-Element.
<code>XmlDeclaration CreateXmlDeclaration(string version, string encoding, string standalone)</code>	erzeugt eine XML-Deklaration mit den angegebenen Attributwerten.
<code>XmlElement GetElementById(string elementId)</code>	sucht ein XML-Element mit der angegebenen Id und gibt eine Referenz darauf zurück. Attribute, die als Id gewertet werden, müssen in einer DTD, die dem XML-Dokument zugeordnet ist, als solche gekennzeichnet sein. Das Attribut <i>id</i> , das häufig in HTML-Dokumenten verwendet wird, ist übrigens nicht automatisch ein Id-Attribut.
<code>XmlNodeList GetElementsByTagName(string name)</code> <code>XmlNodeList GetElementsByTagName(string localName, string namespaceURI)</code>	Über diese Methode können Sie alle XML-Elemente eines Dokuments, deren Tags einen bestimmten Namen tragen, in Form einer <code>XmlNodeList</code> -Auflistung ermitteln. Dazu übergeben Sie entweder nur den lokalen Namen des Tags oder alternativ auch den Namensraum um nur die Elemente auszuwählen, die einem bestimmten Namensraum zugeordnet sind.
<code>XmlNode ImportNode(XmlNode node, bool deep)</code>	importiert einen Knoten (der normalerweise aus einem anderen Dokument stammt) in das Dokument. Wenn Sie am Argument <i>deep</i> <code>true</code> angeben, werden auch alle untergeordneten Knoten kopiert.
<code>void Load(Stream inputStream)</code> <code>void Load(string filename)</code> <code>void Load(TextReader txtReader)</code> <code>void Load(XmlReader reader)</code>	lädt ein XML-Dokument. Sie können ein geöffnetes <code>Stream</code> -, <code>TextReader</code> - oder <code>XmlReader</code> -Objekt übergeben, aus dem die Daten gelesen werden sollen. Alternativ können Sie auch einfach nur den Dateinamen der XML-Datei übergeben.
<code>void LoadXml(string xml)</code>	lädt ein XML-Dokument, das als String übergeben wird. Diese Methode eignet sich auch, um den Rumpf eines XML-Dokuments effizienter zu erzeugen als über die »normalen« Methoden von der <code>XmlDocument</code> : <pre>xmlDoc.LoadXml("<?xml version=\"1.0\" " + "encoding=\"utf-8\" " + "standalone=\"yes\"?>" + "<persons></persons>");</pre>

Tabella 39: Wichtige Instanzmethoden der `XmlDocument`-Klasse (außer die von `XmlNode` geerbten) (Forts.)

Methode	Beschreibung
<pre>void Save(Stream outputStream) void Save(string filename) void Save(TextWriter writer) void Save(XmlWriter w)</pre>	speichert das XML-Dokument in einen Stream oder eine Datei. Sie können einen Stream, ein <code>TextWriter</code> -Objekt, ein <code>XmlWriter</code> -Objekt oder einfach nur einen Dateinamen übergeben.

Tabelle 39: Wichtige Instanzmethoden der `XmlDocument`-Klasse (außer die von `XmlNode` geerbten) (Forts.)

XmlNode

`XmlNode` ist die Basisklasse für alle XML-Elemente eines XML-Dokuments und repräsentiert einen allgemeinen XML-Knoten in einem solchen.

Instanzeigenschaften

Eigenschaft	Beschreibung
Attributes	referenziert eine Auflistung vom Typ <code>XmlAttributeCollection</code> , die die einzelnen Attribute des Knotens verwaltet. Über den Indizierer erhalten Sie <code>XmlAttribute</code> -Objekte, indem Sie den Namen oder den <code>int</code> -Index des Attributs übergeben. Da die <code>XmlAttribute</code> -Klasse auch von <code>XmlNode</code> abgeleitet ist, besitzt diese dieselben Eigenschaften und Methoden.
ChildNodes	ergibt eine <code>XmlNodeList</code> -Auflistung, die alle untergeordneten Knoten in Form von <code>XmlNode</code> -Objekten enthält. Zum Zugriff auf diese Knoten übergeben Sie dem Indizierer der <code>ChildNodes</code> -Auflistung den Namen oder den <code>int</code> -Index des Knotens.
FirstChild LastChild	ergibt eine Referenz auf den ersten bzw. letzten untergeordneten Knoten in Form einer <code>XmlNode</code> -Instanz. Ist kein untergeordneter Knoten vorhanden, geben diese Eigenschaften <code>null</code> zurück.
InnerText	verwaltet den textuellen Inhalt eines XML-Elements.
InnerXml	verwaltet den XML-Inhalt eines XML-Elements (inklusive aller untergeordneten Elemente).
IsReadOnly	gibt an, ob der aktuelle Knoten schreibgeschützt ist.
this[string localName , string ns]	Über den Indizierer können Sie ein dem Knoten untergeordnetes XML-Element in Form eines <code>XmlElement</code> -Objekts referenzieren. Neben dem lokalen Namen des Elements können Sie im optionalen zweiten Argument auch den Namensraum übergeben.
LocalName	ergibt den lokalen Namen des Knotens (der Name ohne Namensraum).
NamespaceURI	ergibt den URI des Namensraums des Knotens als String.

Tabelle 40: Die wichtigsten Instanzeigenschaften der `XmlNode`-Klasse

Eigenschaft	Beschreibung
NextSibling PreviousSibling	ergibt eine Referenz auf den Knoten, der dem Knoten unmittelbar folgt (das nächste »Geschwister«) bzw. vorsteht (das vorherige »Geschwister«).
NodeType	ergibt in Klassen, die von <code>XmlNode</code> abgeleitet sind, den Typ des Knotens in Form eines Wertes der <code>XmlNodeType</code> -Aufzählung. Die wichtigsten Werte sind: ▶ Attribute: Attribut ▶ CDATA: CDATA-Abschnitt ▶ Comment: Kommentar ▶ Document: das Dokument-Element ▶ Element: XML-Start-Tag ▶ EndElement: XML-End-Tag ▶ ProcessingInstruction: Verarbeitungsanweisung (<code><? ... ?></code>) ▶ Text: Textinhalt eines Elements ▶ XmlDeclaration: XML-Deklaration
ParentNode	ergibt eine Referenz auf den übergeordneten Knoten oder <code>null</code> , falls kein solcher vorhanden ist.
Value	ergibt den Wert eines Knotens.

Tabelle 40: Die wichtigsten Instanzeigenschaften der `XmlNode`-Klasse (Forts.)

Instanzmethoden

Methode	Beschreibung
<code>XmlNode AppendChild(XmlNode newChild)</code>	hängt einem Knoten einen neuen Unterknoten an.
<code>XPathNavigator CreateNavigator()</code>	erzeugt ein <code>XPathNavigator</code> -Objekt zum Navigieren im Knoten.
<code>XmlNode InsertAfter(XmlNode newChild, XmlNode refChild)</code>	fügt dem Knoten einen Unterknoten hinter einem anzugebenden anderen Unterknoten an.
<code>XmlNode InsertBefore(XmlNode newChild, XmlNode refChild)</code>	fügt dem Knoten einen Unterknoten vor einem anzugebenden anderen Unterknoten an.
<code>void RemoveAll()</code>	löscht alle Unterknoten.
<code>XmlNode RemoveChild(XmlNode oldChild)</code>	löscht einen Unterknoten.
<code>XmlNode ReplaceChild(XmlNode newChild, XmlNode oldChild)</code>	ersetzt einen Unterknoten durch einen anderen.

Tabelle 41: Die wichtigsten Methoden der `XmlNode`-Klasse

Methode	Beschreibung
<code>XmlNodeList SelectNodes(string xpath [, XmlNamespaceManager nsmgr])</code>	ergibt eine Liste aller einem Knoten untergeordneten Knoten, die dem übergebenen XPath-Ausdruck entsprechen. Das zweite Argument müssen Sie verwenden, wenn der Knoten oder dessen Unterknoten einem oder mehreren Namensräumen zugeordnet ist. Über eine <code>XmlNamespaceManager</code> -Instanz legen Sie die Namensräume fest, in denen gesucht werden soll (siehe Rezept 137).
<code>XmlNode SelectSingleNode(string xpath [, XmlNamespaceManager nsmgr])</code>	<code>SelectSingleNode</code> arbeitet ähnlich wie <code>SelectNodes</code> , nur dass diese Methode lediglich eine Referenz auf den ersten gefundenen Knoten zurückgibt.

Tabelle 41: Die wichtigsten Methoden der `XmlNode`-Klasse (Forts.)

4 Referenz der im Buch verwendeten Windows-API-Funktionen

BitBlt

Die Funktion `BitBlt` (Bit Blast) kopiert die Bildinformationen eines Quell-DC (Device Context) in einen Ziel-DC. Damit können Sie z.B. ein Bitmap-Bild auf ein anderes kopieren und dabei die Bildinformationen des Ziel-Bitmaps gezielt vermischen oder überschreiben. Beim Kopieren geben Sie neben den DCs auch die jeweilige X/Y-Position und die Ziel-Breite und -Höhe des zu kopierenden Ausschnitts an und können so auch nur Teile eines Bildes auf Teile eines anderen Bilds kopieren. Über das letzte Argument steuern Sie die Art der Vermischung der Bildpunkte.

Deklaration

```
[DllImport("gdi32.dll", SetLastError=true)]
private static extern int BitBlt(IntPtr hdcDest, int nXDest, int nYDest,
    int nWidth, int nHeight, IntPtr hdcSrc, int nXSrc, int nYSrc,
    int dwRop);
```

Argumente

- ▶ `hDest`: DC des Ziels,
- ▶ `nXDest`, `nYDest`: X/Y-Position der linken, oberen Ecke auf dem Ziel-DC in logischen Einheiten,
- ▶ `nWidth`, `nHeight`: Breite und Höhe des Quell- und des Zielausschnitts,
- ▶ `hdcSrc`: DC der Quelle,
- ▶ `nXSrc`, `nYSrc`: X/Y-Position der linken, oberen Ecke auf dem Quell-DC,

- **dwRop:** Dieses Argument bestimmt den *Raster Operation Code*, der definiert, auf welche Weise die Farbinformationen des Quellausschnitts mit den Farbinformationen des Zielausschnitts vermischt werden.

Für den Raster Operation Code können Sie einige Konstanten kombinieren. Tabelle 42 beschreibt die wichtigsten.

Konstante	Wert	Bedeutung
SRCCOPY	0x00CC0020	Die Quelle wird 1:1 in das Ziel kopiert.
SRCPAINT	0x00EE0086	Beim Kopieren werden die Farbinformationen mit einem logischen OR verknüpft.
SRCAND	0x008800C6	Beim Kopieren werden die Farbinformationen mit einem logischen AND verknüpft.
SRCINVERT	0x00660046	Beim Kopieren werden die Farbinformationen mit einem logischen XOR verknüpft.
SRCERASE	0x00440328	Beim Kopieren werden die Farbinformationen der Quelle mit den invertierten Informationen des Ziels über ein logisches AND verknüpft.
NOTSRCCOPY	0x00330008	Die Quelle wird invertiert in das Ziel kopiert.
DSTINVERT	0x00550009	invertiert das Ziel.
BLACKNESS	0x00000042	füllt das Ziel mit der Farbe, die in der aktuellen Palette den Index 0 besitzt (per Default ist das Schwarz).
WHITENESS	0x00FF0062	füllt das Ziel mit der Farbe, die in der aktuellen Palette den Index 1 besitzt (per Default ist das Weiß).
NOMIRRORBITMAP	0x80000000	Dieses Flag bewirkt, dass das Bitmap nicht gespiegelt wird.

Tabelle 42: Wichtige Konstanten für den Raster Operation Code

Rückgabe

BitBlt gibt einen Wert ungleich Null zurück, wenn die Funktion erfolgreich ausgeführt wurde. Im anderen Fall wird Null zurückgegeben. Den Fehlercode können Sie über `Marshal.GetLastWin32Error` auslesen.

CreateDC

CreateDC erzeugt einen Device Context (DC) für ein Gerät (Fenster, Bild, Drucker etc.), auf dem Grafikausgaben möglich sind, und gibt den Handle dieses DC zurück.

Deklaration

```
[DllImport("gdi32.dll", SetLastError=true)]
private static extern IntPtr CreateDC(string lpszDriver, string lpszDevice,
    string lpszOutput, IntPtr lpInitData);
```

Argumente

- ▶ `lpSzDriver`: Der Name des zu verwendenden Treibers. Für den Bildschirm geben Sie »DISPLAY« an. Dieses Argument kann `null` sein, wenn kein spezieller Treiber verwendet werden soll,
- ▶ `lpSzDevice`: Der Name des Geräts, z.B. der Namen eines Druckers
- ▶ `lpSzOutput`: Wird nicht verwendet. Sie sollten hier `null` übergeben.
- ▶ `lpInitData`: Zeiger auf eine `DEVMODE`-Struktur mit gerätespezifischen Angaben zum Treiber. Unter anderem können Sie beim Drucken über API-Funktionen in dieser Struktur die Papiergröße und -ausrichtung bestimmen. Die `DEVMODE`-Struktur ist recht komplex und wird hier nicht erläutert, auch deswegen, weil der Einsatz von `CreateDC` unter .NET diese Struktur kaum erfordert. Sie können `IntPtr.Zero` übergeben, wenn Sie keine speziellen Daten übergeben wollen.

Rückgabe

`CreateDC` gibt den Handle des erzeugten DC zurück, wenn der Aufruf erfolgreich war. Im anderen Fall gibt `CreateDC` `IntPtr.Zero` zurück.

DeleteMenu

`DeleteMenu` löscht einen Eintrag aus einem Menü (speziell dem Systemmenü eines Fensters).

Deklaration

```
[DllImport("user32.dll")]
private static extern int DeleteMenu(IntPtr hMenu, int uPosition, int uFlags);
```

Argumente

- ▶ `hMenu`: Der Handle des Menüs, der z.B. über `GetSystemMenu` ermittelt wurde
- ▶ `uPosition`: Gibt die Position des Menüs an. Für das Systemmenü sind einige Konstanten definiert (siehe Tabelle 43).
- ▶ `uFlags`: Definiert, wie `uPosition` interpretiert wird. Über `MF_BYCOMMAND` (0) wird `uPosition` als Id des Menüs interpretiert, über `MF_BYPOSITION` (0x00000400) als Index.

Konstante	Wert	Menübefehl
<code>SC_CLOSE</code>	<code>0xF060</code>	Schließen
<code>SC_CONTEXTHELP</code>	<code>0xF180</code>	Kontextsensitive Hilfe
<code>SC_MAXIMIZE</code>	<code>0xF030</code>	Maximieren
<code>SC_MINIMIZE</code>	<code>0xF020</code>	Minimieren
<code>SC_MOVE</code>	<code>0xF010</code>	Verschieben
<code>SC_RESTORE</code>	<code>0xF120</code>	Wiederherstellen
<code>SC_SIZE</code>	<code>0xF000</code>	Größe ändern

Tabelle 43: Konstanten für die Systemmenüeinträge

Rückgabe

`DeleteMenu` gibt einen Wert ungleich Null zurück, wenn der Aufruf erfolgreich war. Im Fehlerfall gibt die Funktion Null zurück. Den Fehlercode können Sie über `Marshal.GetLastWin32Error` auslesen.

FindWindow

`FindWindow` ermittelt den Fenster-Handle eines Fensters. Über diese Funktion können Sie ein bestimmtes Fenster suchen und über den Handle weiterverarbeiten, indem Sie diesen an andere Fenster-API-Funktionen übergeben.

Deklaration

```
[DllImport("User32.dll", SetLastError=true)]
private static extern int FindWindow(string lpClassName, string lpWindowName);
```

Argumente

Am Argument `lpClassName` können Sie den Namen der Fensterklasse übergeben um nur Fenster zu suchen, die einer bestimmten Klasse angehören. Der Fensterklassenname ist der Windows-interne Name der Klasse des Fensters. Wenn Sie für den Klassennamen `null` übergeben, sucht `FindWindow` ein beliebiges Fenster mit dem übergebenen Titel. Am zweiten Argument übergeben Sie den Titel des Fensters.

Rückgabe

`FindWindow` gibt den Handle des Fensters zurück, wenn eines gefunden wurde. Wird kein Fenster gefunden, das den übergebenen Kriterien entspricht, gibt diese Funktion 0 zurück.

FormatMessage

`FormatMessage` kann eine Zeichenkette formatieren (was für uns nicht interessant ist) und eine im System oder in einer DLL-Datei vordefinierte Meldung auslesen. Der für uns interessanteste Einsatzzweck ist das Auslesen von Fehlerbeschreibungen über einen Fehlercode.

Deklaration

```
[DllImport("Kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
    int dwMessageId, int dwLanguageId, System.Text.StringBuilder lpBuffer,
    int nSize, string [] Arguments);
```

Argumente

Argument	Bedeutung
<code>dwFlags</code>	spezifiziert die Arbeitsweise der Funktion mit festgelegten Konstanten, von denen allerdings nur wenige für das Ermitteln einer Fehlerbeschreibung in einem C#-Programm notwendig sind: ► <code>FORMAT_MESSAGE_FROM_SYSTEM</code> (0x1000): Die Fehlermeldung wird aus der System-Meldungstabelle ausgelesen. ► <code>FORMAT_MESSAGE_FROM_HMODULE</code> (0x0800): Die Fehlermeldung wird aus einem Modul (einer DLL) ausgelesen, dessen Handle am Argument <code>lpSource</code> angegeben ist.
<code>lpSource</code>	Dieses Argument gibt (u.a.) den Handle eines Moduls (einer DLL) an, aus dem die Fehlermeldung ausgelesen werden soll, wenn diese nicht in den System-DLLs gespeichert ist. Diesen Handle können Sie über die API-Funktion <code>LoadLibrary</code> auslesen. Wenn Sie (was in den meisten Fällen wohl ausreichen wird) die System-Meldungstabelle auslesen, übergeben Sie an diesem Argument <code>(IntPtr)0</code>. Das Argument <code>lpSource</code> ist im Original als Zeiger auf <code>void</code> deklariert. Deshalb können Sie an diesem Argument auch einen Quellstring angeben, wenn Sie <code>FormatMessage</code> zur Stringformatierung verwenden. Dann müsste das Argument in C# aber als <code>string</code> deklariert werden.
<code>dwMessageId</code>	Die Id der Meldung. Bei Fehlermeldungen ist dies der API-Fehlercode.
<code>dwLanguageId</code>	Hier können Sie eine Sprach-Id angeben, um die Meldung in einer bestimmten Sprache zu ermitteln (sofern diese verfügbar ist). Geben Sie 0 an um die System-sprache zu verwenden.
<code>lpBuffer</code>	Dieses Argument muss mit einem ausreichend groß initialisierten <code>StringBuilder</code> -Objekt belegt werden, in das die Funktion die ermittelte Meldung einträgt.
<code>nSize</code>	spezifiziert die Initialgröße des <code>StringBuilder</code> -Objekts, sodass die Funktion weiß, wie viele Zeichen geschrieben werden können.
<code>Arguments</code>	<code>Arguments</code> ist für unsere Zwecke nicht interessant: <code>FormatMessage</code> kann ähnlich <code>String.Format</code> auch Zeichenketten mit festgelegten Platzhaltern formatieren, in die die Inhalte des Arrays eingetragen werden. Übergeben Sie hier einfach <code>null</code> .

Tabelle 44: Die Argumente der Funktion `FormatMessage`

Rückgabe

Im Erfolgsfall gibt die Funktion die Anzahl der gelesenen Zeichen zurück. Im Fehlerfall wird 0 zurückgegeben.

GetClassName

`GetClassName` ermittelt den Klassennamen eines Fensters.

Deklaration

```
[DllImport("User32.dll", SetLastError=true)]  
private static extern int GetClassName(IntPtr hWnd, StringBuilder lpClassName,  
    int nMaxCount);
```

Argumente

Am Argument `hWnd` übergeben Sie den Handle des Fensters, dessen Klassennamen Sie ermitteln wollen. An `lpClassName` übergeben Sie ein `StringBuilder`-Objekt, das ausreichend groß dimensioniert sein muss. Das Argument `nMaxCount` nimmt die Anzahl der Zeichen im `StringBuilder`-Objekt auf.

Rückgabe

`GetClassName` gibt die Anzahl der gelesenen Zeichen zurück, wenn die Ausführung erfolgreich war. In diesem Fall wurde der Klassenname in das `StringBuilder`-Objekt geschrieben. Im Fehlerfall gibt `GetClassName` 0 zurück.

GetDiskFreeSpaceEx

`GetDiskFreeSpaceEx` liefert Informationen zur Größe und zum verfügbaren Platz eines Laufwerks.

Deklaration

```
private static extern int GetDiskFreeSpaceEx(  
    string lpDirectoryName,  
    ref ulong lpFreeBytesAvailable,  
    ref ulong lpTotalNumberOfBytes,  
    ref ulong lpTotalNumberOfFreeBytes);
```

Argumente

Am Argument `lpDirectoryName` übergeben Sie das Kürzel des Laufwerks (*c:*, *d:* etc.). In der am Argument `lpFreeBytesAvailable` übergebenen Variablen liefert die Funktion dann die Anzahl der freien Bytes zurück, die dem aktuellen Benutzer zur Verfügung steht. Diese Anzahl kann kleiner sein als die Anzahl der gesamten freien Bytes, falls das System eine benutzerspezifische Verwaltung des freien Platzes unterstützt. Am Argument `lpTotalNumberOfBytes` erhalten Sie die Größe des Laufwerks, das Argument `lpTotalNumberOfFreeBytes` wird mit der Gesamtanzahl der freien Bytes (benutzerunabhängig) belegt.

Rückgabe

Im Erfolgsfall gibt `GetDiskFreeSpaceEx` einen Wert ungleich Null zurück. Falls Sie einen ungültigen Laufwerksbezeichner übergeben oder ein sonstiger Fehler auftritt, wird Null zurückgegeben.

GetDriveType

`GetDriveType` ermittelt den Typ eines Laufwerks. Übergeben wird der Stammordner des Laufwerks. Die Rückgabe ist ein `uint`-Wert, der den Typ definiert.

Deklaration

```
[DllImport("Kernel32.dll")]
private static extern uint GetDriveType(string lpRootPathName);
```

Konstanten für die Rückgabe

```
private const uint DRIVE_UNKNOWN = 0;      // Unbekannter Typ
private const uint DRIVE_NO_ROOT_DIR = 1;  // Pfad ist ungültig
private const uint DRIVE_REMOVABLE = 2;    // Entfernbares LW
private const uint DRIVE_FIXED = 3;        // Festplatten-LW
private const uint DRIVE_REMOTE = 4;       // Netzwerklaufwerk
private const uint DRIVE_CDROM = 5;       // CD-ROM-Laufwerk
private const uint DRIVE_RAMDISK = 6;      // RAM-Disk
```

GetShortPathName

GetShortPathName ermittelt den kurzen (8.3-)Dateinamen einer Datei.

Deklaration

```
[DllImport("kernel32.dll", SetLastError=true)]
private static extern int GetShortPathName(string lpzLongPath,
    System.Text.StringBuilder lpzShortPath, int cchBuffer);
```

Anwendung

Am Argument `lpzLongPath` übergeben Sie den langen Dateinamen. Das Argument `lpzShortPath` muss mit einem für Pfadangaben ausreichend groß dimensionierten `StringBuilder`-Objekt belegt werden. Am letzten Argument übergeben Sie die Größe dieses Objekts. `GetShortPathName` ermittelt an diesem Argument, wie viele Zeichen in den String geschrieben werden können.

Rückgabe

Im Erfolgsfall gibt `GetShortPathName` die Anzahl der gelesenen Zeichen zurück, im Fehlerfall wird 0 zurückgegeben. Bei Fehlern setzt diese Funktion den Windows-Fehlercode, den Sie dann über `Marshal.GetLastWin32Error` auslesen und über `FormatMessage` in eine Fehlerbeschreibung umwandeln können.

GetSystemMenu

Die Funktion `GetSystemMenu` ermöglicht den Zugriff auf das Systemmenü eines Formulars. Sie können das Menü kopieren und über den zurückerhaltenen Menü-Handle und spezielle API-Funktionen (wie z. B. `DeleteMenu`) verändern oder das originale Menü wiederherstellen.

Deklaration

```
[DllImport("user32.dll")]  
private static extern IntPtr GetSystemMenu(IntPtr hWnd, int bRevert);
```

Argumente

Am Argument `hWnd` übergeben Sie den Handle des Fensters. `bRevert` bestimmt mit 0 (FALSE), dass das Systemmenü kopiert wird und Sie diese Kopie dann über den zurückerhaltenen Handle verändern können. Wenn Sie an `bRevert` einen Wert ungleich 0 übergeben (TRUE), wird das originale Systemmenü wiederhergestellt.

Rückgabe

`GetSystemMenu` gibt den Handle des Systemmenüs zurück, wenn `bRevert true` ist. Ansonsten wird Null zurückgegeben.

GetWindowsDirectory

`GetWindowsDirectory` ermittelt den Pfad des Windows-Ordners.

Deklaration

```
[DllImport("kernel32.dll", SetLastError=true)]  
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,  
uint uSize);
```

Argumente

Am Argument `lpBuffer` übergeben Sie ein `StringBuilder`-Objekt, das ausreichend groß dimensioniert sein muss. Das Argument `uSize` nimmt die Anzahl der Zeichen im `StringBuilder`-Objekt auf.

Rückgabe

`GetWindowsDirectory` gibt die Anzahl der gelesenen Zeichen zurück, wenn der Pfad des Windows-Ordners ermittelt werden konnte. Im Fehlerfall gibt `GetWindowsDirectory` Null zurück.

InternetAutodial

Die Funktion `InternetAutodial` öffnet die Standard-DFÜ-Netzwerkverbindung, falls diese zurzeit geschlossen ist. Leider arbeitet diese Funktion nicht mit einer DSL-Verbindung über eine Netzwerkkarte.

Deklaration

```
[DllImport("wininet.dll", SetLastError=true)]  
public static extern int InternetAutodial(int dwFlags, IntPtr hwndParent);
```

Argumente

Am Argument `dwFlags` können Sie eine (und nur eine!) der folgenden Konstanten übergeben:

```
private const int INTERNET_AUTODIAL_FORCE_ONLINE = 1;
private const int INTERNET_AUTODIAL_FORCE_UNATTENDED = 2;
private const int INTERNET_AUTODIAL_FAILIFSECURITYCHECK = 4;
```

- ▶ `INTERNET_AUTODIAL_FORCE_ONLINE` bewirkt, dass eine Online-Verbindung erzeugt wird,
- ▶ `INTERNET_AUTODIAL_FORCE_UNATTENDED` führt dazu, dass die Verbindung im Hintergrund geöffnet wird, ohne dass der Anwender im Einwahldialog den WÄHLEN-Schalter betätigen muss,
- ▶ `INTERNET_AUTODIAL_FAILIFSECURITYCHECK` bewirkt, dass `InternetAutodial` fehlschlägt, wenn die Drucker- und Datei-Freigabe deaktiviert ist.

Rückgabe

`InternetAutodial` gibt einen Wert ungleich 0 zurück, wenn die Verbindung erzeugt werden konnte.

Voraussetzungen

Die Funktion ist ab Windows 95 verfügbar, wenn der Internet Explorer ab Version 4 installiert ist.

InternetAutodialHangup

Die Funktion `InternetAutodialHangup` schließt die Standard-DFÜ-Internetverbindung. Leider arbeitet diese Funktion nicht mit einer DSL-Verbindung über eine Netzwerkkarte.

Deklaration

```
[DllImport("wininet.dll", SetLastError=true)]
public static extern int InternetAutodialHangup(int dwReserved);
```

Argumente

Das Argument `dwReserved` ist für eventuelle spätere Zwecke reserviert und wird nicht verwendet.

Rückgabe

Die Funktion gibt einen Wert ungleich Null zurück, wenn die Ausführung erfolgreich war.

Voraussetzungen

`InternetAutodialHangup` ist ab Windows 95 verfügbar, wenn der Internet Explorer ab Version 4 installiert ist.

InternetGetConnectedStateEx

Die `InternetGetConnectedStateEx`-Funktion gibt zurück, ob gerade eine Internetverbindung besteht (leider nicht zuverlässig für Internetverbindungen über ein LAN). Außerdem erhalten Sie Informationen über den Namen der Internetverbindung und über den Status.

Deklaration

```
[DllImport("wininet.dll")]
private static extern int InternetGetConnectedStateEx(out int lpdwFlags,
    StringBuilder lpszConnectionName, int dwNameLen, int dwReserved);
```

Argumente

Am Argument `lpdwFlags` übergeben Sie eine `int`-Variable, die die Funktion mit dem Status der Internetverbindung füllt. Dieses Argument ist eine Bitmaske mit den folgenden Konstanten:

```
private const int INTERNET_CONNECTION_MODEM = 0x01; // Modemverbindung besteht
private const int INTERNET_CONNECTION_LAN = 0x02;  // LAN-Verbindung besteht
private const int INTERNET_CONNECTION_PROXY = 0x04; // Internetverbindung
                // ist für die Verwendung eines Proxy-Servers konfiguriert
private const int INTERNET_RAS_INSTALLED = 0x10;   // RAS ist installiert
private const int INTERNET_CONNECTION_OFFLINE = 0x20; // Verbindung ist
                // offline
private const int INTERNET_CONNECTION_CONFIGURED = 0x40; // Verbindung ist
                // konfiguriert
```

Das Flag `INTERNET_CONNECTION_MODEM` wird zuverlässig gesetzt, wenn die Internetverbindung über ein (Telefon-, ISDN- oder DSL-)Modem oder eine ISDN-Karte erfolgt. Ist keine solche Verbindung konfiguriert, ist normalerweise das Flag `INTERNET_CONNECTION_LAN` immer gesetzt, wenn eine LAN-Verbindung besteht. `INTERNET_CONNECTION_PROXY` ist gesetzt, wenn (z.B. über die Optionen des Internet Explorers) die Verwendung eines Proxy-Servers konfiguriert ist. Dieses Flag ist wie `INTERNET_CONNECTION_LAN` auch dann gesetzt, wenn keine Internetverbindung besteht oder der Proxy-Server nicht ausgeführt wird. Das Flag `INTERNET_CONNECTION_OFFLINE` hat wohl eine andere Bedeutung, als der Name und die Dokumentation vermuten lassen. Bei meinen Versuchen war dieses Flag auch dann nicht gesetzt, als definitiv keine Internetverbindung bestand.

Am Argument `lpszConnectionName` übergeben Sie ein ausreichend groß initialisiertes `StringBuilder`-Objekt, in das die Funktion dann den Namen der Internetverbindung schreibt. Das dritte Argument nimmt dazu die Anzahl der Zeichen im `StringBuilder`-Objekt auf.

Das letzte Argument ist für spätere Zwecke reserviert und wird zurzeit nicht verwendet.

Rückgabe

`InternetGetConnectedStateEx` gibt bei Systemen mit einer konfigurierten Modem-, DSL- oder ISDN-Verbindung einen Wert ungleich Null zurück, wenn eine Internetverbindung verfügbar ist (die aber dazu nicht online sein muss). Ist keine normale Internetverbindung konfigu-

riert, gibt die Funktion auch dann einen Wert ungleich Null zurück, wenn eine LAN-Verbindung verfügbar ist.

IsIconic

IsIconic ermittelt, ob ein Windows-Fenster gerade verkleinert ist.

Deklaration

```
[DllImport("User32.dll", SetLastError=true)]  
public static extern int IsIconic(IntPtr hwnd);
```

Argumente

Am Argument `hwnd` übergeben Sie den Handle des Fensters, dessen Zustand Sie überprüfen wollen.

Rückgabe

IsIconic gibt einen Wert ungleich Null zurück, wenn das Fenster gerade verkleinert ist. Im anderen Fall wird Null zurückgegeben.

LogonUser

LogonUser erzeugt ein User-Token für Benutzer auf dem lokalen Computer. Dieses Token können Sie z. B. verwenden, um einen Thread (bzw. eine Anwendung) über ein `WindowsPrincipal`-Objekt zu personifizieren.

Deklaration

```
[DllImport("advapi32.dll", SetLastError=true)]  
private static extern int LogonUser(string lpszUsername,  
    string lpszDomain, string lpszPassword, int dwLogonType,  
    int dwLogonProvider, out int phToken);
```

Argumente

Am Argument `lpszUsername` geben Sie den Namen des Benutzers an. Das Argument `lpszDomain` nimmt den Namen der Domäne auf, wenn es sich um ein Domänen-Benutzerkonto handelt. Für ein lokales Konto geben Sie einen Punkt an (».«) an. Das Passwort des Benutzers übergeben Sie am Argument `lpszPassword`.

Das Argument `dwLogonType` bestimmt die Art des Logon. Die dazu verwendeten Konstanten werden folgendermaßen deklariert:

```
private const int LOGON32_LOGON_INTERACTIVE = 2;  
private const int LOGON32_LOGON_NETWORK = 3;  
private const int LOGON32_LOGON_BATCH = 4;  
private const int LOGON32_LOGON_SERVICE = 5;  
private const int LOGON32_LOGON_UNLOCK = 7;
```

```
// Die folgenden Konstanten gelten erst ab Windows 2000
private const int LOGON32_LOGON_NETWORK_CLEARTXT = 8;
private const int LOGON32_LOGON_NEW_CREDENTIALS = 9;
```

Tabelle 45 beschreibt die wichtigsten dieser Konstanten.

Logon-Konstante	Bedeutung
LOGON32_LOGON_INTERACTIVE	Der Benutzer wird als interaktiver Benutzer eingeloggt. Ein solches Benutzer-Token besitzt mehr Möglichkeiten als ein einfaches Token, verbraucht deswegen aber auch mehr Ressourcen. Dieser Typ wird für Terminal-Programme, Remote-Zugriffs-Programme und Ähnliches verwendet.
LOGON32_LOGON_NETWORK	Bei diesem Typ werden keine Beglaubigungsdaten im Token gespeichert. Das Token besitzt damit keine Rechte auf dem lokalen System. Ein solches Token wird nur für Netzwerkzugriffe verwendet, da bei diesen lediglich der Name und das Passwort des Benutzers übertragen werden.
LOGON32_LOGON_NETWORK_CLEARTXT	Bei diesem Typ werden neben den für lokale Zugriffe notwendigen Beglaubigungsdaten auch der Benutzername und dessen Passwort im Token gespeichert. Das Token kann damit auf lokale und auf Netzwerkressourcen zugreifen.
LOGON_BATCH	Bei diesem Typ werden ebenfalls keine Beglaubigungsdaten im Token gespeichert. Das Token wird für Zugriffe auf lokale Server verwendet, die eine Authentifizierung über Name und Passwort erfordern.
LOGON32_LOGON_SERVICE	Bei diesem Typ wird das Token für einen Dienst erzeugt.

Tabelle 45: Die wichtigsten Konstanten für den Logon-Typ

Am Argument `dwLogonProvider` bestimmen Sie, welcher Logon-Provider verwendet werden soll. Für einen lokalen Logon macht eigentlich nur die folgende Konstante Sinn:

```
private const int LOGON32_PROVIDER_DEFAULT = 0;
```

Über diese Konstante bestimmen Sie, dass der Default-Provider verwendet wird. Sie können aber auch explizit bestimmen, dass ein älterer Provider eingesetzt wird:

```
private const int LOGON32_PROVIDER_WINNT35 = 1;
private const int LOGON32_PROVIDER_WINNT40 = 2;
```

```
// Die folgenden Konstanten gelten erst ab Windows 2000
private const int LOGON32_PROVIDER_WINNT50 = 3;
```

Am out-Argument `phToken` übergeben Sie schließlich eine int-Variable, in die `LogonUser` den Handle des erzeugten Tokens schreibt.

Rückgabe

`LogonUser` gibt einen Wert ungleich Null zurück, wenn das Token erzeugt werden konnte. Im Fehlerfall gibt die Funktion Null zurück.

Voraussetzungen

Unter Windows 2000 ist es leider erforderlich, dass der Benutzer, unter dessen Konto das Programm ausgeführt wird, das Privileg `SE_TCB_NAME` besitzt. Dieses Privileg vergeben Sie über die Lokalen Sicherheitseinstellungen (SYSTEMSTEUERUNG / VERWALTUNG / LOKALE SICHERHEITSRICHTLINIE / SICHERHEITSEINSTELLUNGEN / LOKALE RICHTLINIEN / ZUWEISEN VON BENUTZERRECHTEN). Dort wird dieses Privileg als EINSETZEN ALS TEIL DES BETRIEBSSYSTEMS bezeichnet (was eine etwas unklare Übersetzung für das originale »Act as part of the operating system« ist).

MessageBeep

`MessageBeep` gibt einen einfachen Piepston oder einen Meldungston aus.

Deklaration

```
[DllImport("User32.dll")]  
private static extern int MessageBeep(uint uType);
```

Argumente

Am Argument `uType` geben Sie den Typ des auszugebenden Tons an. Der Wert 1 steht hier für einen einfachen Piepston. Für spezielle Töne können Sie die folgenden Konstanten verwenden:

```
private const uint MB_ICONHAND = 0x00000010;    // Fehler-Warnton  
private const uint MB_ICONQUESTION = 0x00000020; // Frage-Ton  
private const uint MB_ICONEXCLAMATION = 0x00000030; // Warnmeldungs-Ton  
private const uint MB_ICONASTERISK = 0x00000040; // Informations-Warnton
```

`MessageBeep` spielt je nach der übergebenen Konstante den im System für das jeweilige Ereignis festgelegten Ton ab. Ist eine Soundkarte installiert, werden dazu normalerweise Wave-Dateien abgespielt.

Rückgabe

`MessageBeep` gibt bei einer erfolgreichen Ausführung einen Wert ungleich 0 zurück.

OpenIcon

`OpenIcon` stellt ein minimiertes Fenster wieder in dem vorherigen Zustand her.

Deklaration

```
[DllImport("User32.dll", SetLastError=true)]  
public static extern int OpenIcon(IntPtr hwnd);
```

Argumente

Am Argument `hwnd` übergeben Sie den Handle des Fensters, das Sie wiederherstellen wollen.

Rückgabe

`OpenIcon` gibt einen Wert ungleich Null zurück, wenn die Funktion erfolgreich ausgeführt wurde. Im anderen Fall wird Null zurückgegeben.

QueryPerformanceCounter

`QueryPerformanceCounter` liest den Wert des hochauflösenden Performance-Counters aus, den jedes moderne Motherboard zur Verfügung stellt und der auf der Motherboard-Grundfrequenz basiert. Die Frequenz des Performance-Counters können Sie über `QueryPerformanceFrequency` auslesen.

Deklaration

```
[DllImport("kernel32.dll", SetLastError=true)]  
private extern static int QueryPerformanceCounter(  
    ref long lpPerformanceCount);
```

Argumente

Am Argument `lpPerformanceCount` übergeben Sie eine `long`-Variable, in die die Funktion den aktuellen Zählerstand des Performance-Counters schreibt.

Rückgabe

`QueryPerformanceCounter` gibt einen Wert ungleich Null zurück, wenn der Wert des Performance-Counters ermittelt werden konnte. Im Fehlerfall gibt die Funktion Null zurück. Den Fehlercode können Sie über `Marshal.GetLastWin32Error` auslesen.

QueryPerformanceFrequency

`QueryPerformanceFrequency` liest die Frequenz (Zählungen pro Sekunde) des Performance-Counters des Motherboards aus, sofern dieser verfügbar ist. Diese Frequenz wird für Zeitmessungen über `QueryPerformanceCounter` benötigt.

Deklaration

```
[DllImport("kernel32.dll", SetLastError=true)]  
private extern static int QueryPerformanceFrequency(  
    ref long lpFrequency);
```

Argumente

Am Argument `lpFrequency` übergeben Sie eine `long`-Variable, in die die Funktion den Wert der Performance-Counter-Frequenz schreibt.

Rückgabe

`QueryPerformanceFrequency` gibt einen Wert ungleich Null zurück, wenn der Wert des Performance-Counters ermittelt werden konnte. Im Fehlerfall gibt die Funktion Null zurück. Den Fehlercode können Sie über `Marshal.GetLastWin32Error` auslesen.

ReleaseCapture

`ReleaseCapture` gibt die Maus frei, die von einem Fenster »eingefangen« wurde. Nach dem Aufruf von `ReleaseCapture` werden aktuelle Mauereignisse für das Fenster, in dem diese Funktion aufgerufen wurde, nicht weiter verarbeitet. Damit können Sie z.B. über `SendMessage` andere Mausereignisse simulieren.

Deklaration

```
[DllImport("User32.dll")]  
public static extern int ReleaseCapture();
```

Rückgabe

`ReleaseCapture` gibt einen Wert ungleich Null zurück, wenn der Aufruf erfolgreich war.

SendMessage

Über `SendMessage` können Sie eine Windows-Nachricht an ein Fenster oder an alle Fenster des Systems senden. Das Fenster wertet diese Nachricht wie in Windows üblich in seiner Fenster-Main-Funktion aus. Spezielle Konstanten definieren die Art der Nachricht. Über die Konstante `LB_INSERTSTRING` können Sie einer `ListBox` z.B. einen Eintrag hinzufügen.

Einige spezielle Features von Windows-Fenstern und -Steuerelementen sind nicht über die äquivalenten .NET-Objekte verfügbar. Über `SendMessage` können Sie diese Features häufig dennoch verwenden, indem Sie dem Fenster bzw. dem Steuerelement eine entsprechende Nachricht senden.

Deklaration

```
[DllImport("User32.dll")]  
private static extern int SendMessage(IntPtr handle, int msg, int wParam,  
    int lParam);
```

Argumente

Argument	Bedeutung
<i>handle</i>	Der Handle des Fensters, an das die Nachricht gesendet werden soll, oder <code>HWND_BROADCAST (0xFFFF)</code> , wenn die Nachricht an alle Fenster des Systems gesendet werden soll.
<i>msg</i>	Dieses Argument definiert die zu sendende Nachricht als <code>int</code> -Konstante. Die verfügbaren Konstanten sind in verschiedenen C++-Headerdateien definiert, die Bestandteil des Plattform-SDK sind. In der Datei <i>winuser.h</i> finden Sie die meisten dieser Konstanten.
<i>wParam</i>	Viele Nachrichten erwarten Parameter. Am Argument <i>wParam</i> wird häufig der erste Parameter übergeben.
<i>lParam</i>	Das Argument <i>lParam</i> nimmt bei vielen Nachrichten den zweiten Parameter auf.

Tabelle 46: Die Eigenschaften der *IDbTransaction*-Schnittstelle

Rückgabe

Die Rückgabe der Funktion ist von der Art der Nachricht abhängig und liefert Informationen zur Ausführung der Nachricht.

SetForegroundWindow

`SetForegroundWindow` holt ein Fenster in den Vordergrund.

Deklaration

```
[DllImport("User32.dll", SetLastError=true)]
public static extern int SetForegroundWindow(IntPtr hwnd);
```

Argumente

Am Argument `hwnd` übergeben Sie den Handle des Fensters, das Sie nach vorne holen wollen.

Rückgabe

`SetForegroundWindow` gibt einen Wert ungleich Null zurück, wenn das Fenster in den Vordergrund geholt werden konnte. Im anderen Fall wird Null zurückgegeben.

SetSystemTime

Über die Funktion `SetSystemTime` können Sie das Systemdatum und die Systemzeit setzen.

Deklaration

```
[DllImport("kernel32.dll", SetLastError=true)]
static extern int SetSystemTime(ref SYSTEMTIME lpSystemTime);
```

Argumente

Das Argument `lpSystemTime` ist eine Struktur vom (in *winbase.h* definierten) Typ `SYSTEMTIME`, die die einzelnen Teile der Datums- und Zeitangaben aufnimmt. Diese Struktur wird folgendermaßen deklariert:

```
[StructLayout(LayoutKind.Sequential)]
private struct SYSTEMTIME
{
    public short uYear;           // Das Jahr des Datums
    public short uMonth;         // Der Monat des Datums
    public short uDayOfWeek;     // Wochentag; wird bei SetSystemTime nicht verwendet
    public short uDay;           // Der Tag des Datums
    public short uHour;          // Die Stunden der Zeitangabe
    public short uMinute;        // Die Minuten der Zeitangabe
    public short uSecond;        // Die Sekunden der Zeitangabe
    public short uMilliseconds;  // Die Millisekunden der Zeitangabe
}
```

Rückgabe

`SetSystemTime` gibt einen Wert ungleich Null zurück, wenn die Funktion erfolgreich ausgeführt wurde. Fehlercodes können über `Marshal.GetLastWin32Error` ausgelesen und über `FormatMessage` in eine Beschreibung konvertiert werden.

Voraussetzungen

`SetSystemTime` erfordert unter Windows NT, 2000 und XP, dass der Benutzer, unter dem die Funktion ausgeführt wird, das Privileg `SE_SYSTEMTIME_NAME` besitzt. Dieses Privileg können Sie über die lokalen Sicherheitsrichtlinien in der Systemsteuerung vergeben. Dort wird das Privileg als *Ändern der Systemzeit* bezeichnet.

SHFileOperation

Die Funktion `SHFileOperation` erlaubt das Kopieren, Verschieben, Umbenennen und Löschen von Dateien und Ordnern.

Deklaration

```
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);
```

Argumente

Im einzigen Argument übergeben Sie eine Instanz der Struktur `SHFILEOPSTRUCT`, über die die auszuführende Dateioperation spezifiziert wird. Diese Struktur wird folgendermaßen deklariert:

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
```

```

public IntPtr hwnd;
public int wFunc;
public string pFrom;
public string pTo;
public short fFlags;
[MarshalAs(UnmanagedType.Bool)]
public bool fAnyOperationsAborted;
public IntPtr hNameMappings;
public string lpszProgressTitle;
}

```

Tabelle 47 beschreibt die Felder dieser Struktur.

Feld	Bedeutung
hwnd	Hier können Sie den Handle eines Fensters eintragen, in dem SHFileOperation den Fortschritt beim Kopieren oder Verschieben ausgeben soll.
wFunc	bestimmt die Art der auszuführenden Dateioperation mit den folgenden int-Konstanten: ► FO_MOVE (0x0001): Verschieben ► FO_COPY (0x0002): Kopieren ► FO_DELETE (0x0003): Löschen ► FO_RENAME (0x0004): Umbenennen
pFrom	Dieses Feld wird mit einer oder mehreren Datei- oder Ordnerangaben belegt, die die Quelldateien bzw. -ordner festlegen. Die einzelnen Pfadangaben müssen durch ein Null-Zeichen ('\0') abgeschlossen werden. Am Ende muss ein zusätzliches Null-Zeichen stehen.
pTo	Dieses Feld verwaltet beim Kopieren, Verschieben und Umbenennen ähnlich dem pFrom-Feld Pfadangaben zu Zielordner bzw. Zieldateien. Beim Löschen von Dateien oder Ordnern muss dieses Feld leer bleiben (also einen Leerstring oder null speichern, was per Voreinstellung der Fall ist).
fFlags	In fFlags können Sie mit festgelegten Flag-Konstanten spezielle Einstellungen zur ausgewählten Dateioperation festlegen. Sie können die in Tabelle 48 beschriebenen Konstanten verwenden. Die einzelnen Konstanten müssen Oder-verknüpft werden.
fAnyOperationsAborted	In diesem Feld legt die SHFileOperation-Funktion true ab, wenn der Anwender die Operation abgebrochen hat.
hNameMappings	Dieses Feld können Sie mit dem Handle eines Name-Mapping-Objekts belegen, in das SHFileOperation beim Kopieren/Verschieben automatisch umbenannte Dateien einträgt, wenn die Flags FOF_RENAMEONCOLLISION und FOF_WANTMAPPINGHANDLE gesetzt sind.

Tabelle 47: Die Felder der Struktur SHFILEOPSTRUCT

Feld	Bedeutung
<code>lpszProgressTitle</code>	In diesem Feld können Sie den Titel des Fortschritt-Fensters festlegen, wenn <code>fFlags</code> den Wert <code>FOF_SIMPLEPROGRESS</code> enthält.

Tabelle 47: Die Felder der Struktur *SHFILEOPSTRUCT* (Forts.)

Flag	Bedeutung
<code>FOF_ALLOWUNDO (0x0040)</code>	Rückgängigmachen ermöglichen
<code>FOF_CONFIRMMOUSE (0x0002)</code>	Zurzeit nicht in Verwendung
<code>FOF_FILESONLY (0x0080)</code>	*.*-Angaben nur auf Dateien anwenden
<code>FOF_MULTIDESTFILES (0x0001)</code>	gibt an, dass <code>pTo</code> anstatt eines Zielordners mehrere Zielfordner verwaltet.
<code>FOF_NOCONFIRMATION (0x0010)</code>	Benutzer vor dem Überschreiben von Dateien/Ordner nicht fragen
<code>FOF_NOCONFIRMMKDIR (0x0200)</code>	Keine Bestätigung für zu erzeugende Ordner anfordern
<code>FOF_NO_CONNECTED_ELEMENTS (0x2000)</code>	Verbundene Dateien nicht als Gruppe verschieben. Gilt nur mit installiertem Internet Explorer ab Version 5.
<code>FOF_NOCOPYSECURITYATTRIBS (0x0800)</code>	Sicherheits-Attribute der Datei nicht mit kopieren. Gilt nur mit installiertem Internet Explorer ab Version 4.71.
<code>FOF_NOERRORUI (0x0400)</code>	Keinen Fehlerdialog bei Fehlern anzeigen
<code>FOF_NORECURSION (0x1000)</code>	Nicht rekursiv in Unterordner verzweigen
<code>FOF_RENAMEONCOLLISION (0x0008)</code>	Dateien, die bereits existieren, automatisch neu benennen
<code>FOF_SILENT (0x0004)</code>	Fortschrittsdialog nicht anzeigen
<code>FOF_SIMPLEPROGRESS (0x0100)</code>	Im Fortschrittsdialog keine Dateinamen anzeigen
<code>FOF_WANTMAPPINGHANDLE (0x0020)</code>	Wenn <code>FOF_RENAMEONCOLLISION</code> angegeben ist und Dateien umbenannt werden, steht dieses Flag dafür, dass die umbenannten Dateien in einem Mapping-Objekt festgehalten werden, das im Feld <code>hNameMappings</code> referenziert wird.
<code>FOF_WANTNUKEWARNING (0x4000)</code>	Warnung ausgeben, falls eine Datei beim Löschen zerstört wird. Gilt nur mit installiertem Internet Explorer ab Version 5.

Tabelle 48: Die Flags der *SHFILEOPSTRUCT*-Struktur

Rückgabe

Die Rückgabe der Funktion ist im Erfolgsfall 0. Im Fehlerfall wird ein leider von Microsoft nicht dokumentierter Fehlercode ungleich 0 zurückgegeben.

SHQueryRecycleBin

`SHQueryRecycleBin` ermöglicht die Ermittlung der Größe und die Anzahl der im Papierkorb enthaltenen Dateien eines Laufwerks oder aller Laufwerke.

Deklaration

```
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHQueryRecycleBin(string rootPath,
    ref SHQUERYRBINFO shQueryRBInfo);
```

Argumente

Am Argument `rootPath` können Sie den Wurzel-Pfad eines Laufwerks übergeben (`C:\`, `D:\` etc) wenn Sie den Papierkorb für ein einzelnes Laufwerk erforschen wollen. Wenn Sie einen leeren String übergeben, erhalten Sie die Informationen für den gesamten Papierkorb (außer unter Windows 2000).

Stopp

Unter Windows 2000 führt der Aufruf von `SHQueryRecycleBin` zu dem Fehler `E_INVALIDARG (0x80070057)`, wenn Sie `null` oder einen leeren String übergeben. Dieser Bug ist im Microsoft Knowledge Base-Artikel Q266366 dokumentiert. Als Workaround wird dort vorgeschlagen, zur Ermittlung der Informationen für den gesamten Papierkorb die Infos aus den Papierkörben der einzelnen Festplattenpartitionen auszulesen und zu summieren.

Das Argument `shQueryRBInfo` ist eine Instanz der Struktur `SHQUERYRBINFO`, in die `SHQueryRecycleBin` die ermittelten Informationen schreibt. Diese Struktur wird folgendermaßen deklariert:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
private struct SHQUERYRBINFO
{
    public int cbSize;
    public long i64Size;
    public long i64NumItems;
}
```

Tabelle 49 beschreibt die Felder dieser Struktur.

Feld	Bedeutung
<code>cbSize</code>	In dieses Feld müssen Sie die Größe der Struktur (in Bytes) schreiben, damit die API-Funktion die Daten korrekt schreiben kann. Die Größe erhalten Sie über <code>Marshal.SizeOf(typeof(SHQUERYRBINFO))</code> .
<code>i64Size</code>	In dieses Feld schreibt <code>SHQueryRecycleBin</code> die Gesamtgröße aller im Papierkorb enthaltenen Dateien (in Bytes).
<code>i64NumItems</code>	Dieses Feld speichert nach dem Aufruf von <code>SHQueryRecycleBin</code> die Anzahl der im Papierkorb enthaltenen Dateien.

Tabelle 49: Die Felder der Struktur `SHQUERYRBINFO`

Rückgabe

SHQueryRecycleBin gibt S_OK (0) zurück, wenn der Aufruf erfolgreich war. Im Fehlerfall wird laut der Dokumentation ein »OLE-definierter Fehlercode« zurückgeliefert, ohne dass diese beschreibt, was damit genau gemeint ist.

sndPlaySound

sndPlaySound spielt eine Wave-Datei ab.

Deklaration

```
[DllImport("winmm.dll", SetLastError=true)]  
private extern static int sndPlaySound(string lpszSound, int fuSound);
```

Argumente

- ▶ lpszSound: Der Dateiname der Wave-Datei oder der Namen eines Registry-Eintrags der vordefinierten Windows-Sounds.
- ▶ fuSound: Eine Kombination verschiedener Flags, die den Abspielvorgang steuern.

Am Argument fuSound können Sie die folgenden Flags übergeben:

Flag	Wert	Bedeutung
SND_SYNC	0x0000	Der Sound wird synchron abgespielt. Dieses Flag ist aufgrund des Werts von 0 ziemlich sinnlos.
SND_ASYNC	0x0001	Der Sound wird asynchron abgespielt und sndPlaySound kehrt direkt zurück.
SND_MEMORY	0x0004	lpszSound zeigt auf eine Sounddatei im Speicher.
SND_LOOP	0x0008	Der Sound wird am Ende wiederholt.
SND_NOSTOP	0x0010	Der Sound kann nicht durch einen wiederholten Aufruf von sndPlaySound gestoppt werden (was ansonsten möglich ist).
SND_NODEFAULT	0x0002	Bei nicht gefundenen oder nicht unterstützten Dateien wird nicht der Windows-Defaultsound abgespielt.

Tabelle 50: Flags für das fuSound-Argument

Rückgabe

sndPlaySound gibt einen Wert ungleich Null zurück, wenn der Aufruf erfolgreich war.

Stichwortverzeichnis

!

#ziplib 337

A

Abs 713

Abschalten, des Monitors 452

Absolutwert berechnen 713

Acos 713

Activator 52

Active Directory 67

activeds.tlb 557

Add 714

AddDays 712

AddHours 712

AddMinutes 711

AddMonths 712

AddNamespace 382

AddressList 467

AddYears 712

ADSI 67, 547

Benutzer anlegen 570

Benutzer aus Gruppe entf. 575

Benutzer auslesen 547

Benutzer löschen 576

Benutzer zu Gruppe hinzufg. 573

Benutzereigenschaften 562

Benutzergruppen 560

Container durchgehen 79

Eigenschaften lesen 76

Eigenschaften schreiben 77

Fehlerauswertung 76

Gruppen anlegen 568

Gruppen ermitteln 553

Gruppen löschen 577

Gruppenbenutzer ermitteln 556

Instanzen erzeugen 75

Pfad 71

ADSI-Pfad 71

Ändern

des Systemmenüs 491

von Textdateien 331

XML-Dateien 392, 403

Aktuelle Zeit siehe Systemzeit

AllowDrop 532

AllowNull 521

Alphawert 646

Alter berechnen 152

ALTER TABLE 692

Andockende Formulare 492

Anhalten, Programm 141

Anlegen

Benutzer 570

Benutzergruppe 568

Anwendung

aktivieren 223f, 226

beenden 242

Dateiname ermitteln 195, 249

für assoziierte Dateien starten 239

Java-Anw. starten 243

konfigurieren 197f, 202, 204

Mehrfachen Start verhindern 218

Ordner ermitteln 252

starten 239

starten und auf Ende warten 240

Verzeichnis ermitteln 196

Anwendungskonfiguration siehe Konfiguration

Anzahl

Datensätze 688

Tage im Monat 154

Wörter in String 114

AppActivate 223

AppendChild 394, 726

Application 195, 249, 252

AppSettings 197

appSettings 197

Arbeitsspeicher

Größe ermitteln 414

überwachen 437, 440, 442

Archive

aus Streams 345

entpacken 348

erzeugen 337

Inhalt lesen 352

Ordner packen 343

Arithmetische Operatoren 703

- Array 707
- ArrayList 129
 - sortieren 129
- Arrays, sortieren 129, 133
- Asin 713
- Assemblies, zur VS-Verw.-Liste 338
- Atan 713
- Atomuhr-Datum 167
- Attachments 469, 715
- Attributes 305, 725
- Audio 618
- Auflistungen
 - Typisierte 135
 - Typsichere 290
- Auflistungsklassen 714
- Aufrufer, einer Methode 677
- Aufzählungen
 - Konst. als Strings 124
 - Wert aus String 125
- Ausschnitt, eines Bildes 636
- AVI-Dateien
 - über DirectX absp. 617
 - über MCI absp. 605

B

- Base64-Codierung 335f
- Bcc 715
- Bedingungsoperator 704
- Beep 103
- Benutzer
 - anlegen 570
 - aus einer Gruppe entf. 575
 - auslesen 547
 - Eigenschaften lesen 562
 - einem Progr. zuweisen 578
 - einer Gruppe 556
 - einer Gruppe hinzufg. 573
 - Existenz prüfen 552
 - Gruppen abfragen 560
 - löschen 576
 - Name des aktuellen 577
- Benutzergruppen siehe Gruppen
- Bilder
 - aus Byte-Array 621
 - aus d. Clipboard 621
 - Ausschnitte 636
 - drehen 632
 - einzelne Pixel ändern 644

- Farben verändern 637
- Graustufen 645
 - in Byte-Array 620
 - in einem DataGrid 523
 - in einer Datenbank 693
- konvertieren 629
- mit Schatten 648
- Negativ 643f
- Qualität beim Speichern 629
- skalieren 625
- spiegeln 632
- Thumbnails 628
- Transparente 646
- Bildschirm, Screenshot 622
- Bildschirm siehe Monitor
- Binäre Dateien
 - aus Base64-Strings 336
 - Base64-codieren 335
 - lesen 332
 - schreiben 334
- Binäre Daten
 - aus XML lesen 406
 - in einer Datenbank 693
 - in XML 404
- BinaryFormatter 680
- BinarySearch 133, 707
- BindingFlags 53
- BitBlt 622
 - Referenz 727
- Bitmap 352
- Bitmaps siehe Bilder
- Bitweise Operatoren 703
- BLOB 693
- Body 715
- BodyEncoding 716
- BodyFormat 716
- bool 701
- Booten, des Computers 429
- break 700
- Breite eines Textes 653
- Browser
 - in Anw. integrieren 474
 - starten 473
- Bubblesort 519
- byte 701
- Byte-Arrays
 - aus binärer Datei 332
 - aus Strings erz. 126

binäre Datei erz. 334
in Bitmap umw. 621
in String umw. 128
Byte-Werte, lesbares Format 93

C

Calendar 155
CalendarWeekRule 157
CallingConvention 32
CanPauseAndContinue 234
CBT-Hook 537
Cc 716
char 701
CharSet 32, 272
ChildNodes 725
Children 79, 568
CIM 64
ClassDefNotFoundError 244
Classpath 244
Clear 714
Clipboard, Bitmaps lesen 621
Clone 707
Codierung 126
CollectionBase 135, 278, 288, 421
Collections siehe Auflistungen
ColorMap 637f
ColorMatrix 637
ComboBox
 Auto-Complete 510
 in DataGrid 525
 Zusätzl. Daten 512
COM-Komponenten
 laufende Instanzen ref. 655
 späte Bindung 51
CommandStateChange 480
CommitChanges 78, 569f
Common Name 73
Compare 710
CompareTo 181
Computer
 booten 429
 Name ermitteln 409
config-Datei
 eigene Handler 198
 lesen 197f
 schreiben 202
ConfigurationSettings 197
Contains 714
Continue 233

continue 700
Convert 336, 706
Copy 295
CopyTo 707
Cos 713
CounterCreationData 447
CounterCreationDataCollection 447
CPU, Geschwindigkeit 413
Create 198
CREATE DATABASE 690
CREATE TABLE 691
CreateAttribute 395, 723
CreateComment 723
CreateDecryptor 593
CreateDirectory 269
CreateElement 394, 723
CreateEncryptor 592
CreateGraphics 624
CreateInstance 52
CreateNavigator 726
CreateSpecificCulture 91, 146
CreateSubKey 212, 215
CreateTextNode 724
CreateXmlDeclaration 394, 724
CryptoStream 592
CultureInfo 91, 146, 155
CurrentCulture 91
CurrentDirectory 244
CurrentThread 91

D

DataGrid
 anpassen 520
 Bilder im Grid 523
 mit ComboBox 525
 Readonly-Spalten 521
DataGridBoolColumn 521
DataGridTableStyle 521
DataGridTextBoxColumn 521, 526
DataSet 385, 401
DataSetName 401
DateAndTime 151
DateDiff 151
Dateiattribute lesen 305
Dateien
 8.3-Namen lesen 307
 Attribute lesen 305
 aus (ZIP-)Archiven 348
 Base64-codierte 335f

- Binäre lesen 332
- Binäre schreiben 334
- downloaden 481
- Drag & Drop 532
- eines Ordners aufl. 309
- Endung ändern 256
- Existenz prüfen 293
- Größe ermitteln 304
- in Papierkorb versch. 317
- in Sys.-Pfad suchen 294
- komprimieren 337
- kopieren 295
- löschen 298
- mit assoziierter Anw. öffnen 239
- suchen 310
- Textdateien ändern 331
- Textdateien lesen 325
- Textdateien schreiben 328
- Texte anfügen 330
- umbenennen 297, 313
- vergleichen 299
- Verknüpfungen 322
- verschieben 298, 313
- Version auslesen 306
- XML transf. 407
- XML-Dateien ändern 403
- XML-Dateien lesen 355, 371, 375, 385
- XML-Dateien schreiben 389, 392, 401
- Dateiname
 - der Anwendung 195
 - einer Klassenbibliothek 250
 - einer Windows-Anw. 249
 - für temporäre Datei 253
- Daten
 - entschlüsseln 584
 - Hashs erzeugen 597
 - verschlüsseln 584
 - zus. in Listbox 512
- Datenbanken
 - Anzahl Datensätze 688
 - Bilder verwalten 693
 - Binäre Daten 693
 - erzeugen 689
- Datensätze, autom. Id abfragen 692
- Datentypen 701
- DateTime 154, 156, 711
- Datum
 - addieren, subtr. 153
 - Aktuelles 143
 - Differenz 149f, 152
 - formatieren 143, 702
 - Gültigkeit 97
 - inWMI 64
 - ISO-Datum 147
 - Kalenderwoche 156
 - Länderformate 145
 - normalisieren 174
 - Systemdatum setzen 164
 - vergleichen 148
 - von Atomuhr 167
- Day 712
- DayOfWeek 157, 712
- DayOfYear 712
- DaysInMonth 154, 712
- decimal 701
- Decrement 449
- default 700
- Delete 216, 298
- DeleteMenu 491
 - Referenz 729
- DeleteSubKey 216
- DES 586
- DESCryptoServiceProvider 586
- Deserialisieren 680
- Deserialize 680
- Device Context 622
- DictionaryBase 135, 206
- Dienste
 - anhalten 233
 - Informationen auslesen 230
 - starten 233
 - stoppen 233
- Differenz, Datumswerte 149, 150, 152
- Directory 269, 285
- DirectoryEntry 548, 556, 568, 570, 576
 - Eigenschaften lesen 76
 - Eigenschaften schreiben 77
 - Erzeugen von Instanzen 75
- DirectoryInfo 268, 287, 309f
- DirectoryServices 75
- DirectX 617
- DllImport 31, 266
- DllImportAttribute 31
- DocumentElement 398
- Documents 658
- Dokumentenvorlagen, verwenden 660
- DOM 371, 392

Domäne
 Benutzer auslesen 547
 Gruppen auslesen 553
double 701
Downloaden, Dateien 481
DownloadFile 481
Drag & Drop, Dateien & Ordner 532
DragDrop 533
DragEnter 532
Drehen
 einer Zeichnung 633
 von Bildern 632
Drive 258
DriveLetter 258
DriveType 258
Duale Darstellung 84

E

Eigenschaften
 Array 707
 Auflistungen 714
 DateTime 711
 MailMessage 715
 Math 713
 Regex 717
 SmtptMail 722
 Standardtypen 705
 String 708
 XmlNode 725
E-Mails
 über MAPI vers. 471
 über SMTP verse. 469
EnablePrivileges 429
EncoderParameter 630
EncoderParameters 630
Encoding 126, 325
EndsWith 708
Energiesparmodus, Monitor 452
Entry 715
EntryPoint 32
Enum 124
Enumerationen siehe Aufzählungen
Environment 253, 256, 294, 409
Equals 705
Erzeugen
 Binäre Dateien 334
 Datenbanken 689
 Textdateien 328
 XML-Dateien 389, 392, 401

Escape 119, 717
Escape-Sequenzen 700
ExactSpelling 32
Excel-Arbeitsmappen
 erweitern 669
 erzeugen 664
ExecutablePath 249
Existenz
 einer Datei 293
 eines Benutzers prüfen 552
 eines Ordners 268
Exists 268, 293
Exp 713
Express ClickYes 473, 657

F

Farben, eines Bilds ändern 637
Fehlerrückmeldung, ADSI 76
Feiertage
 berechnen 178
 Datum = Feiertag 188
File 268, 293, 295, 297, 325
FileInfo 293, 304f, 325, 332
FileStream 332, 334, 340
FileSystemObject 257
FileVersionInfo 307
FindWindow 224
 Referenz 730
Finish 340
FirstChild 725
FirstDayOfWeek 151
float 701
FolderBrowserDialog 534
FolderNameEditor 533
Folders 675
Font-Objekte serialisieren 687
Format 144
Formatierungen
 Datumswerte 702
 Zahlen 702
FormatMessage 45, 266
 Referenz 730
Formatting 389
FormBorderStyle 489
Formulare
 Andockende 492
 ohne Titelleiste 489
 Rahmenlose bewegen 489
 Schließen-Schalter entf. 493

Screenshot 622
 Spezielle Formen 495
 Splash-Forms 500
 Sys-Menü ändern 493
 über Clientbereich versch. 489
 Unbewegbare 491
 verlaufend füllen 494
 Freier Speicherplatz, eines Laufwerks 266
 Freigaben, Netzwerk 66
 From 716
 FromBase64String 336, 406

G

Gedreht zeichnen 649
 GenericTypographic 653
 Geschwindigkeit, des Prozessors 413
 GetActiveObject 655
 GetAttribute 356
 GetBytes 126
 GetCategories 432
 GetClassName 731
 GetCounters 432
 GetData 621
 GetDataPresent 532
 GetDaysInYear 155
 GetDirectories 287
 GetDiskFreeSpaceEx 266, 732
 GetDriveType 261
 Referenz 732
 GetElementById 381, 724
 GetElementsByTagName 372, 724
 GetEncoding 325
 GetEntryAssembly 195, 249
 GetEnvironmentVariable 294
 GetExecutingAssembly 249f
 GetField 53
 GetFiles 287, 310
 GetFolderPath 254, 409
 GetFrame 677
 GetGroupNames 718
 GetGroupNumbers 718
 GetHashCode 705
 GetHdc 623
 GetHostByName 467
 GetInstanceNames 432
 GetInstances 60
 GetLastError 43
 GetLastWin32Error 226, 266

GetLength 707
 GetLogicalDrives 256
 GetName 124
 GetNames 124
 GetNextEntry 346
 GetNonZeroBytes 123
 GetProcesses 227, 229
 GetProcessesByName 219, 237, 242
 GetProperty 53
 GetResponse 481f
 GetResponseStream 481
 GetShortPathName 308
 Referenz 733
 GetString 128
 GetSubKeyNames 418
 GetSystemMenu, Referenz 733
 GetTempFileName 253
 GetTempPath 253
 GetType 705
 GetTypeFromCLSID 52
 GetTypeFromProgID 52
 GetValue 212
 GetVersionInfo 307
 GetWeekOfYear 157
 GetWindowsDirectory 254, 409, 734
 GNU General Public License 337
 GoBack 477
 GoForward 477
 GoHome 477
 Google, Newsgroup-Recherche 23
 goto 700
 GPL-Lizenz 337
 Grafiken
 schräg ausgeben 649
 Transparente 646
 verlaufend füllen 494
 Graphic 649
 Graphics 626, 632, 638, 641
 GraphicsPath 497
 Graustufen 645
 Größe
 aller Unterordner 288
 des Arbeitsspeichers 414
 einer Datei 304
 eines Laufwerks 266
 eines Ordners 287
 GroupNameFromNumber 718
 GroupNumberFromName 718

Gruppen

- anlegen 568
- auslesen 553
- Benutzer entfernen 575
- Benutzer ermitteln 556
- Benutzer hinzufügen 573
- eines Benutzers 560
- löschen 577

Gruppierungen 721**Gültigkeit**

- Datumseingabe 97
- von XML prüfen 365
- von Zahlen überpr. 94

Guid 608**GZIP-Dateien siehe Archive****H****Handler, für Konfig.-Daten 198****HashAlgorithm 598****Hashcodes 597****Hashing-Verfahren 597****Hashtable, Typsichere 138****Headers 716****HeaderText 521****Herunterfahren siehe Booten****Herunterladen, Dateien 481****High-Word 101f****Hintergrund, transparenter 495****Höhe eines Textes 653****Hooking 537****Hour 712****HTML-Dokumente**

- in Anw. darstellen 474
- in Browser öffnen 473

I**IAds 69****IAdsContainer 70****IADsGroup 557****IADsUser 558, 560****ICMP, Prüfsumme berechnen 461****IComparable 129, 181****IConfigurationSectionHandler 198****Icons**

- aus Byte-Array 621
- in Byte-Array 620

ICryptoTransform 592**IFormatProvider 92****IgnoreCase 53****IHTMLDocument2 478****ICollection 714****ImageAttributes 638****ImageCodecInfo 630****Impersonation 578****ImportNode 724****Increment 449****Indentation 389****IndentChar 389****IndexOf 707f, 714****IndexOfAny 709****Initialize 707****InnerText 372, 725****InnerXml 725****Insert 709, 714****InsertAfter 726****InsertBefore 394, 726****Instance 53****InstanceName 441****int 701****Internet****Verb. öffnen/schließen 463****Verbindung prüfen 458****InternetAutodial 463****Referenz 734****InternetAutodialHangup 463****Referenz 735****InternetGetConnectedStateEx 455, 458****Referenz 736****Internetverbindung, Status abfragen 455****InteropServices 31****InterpolationMode 626****Invoke 570, 573****InvokeMember 53****InvokeMethod 53, 65****IP-Adressen, ermitteln 467****IPHostEntry 467****IShellLink 322****IsIconic 224, 737****IsLeapYear 156, 712****IsMatch 718****ISO-8601 147, 156****ISO-Datum 147****is-Operator 704****IsReadOnly 725**

J

java 243
Java-Anw. starten 243
Java-Interpreter 243
javaw 243
JPEG-Bilder, Qualität 629

K

Kalenderwoche
 eines Datums 156
 Jahres-Anzahl 163
 Startdatum 160f
Key 715
KeyedHashAlgorithm 600
Kill 242
Klassenbibliothek
 Dateiname ermitteln 250
 Ordner ermitteln 252
Knowledge Base 26
Komprimieren, Ordner und Dateien 337
Konfiguration, eigene Klasse 204
Konfigurationsdaten
 aus .config lesen 197f
 Fonts verwalten 687
 in .config speichern 202
 in XML-Datei speichern 204
Kontakte, aus Outlook 673
Konvertieren
 nach JPEG 629
 von Bildern 629
Kopieren
 eines Ordners 269, 271
 von Dateien 295
Krüger, Mike 337
Kürzen, von Strings 117
Kultur umstellen 91

L

LastChild 725
LastIndexOf 709
LastIndexOfAny 709
Laufwerke
 ermitteln 256
 Freier Speicherplatz 266
 Größe ermitteln 266
 Typ ermitteln 257
LDAP 67
Leerzeichen, in ein Leerz. 110

Leistungsindikatoren 430

 auflisten 431
 Eigene implementieren 447
 Speicher 437
 Speicher und CPU 440

Length 304, 708

Lesen

 Anw.-Konfiguration 197
 aus der Registry 212
 Binäre Dateien 332
 Konfigurationsdaten 197
 von Konfig.-Daten 198, 204
 von Textdateien 325
 XML-Dateien 371, 375, 385

ListBox, Zusätzl. Daten 512

ListView, sortieren 514

Load 375, 724

LoadXml 209, 724

LocalName 725

lock 678

Löschen

 Benutzer 576
 Benutzergruppe 577
 in den Papierkorb 317
 von Dateien 298
 von Ordnern 286

Log 713

Logische Laufwerke, ermitteln 256

Logische Operatoren 704

LogonUser 578, 737

long 701

Low-Word 101f

M

MachineName 409

MailAttachment 469

MailFormat 716

MailMessage 469

 Referenz 715

MailPriority 716

Mails

 über MAPI vers. 471
 über SMTP vers. 469

MainWindowTitle 229

MakeTransparent 646

Mallen, Ronald W. 179

ManagementBaseObject 60

ManagementClass 60

ManagementObject 57, 413f, 429

ManagementObjectCollection 60
ManagementObjectSearcher 61
ManagementScope 63, 429
MAPI 471
MappingName 521
Marshal 226, 266, 655
MarshalAs 34, 273
Marshalling 33
Match 718
Matches 114, 121, 719
MatchEvaluator 112
Math 89, 713
Matrix 633, 650
Mattias Sjögren 322
Max 713
MaxValue 705
MCI 605
mciGetErrorString 605
mciSendCommand 605
MD5 597
MeasureString 653
Meldungstöne 103
MemoryStream 352
Menüs, Laufzeit 502
MenuItem 502
MenuItems 502
Message Digest 597
MessageBeep 103
 Referenz 739
MessageBox, Schalterbeschr. ändern 536
Messen, Zeit 189
Metafile
 Aus Byte-Array 621
 In Byte-Array 620
Methoden
 Array 707
 Auflistungen 714
 Aufrufer ermitteln 677
 Convert 706
 DateTime 711
 Math 713
 object 705
 Regex 717f
 SmtpMail 722
 Standardtypen 705
 String 708
 XmlDocument 723
 XmlNode 726

Microsoft Platform SDK 29
Midi-Dateien
 über DirectX absp. 617
 über MCI absp. 605
Min 713
Minute 712
MinValue 705
Missing.Value 50, 658
Monat, Anzahl Tage 154
Monitor
 abschalten 452
 einschalten 452
 Energiesparmodus 452
Month 712
Move 285, 297
MoveToContent 357
MP3
 über DirectX absp. 617
 über MCI absp. 605
MPEG-Dateien
 über DirectX absp. 617
 über MCI absp. 605
MPEGVideo-Treiber 608
Mutex 221

N

Name
 Aktueller Benutzer 577
 des Computers 409
Namensräume, WMI 57
Namespace 401
NamespaceURI 725
NameTable 382
Negativ, eines Bildes 643
Netzwerkfreigaben, erzeugen 66
NewGuid 608
Newsgroup-Recherche 23
NextSibling 726
NIST 167
NodeType 726
Normalisieren, eines Datums 174
Now 713
NumberStyles 96

O

object 705
Objekte
 deserialisieren 680
 serialisieren 680

Office

- Excel-Arbeitsmappen 664, 669
- Excel-Dat. erweitern 669
- Excel-Mappen erz. 664
- Outlook-Kontakte 673
- Word-Dokumente 657, 660
- Word-Dokumente erz. 660
- Word-Dokumente öffnen 657

OpenIcon 224, 739

OpenRead 332

OpenSubKey 212

OpenText 325

OpenWrite 334

Operatoren

- Arithmetische 703
- Bitweise 703
- Logische 704
- Spezielle 704
- Zuweisungen 703f

Ordner

- archivieren 343
- Dateien auflisten 309
- Dialog 533
- Drag & Drop 532
- einer Klassenbibl. 252
- einer Windows-Anw. 252
- erzeugen 269
- Existenz prüfen 268
- für temp. Dateien 253
- Größe aller Unterordner 288
- Größe ermitteln 287
- in Archiv packen 343
- in Papierkorb versch. 317
- komprimieren 337, 343
- kopieren 269, 271
- löschen 286
- umbenennen 285, 313
- vergleichen 301
- Verknüpfungen 322
- verschieben 285, 313
- Windows-Ordner 253

Ordner siehe Verzeichnis

Organization 73

Organizational Unit 73

Ostern berechnen 179

Outlook

- E-Mails versenden 471
- Kontakte auslesen 673

P

PadLeft 709

Papierkorb

- Anzahl der Dateien erm. 319
- Dateien verschieben 317
- Größe ermitteln 319

Parallele Schnittstellen, auflisten 419

Param 630

ParentNode 726

Parse 172, 175, 706, 712

ParseExact 713

Passwort

- per Zufall berechnen 123
- verschlüsseln 584, 597

Path 253

Path-Umgebungsvariable 294

Pause 233

Performance Counter siehe Leistungsindikatoren

Performance-Counter 190

PerformanceCounter 431f, 437, 441, 449

PerformanceCounterCategory 431f, 447

Personifizierung 578

Pfad siehe Ordner

Pfad siehe Verzeichnis

PI 713

Ping 458

PIInvoke 31

Pixel

- Einzelne bearbeiten 644
- transformieren 637, 645

Platform 411

PlaySound 603

Potenz einer Zahl 713

Pow 713

Prefix 401

PreserveSig 33

PreviousSibling 726

Priority 716

Process 227, 229, 239, 242, 473

ProcessName 229, 441

Programme

- anhalten 141

- Installierte auflisten 417

- Personifizierung 578

Properties 549, 568

PropertyValueCollection 568

Prozesse

- aktivieren 226
- Besitzer herausfinden 66
- Laufende ermitteln 227
- personifizieren 578

Prozessor

- Geschwindigkeit 413
- überwachen 440, 442

Prüfen von XML 365

Prüfsumme, TCP und ICMP 461

Public 53

PutNextEntry 340

Q

Quadratwurzel berechnen 714

Qualität von Bildern 629

Quantifizierer 721

QueryPerformanceCounter 190

- Referenz 740

QueryPerformanceFrequency 190

- Referenz 740

R

Rahmenlose Forms 489

- bewegen 489

RAM siehe Arbeitsspeicher

RawValue 449

RC2 586

RC2CryptoServiceProvider 586

Read 325, 340

ReadLine 325

ReadOnly 521

ReadToEnd 325

ReadXml 386

ReadXmlSchema 386

Reboot 429

Referenzliste, Assemblies hinzufügen 338

Reflektion 249

Regex 106, 112, 121

- Referenz 716

Region 497

Registrierdatenbank siehe Registry

Registry

- Daten auslesen 212
- Daten schreiben 212

RegistryKey 212, 418

Reguläre Ausdrücke, Referenz 716

Rekursion 301

ReleaseCapture 490

- Referenz 741

Remove 709, 714

RemoveAll 726

RemoveAt 714

RemoveChild 398, 726

Replace 106, 709

ReplaceChild 726

ResetTransform 650

Resolve 459

return 700

Rijndael 586

RijndaelManaged 586

RNGCryptoServiceProvider 99, 123

Ronald W. Mallen 179

RotateAt 634

RotateTransform 633, 650

Rotieren von Bildern 632

Round 89, 714

Runden 714

- kaufmännisch 89

S

Save 725

SAX 355

sbyte 701

Schaltjahr ermitteln 156

Schatten für ein Bild 648

SchemaClassName 549

Schemas 362

Schräg zeichnen 649

Schreiben

- Anwendungskonfiguration 202

- Binäre Dateien 334

- in die Registry 212

- Konfigurationsdaten 202

- Textdateien 328

- von Konfig.-Daten 204

- XML-Dateien 389, 392, 401

Screenshot 622

SE_SHUTDOWN_NAME 430

SE_TCB_NAME 581, 739

Second 712

section 198

sectionGroup 198

Secure Hash Algorithm 597

Select 459

SelectedPath 534

SelectNodes 377, 727

SelectSingleNode 372, 377, 727
SendMessage 490
 Referenz 741
SentTo 459
Serialisieren 680
 beliebige Objekte 687
 Font-Objekte 687
Serialize 680
ServiceController 230, 233
SetComment 340
SetDlgItemText-Funktion 537
SetField 53
SetForegroundWindow 224, 742
SetLastError 33, 43, 266
SetLevel 340
SetPassword 570
SetProperty 53
SetRemapTable 638
SetSystemTime 164
 Referenz 742
SetValue 212
SHA1 597
SharpZipLib 337
ShellLink.tlb 322
ShellShortcut 323
SHFileOperation 271, 317
 Referenz 743
SHFILEOPSTRUCT 272
 Referenz 744, 746
short 701
ShowNewFolderButton 534
SHQueryRecycleBin 319f
 Referenz 745
Shutdown 429
Sicherheit, Personifizierung 578
Sin 713
Size 287
Sjögren, Mattias 322
Skalieren von Bildern 625
Smtplib 469
 Referenz 722
SMTP-Server, Mails vers. 469
Smtplib 722
sndPlaySound 603
 Referenz 747
Socket 459
Sort 129, 708

Sortieren
 Arrays 133
 ListView 514
 StringCollection 131
 TreeView 518
Sorting 514
Sounddateien
 Midi 605, 617
 MP3 605, 617
 über DirectX absp. 617
 über MCI absp. 605
 Wave 603, 605, 617
Späte Bindung (COM) 51
SpecialFolder 254
Spezielle Operatoren 704
Spiegeln von Bildern 632
Split 709, 720
Spurverfolgung siehe Tracing
SQL
 Datenbanken erz. 689
 in WMI 61
Sqrt 714
StackFrame 677
StackTrace 677
Standardtypen 701
 Methoden 705
Start 233, 239
Starten
 Anwendung 239f
 Browser 473
 Java-Anwendung 243
StartInfo 474
StartsWith 709
StartupPath 196, 252
Static 53
Steuerelemente
 in VS integrieren 506
 Transp. Hintergr. 504
Stop 233, 477
StreamReader 325
StreamWriter 328, 330
String 708
string 701
StringCollection, sortieren 131
Strings
 Anzahl Wörter 114
 Auftrennen 709

- aus Byte-Array 128
- aus Enum 124
- definiert kürzen 117
- entschlüsseln 584
- ersetzen 105
- in Byte-Array umw. 126
- in Enum-Wert 125
- Linker Teilstring 120
- mehrf. Leerz. entf. 110
- Rechter Teilstring 120
- Teilstrings ersetzen 709
- Teilstrings extrahieren 709
- Teilstrings suchen 708
- über Regex ersetzen 112
- vergleichen 104, 710
- verschlüsseln 584
- Whitespaces entfernen 710
- Wörter extrahieren 116
- Wörter normalisieren 111
- Zahl am Anfang? 122
- Zahlen auslesen 121
- Zufalls-Strings 123
- StructLayout 41, 272
- Subject 716
- Substring 120, 709
- Suchen
 - in Newsgroups 23
 - von Dateien 310
- SymmetricAlgorithm 587
- Symmetrische Verschl. 586
- System booten 429
- Systemdatum
 - lesen 143
 - setzen 164
- SystemDirectory 253, 409
- Systemmenü, Schließen-Schalter entf. 491
- Systemmonitor 431
- Systemzeit, setzen 164

T

- TableStyles 521
- Tan 713
- Tar-Dateien siehe Archive
- TargetInvocationException 573
- Tauschen, Knoten in TreeView 516
- TCP, Prüfsumme berechnen 461
- Teilstring, Links/Rechts 120
- Temp-Ordner ermitteln 253

- TextBoxen
 - Auto Select 505
 - Eingabe einschränken 506
- Textdateien
 - ändern 331
 - lesen 325
 - schreiben 328
 - Texte anfügen 330
- Texte, Höhe und Breite 653
- TextureBrush 638, 641
- Texturen, transparente 641
- Threads
 - Daten übergeben 485
 - Parameter übergeben 678
 - Rückgabewerte 678
- ThreadStart 678
- Thumbnails 628
- Ticks 190
- TitleChange 480
- To 716
- ToBase64String 336
- ToCharArray 710
- Today 713
- Töne, für Meldungen 103
- ToLongDateString 712
- ToLongTimeString 712
- ToLower 710
- ToShortDateString 712
- ToShortTimeString 712
- ToString 705
- ToTitleCase 111
- ToUniversalTime 164
- ToUpper 710
- Trace-Protokolle 442
- Tracing, RAM- und CPU-Auslastung 442
- Transform 408, 633f
- Transformation 649
- Transformieren
 - Pixel 637, 645
 - XML-Dateien 407
 - Zeichenmatrix 633
- Translate 634
- TranslateTransform 633, 650
- Transp. Hintergrund 495
- Transparente
 - Bilder 646
 - Steuerelemente 504

- TreeView
 - Knoten sortieren 518
 - Knoten tauschen 516
- Trim 710
- TrimEnd 710
- TrimStart 710
- TripleDES 586
- TripleDESCryptoServiceProvider 586
- TryParse 706
- Typ, eines Laufwerks ermitteln 257
- Type 52
- typeof-Operator 704
- Typisierte Auflistungen 135, 290

U

- Überprüfen
 - Gültigkeit Datum 97
 - Gültigkeit v. Zahlen 94
- Überwachen
 - Arbeitsspeicher 437, 440, 442
 - CPU 440, 442
- Uhrzeit, Aktuelle 143
- uint 701
- ulong 701
- Umbenennen
 - von Dateien 297
 - von Ordnern 285
- Unbewegbare Formulare 491
- Undo 578
- Unescape 717
- UnmanagedType 38
- UrlContentBase 716
- UrlContentLocation 716
- UserName 577
- UseShellExecute 240
- ushort 701
- using 614

V

- ValidationEventHandler 366, 376
- ValidationType 365
- Validieren, von XML 365
- ValidKeySize 589
- Value 715, 726
- Variant 50
- Vergleichen
 - Datumswerte 148
 - von Dateien 299
 - von Ordnern 301
 - von Strings 104

- Vergleichsoperatoren 704
- Verknüpfungen anlegen 322
- Verschieben
 - von Dateien 298
 - von Ordnern 285
- Verschlüsseln
 - Hashcodes 597
 - symmetrisch 584, 586
- Version
 - einer Datei auslesen 306
 - Windows-Vers. 411
- Verzeichnis, der Anwendung 196
- Verzeichnis siehe Ordner
- Video 619
- Videodateien
 - AVI 605, 617
 - MPEG 605, 617
- Videos
 - über DirectX absp. 617
 - über MCI absp. 605
- Visual Studio, Steuerelem. integr. 506

W

- W3Schools 360
- WaitForExit 240, 242
- WaitForStatus 233
- Wave-Dateien
 - über API absp. 603
 - über DirectX absp. 617
 - über MCI absp. 605
- WebRequest 481
- WebResponse 481
- Webseiten
 - in Anwendung 474
 - in Browser 473
- Websites, zu C# und .NET 20
- Win32_OperatingSystem 414, 429
- Win32_ParallelPort 419
- Win32_Processor 413
- Win32_Product 417
- Win32_SerialPort 424
- Win32_SoundDevice 427
- Windows
 - booten 429
 - Systempfade 409
 - Version auslesen 411
- WindowsIdentity 578
- WindowsImpersonationContext 578
- Windows-Nachrichten 537
- Windows-Ordner ermitteln 253

WindowState 239
WinNT-ADSI-Provider 547, 556, 562
WMI 55
 CPU-Geschwindigkeit 413
 Datumswerte konv. 64
 Installation überprüfen 56
 Installierte Progr. listen 417
 Klassenpfad 57
 Methoden aufrufen 66
 Namensräume 57
 Parallele Schnittst. listen 419
 RAM-Infos 414
WMI Object Browser 57
WndProc 489
Wörter
 Anzahl in String 114
 aus einem String 116
 erstes Zeichen groß 111
Word-Dokumente
 erzeugen 660
 öffnen 657
Workbooks 664
Worksheets 665
WQL 61
Write 328
WriteAttributeString 390
WriteBase64 390, 404
WriteChars 390
WriteElementString 390
WriteEndDocument 390
WriteEndElement 390
WriteStartDocument 389
WriteStartElement 390
WriteXml 401
Wurzel berechnen 714

X

XML-Dateien siehe XML-Dokumente
XmlDocument 371, 377, 392
 Referenz 723
 XML validieren 365
XML-Dokumente
 ändern 392, 403
 Binäre Daten 333, 404
 binäre Daten lesen 406
 Elemente gezielt lesen 377
 Gültigkeit überprüfen 365
 transformieren 407
 über das DOM lesen 371

 über DataSet schreiben 401
 über ein DataSet lesen 385
 über XmlDocument schreiben 392
 über XmlTextReader lesen 355
 über XmlTextWriter erzeugen 389
 validierend lesen 360, 375
XmlNamespaceManager 382
XmlNode 372, 392
 Referenz 725
XmlNodeList 372
XML-Schemata 360, 386
XmlSerializer 684
XmlTextReader 355
XmlTextWriter 389
XmlValidatingReader 361, 365f, 375
XPath 377
XPath-Suche
 mit Namensraum 382
 ohne Namensraum 377
XSL 407
XslTransform 408

Y

Year 712

Z

Zahlen
 am Stringanfang? 122
 aus String auslesen 121
 dual darstellen 84
 Gültigkeit 94
 hexadezimal darst. 83
 High- u. Low-Word 101f
 in TextBox 506
 kaufmännisch runden 89
 länderspezifisch 92
 Notationen 88
 von Dual 88
 Zufallszahlen 98
Zahlformatierungen 702
Zeichenketten siehe Strings
Zeichenklassen 720
Zeichnen
 Graustufen 645
 Negativ 643
 Schräg 649
 Transformieren 637
 Transparent 646

Zeit

genau messen 189

Systemzeit setzen 164

ZIP-Archive erzeugen 337

ZIP-Dateien siehe Archive

ZipEntry 340

ZipOutputStream 339

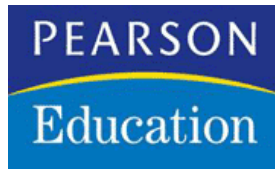
Zufalls-String 123

Zufallszahlen 98

Zuweisungsoperatoren 703

Zwischenablage

Bitmaps lesen 621



Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt.

Dieses eBook stellen wir lediglich als **Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschliesslich der Reproduktion, der Weitergabe, des Weitervertriebs, der Platzierung im Internet, in Intranets, in Extranets anderen Websites, der Veränderung, des Weiterverkaufs und der Veröffentlichung bedarf der schriftlichen Genehmigung des Verlags.

Bei Fragen zu diesem Thema wenden Sie sich bitte an:

<mailto:info@pearson.de>

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf der Website ist eine freiwillige Leistung des Verlags. Der Rechtsweg ist ausgeschlossen.

Hinweis

Dieses und andere eBooks können Sie rund um die Uhr und legal auf unserer Website



(<http://www.informit.de>)

herunterladen